

---

# **Linux Admin-guide Documentation**

**The kernel development community**

**Jul 14, 2020**



## **CONTENTS**



The following is a collection of user-oriented documents that have been added to the kernel over time. There is, as yet, little overall order or organization here — this material was not written to be a single, coherent document! With luck things will improve quickly over time.

This initial section contains overall information, including the README file describing the kernel as a whole, documentation on kernel parameters, etc.



## LINUX KERNEL RELEASE 5.X <[HTTP://KERNEL.ORG/](http://kernel.org/)>

These are the release notes for Linux version 5. Read them carefully, as they tell you what this is all about, explain how to install the kernel, and what to do if something goes wrong.

### 1.1 What is Linux?

Linux is a clone of the operating system Unix, written from scratch by Linus Torvalds with assistance from a loosely-knit team of hackers across the Net. It aims towards POSIX and Single UNIX Specification compliance.

It has all the features you would expect in a modern fully-fledged Unix, including true multitasking, virtual memory, shared libraries, demand loading, shared copy-on-write executables, proper memory management, and multistack networking including IPv4 and IPv6.

It is distributed under the GNU General Public License v2 - see the accompanying COPYING file for more details.

### 1.2 On what hardware does it run?

Although originally developed first for 32-bit x86-based PCs (386 or higher), today Linux also runs on (at least) the Compaq Alpha AXP, Sun SPARC and UltraSPARC, Motorola 68000, PowerPC, PowerPC64, ARM, Hitachi SuperH, Cell, IBM S/390, MIPS, HP PA-RISC, Intel IA-64, DEC VAX, AMD x86-64 Xtensa, and ARC architectures.

Linux is easily portable to most general-purpose 32- or 64-bit architectures as long as they have a paged memory management unit (PMMU) and a port of the GNU C compiler (gcc) (part of The GNU Compiler Collection, GCC). Linux has also been ported to a number of architectures without a PMMU, although functionality is then obviously somewhat limited. Linux has also been ported to itself. You can now run the kernel as a userspace application - this is called UserMode Linux (UML).

### 1.3 Documentation

- There is a lot of documentation available both in electronic form on the Internet and in books, both Linux-specific and pertaining to general UNIX questions. I'd recommend looking into the documentation subdirectories on any Linux FTP site for the LDP (Linux Documentation Project) books. This README is not meant to be documentation on the system: there are much better sources available.
- There are various README files in the Documentation/ subdirectory: these typically contain kernel-specific installation notes for some drivers for example. Please read the Documentation/process/changes.rst file, as it contains information about the problems, which may result by upgrading your kernel.

### 1.4 Installing the kernel source

- If you install the full sources, put the kernel tarball in a directory where you have permissions (e.g. your home directory) and unpack it:

```
xz -cd linux-5.x.tar.xz | tar xvf -
```

Replace "X" with the version number of the latest kernel.

Do NOT use the /usr/src/linux area! This area has a (usually incomplete) set of kernel headers that are used by the library header files. They should match the library, and not get messed up by whatever the kernel-du-jour happens to be.

- You can also upgrade between 5.x releases by patching. Patches are distributed in the xz format. To install by patching, get all the newer patch files, enter the top level directory of the kernel source (linux-5.x) and execute:

```
xz -cd ../patch-5.x.xz | patch -p1
```

Replace "x" for all versions bigger than the version "x" of your current source tree, **in\_order**, and you should be ok. You may want to remove the backup files (some-file-name~ or some-file-name.orig), and make sure that there are no failed patches (some-file-name# or some-file-name.rej). If there are, either you or I have made a mistake.

Unlike patches for the 5.x kernels, patches for the 5.x.y kernels (also known as the -stable kernels) are not incremental but instead apply directly to the base 5.x kernel. For example, if your base kernel is 5.0 and you want to apply the 5.0.3 patch, you must not first apply the 5.0.1 and 5.0.2 patches. Similarly, if you are running kernel version 5.0.2 and want to jump to 5.0.3, you must first reverse the 5.0.2 patch (that is, patch -R) **before** applying the 5.0.3 patch. You can read more on this in Documentation/process/applying-patches.rst.

Alternatively, the script patch-kernel can be used to automate this process. It determines the current kernel version and applies any patches found:

```
linux/scripts/patch-kernel linux
```

The first argument in the command above is the location of the kernel source. Patches are applied from the current directory, but an alternative directory can be specified as the second argument.

- Make sure you have no stale .o files and dependencies lying around:

```
cd linux
make mrproper
```

You should now have the sources correctly installed.

## 1.5 Software requirements

Compiling and running the 5.x kernels requires up-to-date versions of various software packages. Consult Documentation/process/changes.rst for the minimum version numbers required and how to get updates for these packages. Beware that using excessively old versions of these packages can cause indirect errors that are very difficult to track down, so don't assume that you can just update packages when obvious problems arise during build or operation.

## 1.6 Build directory for the kernel

When compiling the kernel, all output files will per default be stored together with the kernel source code. Using the option `make O=output/dir` allows you to specify an alternate place for the output files (including `.config`). Example:

```
kernel source code: /usr/src/linux-5.x
build directory:    /home/name/build/kernel
```

To configure and build the kernel, use:

```
cd /usr/src/linux-5.x
make O=/home/name/build/kernel menuconfig
make O=/home/name/build/kernel
sudo make O=/home/name/build/kernel modules_install install
```

Please note: If the `O=output/dir` option is used, then it must be used for all invocations of `make`.

## 1.7 Configuring the kernel

Do not skip this step even if you are only upgrading one minor version. New configuration options are added in each release, and odd problems will turn up if the configuration files are not set up as expected. If you want to carry your existing configuration to a new version with minimal work, use `make oldconfig`, which will only ask you for the answers to new questions.

- Alternative configuration commands are:

"make config"	Plain text interface.
"make menuconfig" ↪ dialogs.	Text based color menus, radiolists & ↪
"make nconfig"	Enhanced text based color menus.
"make xconfig"	Qt based configuration tool.
"make gconfig"	GTK+ based configuration tool.
"make oldconfig" ↪ contents of ↪ about	Default all questions based on the ↪ your existing <code>./.config</code> file and asking ↪ new config symbols.
"make olddefconfig" ↪ default	Like above, but sets new symbols to their ↪ values without prompting.
"make defconfig" ↪ default ↪ defconfig ↪ defconfig,	Create a <code>./.config</code> file by using the ↪ symbol values from either <code>arch/\$ARCH/ or <code>arch/\$ARCH/configs/\${PLATFORM}_</code> depending on the architecture.</code>
"make \${PLATFORM}_defconfig" ↪ default ↪ available	Create a <code>./.config</code> file by using the ↪ symbol values from <code>arch/\$ARCH/configs/\${PLATFORM}_defconfig</code> . Use "make help" to get a list of all ↪ platforms of your architecture.
"make allyesconfig"	Create a <code>./.config</code> file by setting symbol values to 'y' as much as possible.
"make allmodconfig"	Create a <code>./.config</code> file by setting symbol

(continues on next page)

(continued from previous page)

	values to 'm' as much as possible.
"make allnoconfig"	Create a <code>./config</code> file by setting symbol values to 'n' as much as possible.
"make randconfig"	Create a <code>./config</code> file by setting symbol values to random values.
"make localmodconfig"	Create a config based on current config and loaded modules (lsmod). Disables any option that is not needed for the loaded modules.
↪ machine, file	To create a localmodconfig for another machine, store the lsmod of that machine into a file and pass it in as a LSMOD parameter.
↪ certain folders	Also, you can preserve modules in or kconfig files by specifying their paths in parameter LMC_KEEP.
	<pre>target\$ lsmod &gt; /tmp/mylsmod target\$ scp /tmp/mylsmod host:/tmp  host\$ make LSMOD=/tmp/mylsmod \     LMC_KEEP="drivers/usb:drivers/gpu:fs" \     localmodconfig</pre>
↪ compiling.	The above also works when cross-compiling.
"make localyesconfig"	Similar to localmodconfig, except it will convert all module options to built in (=y) options. You can also preserve modules by LMC_KEEP.
"make kvmconfig"	Enable additional options for kvm guest kernel support.
"make xenconfig"	Enable additional options for xen dom0 guest kernel support.
"make tinyconfig"	Configure the tiniest possible kernel.

You can find more information on using the Linux kernel config tools in `Documentation/kbuild/kconfig.rst`.

- NOTES on make config:

- Having unnecessary drivers will make the kernel bigger, and can under some circumstances lead to problems: probing for a nonexistent controller card may confuse your other controllers.
- A kernel with math-emulation compiled in will still use the co-processor if one is present: the math emulation will just never get used in that case. The kernel will be slightly larger, but will work on different machines regardless of whether they have a math coprocessor or not.
- The “kernel hacking” configuration details usually result in a bigger or slower kernel (or both), and can even make the kernel less stable by configuring some routines to actively try to break bad code to find kernel problems (kmallocc()). Thus you should probably answer ‘n’ to the questions for “development” , “experimental” , or “debugging” features.

## 1.8 Compiling the kernel

- Make sure you have at least gcc 4.9 available. For more information, refer to Documentation/process/changes.rst.

Please note that you can still run a.out user programs with this kernel.

- Do a make to create a compressed kernel image. It is also possible to do make install if you have lilo installed to suit the kernel makefiles, but you may want to check your particular lilo setup first.

To do the actual install, you have to be root, but none of the normal build should require that. Don’ t take the name of root in vain.

- If you configured any of the parts of the kernel as modules, you will also have to do make modules\_install.
- Verbose kernel compile/build output:

Normally, the kernel build system runs in a fairly quiet mode (but not totally silent). However, sometimes you or other kernel developers need to see compile, link, or other commands exactly as they are executed. For this, use “verbose” build mode. This is done by passing V=1 to the make command, e.g.:

```
make V=1 all
```

To have the build system also tell the reason for the rebuild of each target, use V=2. The default is V=0.

- Keep a backup kernel handy in case something goes wrong. This is especially true for the development releases, since each new release contains new code which has not been debugged. Make sure you keep a backup of the modules corresponding to that kernel, as well. If you are installing a new kernel with the same version number as your working kernel, make a backup of your modules directory before you do a make modules\_install.

Alternatively, before compiling, use the kernel config option “LOCALVERSION” to append a unique suffix to the regular kernel version. LOCALVER-

SION can be set in the “General Setup” menu.

- In order to boot your new kernel, you’ ll need to copy the kernel image (e.g. `.../linux/arch/x86/boot/bzImage` after compilation) to the place where your regular bootable kernel is found.
- Booting a kernel directly from a floppy without the assistance of a bootloader such as LILO, is no longer supported.

If you boot Linux from the hard drive, chances are you use LILO, which uses the kernel image as specified in the file `/etc/lilo.conf`. The kernel image file is usually `/vmlinuz`, `/boot/vmlinuz`, `/bzImage` or `/boot/bzImage`. To use the new kernel, save a copy of the old image and copy the new image over the old one. Then, you MUST RERUN LILO to update the loading map! If you don’ t, you won’ t be able to boot the new kernel image.

Reinstalling LILO is usually a matter of running `/sbin/lilo`. You may wish to edit `/etc/lilo.conf` to specify an entry for your old kernel image (say, `/vmlinuz.old`) in case the new one does not work. See the LILO docs for more information.

After reinstalling LILO, you should be all set. Shutdown the system, reboot, and enjoy!

If you ever need to change the default root device, video mode, ramdisk size, etc. in the kernel image, use the `rdev` program (or alternatively the LILO boot options when appropriate). No need to recompile the kernel to change these parameters.

- Reboot with the new kernel and enjoy.

## 1.9 If something goes wrong

- If you have problems that seem to be due to kernel bugs, please check the file `MAINTAINERS` to see if there is a particular person associated with the part of the kernel that you are having trouble with. If there isn’ t anyone listed there, then the second best thing is to mail them to me ([torvalds@linux-foundation.org](mailto:torvalds@linux-foundation.org)), and possibly to any other relevant mailing-list or to the news-group.
- In all bug-reports, please tell what kernel you are talking about, how to duplicate the problem, and what your setup is (use your common sense). If the problem is new, tell me so, and if the problem is old, please try to tell me when you first noticed it.
- If the bug results in a message like:

```
unable to handle kernel paging request at address C0000010
Oops: 0002
EIP: 0010:XXXXXXXX
eax: xxxxxxxx ebx: xxxxxxxx ecx: xxxxxxxx edx: xxxxxxxx
esi: xxxxxxxx edi: xxxxxxxx ebp: xxxxxxxx
ds: xxxx es: xxxx fs: xxxx gs: xxxx
Pid: xx, process nr: xx
xx xx xx xx xx xx xx xx xx xx
```

or similar kernel debugging information on your screen or in your system log, please duplicate it exactly. The dump may look incomprehensible to you, but it does contain information that may help debugging the problem. The text above the dump is also important: it tells something about why the kernel dumped code (in the above example, it's due to a bad kernel pointer). More information on making sense of the dump is in Documentation/admin-guide/bug-hunting.rst

- If you compiled the kernel with `CONFIG_KALLSYMS` you can send the dump as is, otherwise you will have to use the `ksymoops` program to make sense of the dump (but compiling with `CONFIG_KALLSYMS` is usually preferred). This utility can be downloaded from <https://www.kernel.org/pub/linux/utils/kernel/ksymoops/>. Alternatively, you can do the dump lookup by hand:
- In debugging dumps like the above, it helps enormously if you can look up what the EIP value means. The hex value as such doesn't help me or anybody else very much: it will depend on your particular kernel setup. What you should do is take the hex value from the EIP line (ignore the `0010:`), and look it up in the kernel namelist to see which kernel function contains the offending address.

To find out the kernel function name, you'll need to find the system binary associated with the kernel that exhibited the symptom. This is the file `'linux/vmlinux'`. To extract the namelist and match it against the EIP from the kernel crash, do:

```
nm vmlinux | sort | less
```

This will give you a list of kernel addresses sorted in ascending order, from which it is simple to find the function that contains the offending address. Note that the address given by the kernel debugging messages will not necessarily match exactly with the function addresses (in fact, that is very unlikely), so you can't just `'grep'` the list: the list will, however, give you the starting point of each kernel function, so by looking for the function that has a starting address lower than the one you are searching for but is followed by a function with a higher address you will find the one you want. In fact, it may be a good idea to include a bit of "context" in your problem report, giving a few lines around the interesting one.

If you for some reason cannot do the above (you have a pre-compiled kernel image or similar), telling me as much about your setup as possible will help. Please read the `admin-guide/reporting-bugs.rst` document for details.

- Alternatively, you can use `gdb` on a running kernel. (read-only; i.e. you cannot change values or set break points.) To do this, first compile the kernel with `-g`; edit `arch/x86/Makefile` appropriately, then do a `make clean`. You'll also need to enable `CONFIG_PROC_FS` (via `make config`).

After you've rebooted with the new kernel, do `gdb vmlinux /proc/kcore`. You can now use all the usual `gdb` commands. The command to look up the point where your system crashed is `l *0xXXXXXXXX`. (Replace the XXXes with the EIP value.)

`gdb`'ing a non-running kernel currently fails because `gdb` (wrongly) disregards the starting offset for which the kernel is compiled.

## THE KERNEL' S COMMAND-LINE PARAMETERS

The following is a consolidated list of the kernel parameters as implemented by the `__setup()`, `core_param()` and `module_param()` macros and sorted into English Dictionary order (defined as ignoring all punctuation and sorting digits before letters in a case insensitive manner), and with descriptions where known.

The kernel parses parameters from the kernel command line up to “- - “; if it doesn't recognize a parameter and it doesn't contain a ‘.’, the parameter gets passed to `init`: parameters with ‘=’ go into `init`'s environment, others are passed as command line arguments to `init`. Everything after “- -” is passed as an argument to `init`.

Module parameters can be specified in two ways: via the kernel command line with a module name prefix, or via `modprobe`, e.g.:

```
(kernel command line) usbcore.blinkenlights=1
(modprobe command line) modprobe usbcore blinkenlights=1
```

Parameters for modules which are built into the kernel need to be specified on the kernel command line. `modprobe` looks through the kernel command line (`/proc/cmdline`) and collects module parameters when it loads a module, so the kernel command line can be used for loadable modules too.

Hyphens (dashes) and underscores are equivalent in parameter names, so:

```
log_buf_len=1M print_fatal_signals=1
```

can also be entered as:

```
log-buf-len=1M print_fatal_signals=1
```

Double-quotes can be used to protect spaces in values, e.g.:

```
param="spaces in here"
```

## 2.1 cpu lists:

Some kernel parameters take a list of CPUs as a value, e.g. `isolcpus`, `nohz_full`, `irqaffinity`, `rcu_nocbs`. The format of this list is:

`<cpu number>, ..., <cpu number>`

or

`<cpu number>-<cpu number>` (must be a positive range in ascending order)

or a mixture

`<cpu number>, ..., <cpu number>-<cpu number>`

Note that for the special case of a range one can split the range into equal sized groups and for each group use some amount from the beginning of that group:

`<cpu number>-cpu number>:<used size>/<group size>`

For example one can add to the command line following parameter:

`isolcpus=1,2,10-20,100-2000:2/25`

where the final item represents CPUs 100,101,125,126,150,151,...

This document may not be entirely up to date and comprehensive. The command “`modinfo -p ${modulename}`” shows a current list of all parameters of a loadable module. Loadable modules, after being loaded into the running kernel, also reveal their parameters in `/sys/module/${modulename}/parameters/`. Some of these parameters may be changed at runtime by the command `echo -n ${value} > /sys/module/${modulename}/parameters/${parm}`.

The parameters listed below are only valid if certain kernel build options were enabled and if respective hardware is present. The text in square brackets at the beginning of each description states the restrictions within which a parameter is applicable:

ACPI	ACPI support is enabled.
AGP	AGP (Accelerated Graphics Port) is enabled.
ALSA	ALSA sound support is enabled.
APIC	APIC support is enabled.
APM	Advanced Power Management support is enabled.
ARM	ARM architecture is enabled.
ARM64	ARM64 architecture is enabled.
AX25	Appropriate AX.25 support is enabled.
CLK	Common clock infrastructure is enabled.
CMA	Contiguous Memory Area support is enabled.
DRM	Direct Rendering Management support is enabled.
DYNAMIC_DEBUG	Build in debug messages and enable them at runtime
EDD	BIOS Enhanced Disk Drive Services (EDD) is enabled
EFI	EFI Partitioning (GPT) is enabled
EIDE	EIDE/ATAPI support is enabled.
EVM	Extended Verification Module
FB	The frame buffer device is enabled.
FTRACE	Function tracing enabled.
GCOV	GCOV profiling is enabled.

(continues on next page)

(continued from previous page)

HW	Appropriate hardware is enabled.
IA-64	IA-64 architecture is enabled.
IMA	Integrity measurement architecture is enabled.
IOSCHED	More than one I/O scheduler is enabled.
IP_PNP	IP DHCP, BOOTP, or RARP is enabled.
IPV6	IPv6 support is enabled.
ISAPNP	ISA PnP code is enabled.
ISDN	Appropriate ISDN support is enabled.
ISOL	CPU Isolation is enabled.
JOY	Appropriate joystick support is enabled.
KGDB	Kernel debugger support is enabled.
KVM	Kernel Virtual Machine support is enabled.
LIBATA	Libata driver is enabled
LP	Printer support is enabled.
LOOP	Loopback device support is enabled.
M68k	M68k architecture is enabled. These options have more detailed description inside of Documentation/m68k/kernel-options.rst.
MDA	MDA console support is enabled.
MIPS	MIPS architecture is enabled.
MOUSE	Appropriate mouse support is enabled.
MSI	Message Signaled Interrupts (PCI).
MTD	MTD (Memory Technology Device) support is enabled.
NET	Appropriate network support is enabled.
NUMA	NUMA support is enabled.
NFS	Appropriate NFS support is enabled.
OF	Devicetree is enabled.
OSS	OSS sound support is enabled.
PV_OPS	A paravirtualized kernel is enabled.
PARIDE	The ParIDE (parallel port IDE) subsystem is enabled.
PARISC	The PA-RISC architecture is enabled.
PCI	PCI bus support is enabled.
PCIE	PCI Express support is enabled.
PCMCIA	The PCMCIA subsystem is enabled.
PNP	Plug & Play support is enabled.
PPC	PowerPC architecture is enabled.
PPT	Parallel port support is enabled.
PS2	Appropriate PS/2 support is enabled.
RAM	RAM disk support is enabled.
RDT	Intel Resource Director Technology.
S390	S390 architecture is enabled.
SCSI	Appropriate SCSI support is enabled. A lot of drivers have their options described inside the Documentation/scsi/ sub-directory.
SECURITY	Different security models are enabled.
SELINUX	SELinux support is enabled.
APPARMOR	AppArmor support is enabled.
SERIAL	Serial support is enabled.
SH	SuperH architecture is enabled.
SMP	The kernel is an SMP kernel.
SPARC	Sparc architecture is enabled.
SWSUSP	Software suspend (hibernation) is enabled.
SUSPEND	System suspend states are enabled.
TPM	TPM drivers are enabled.
TS	Appropriate touchscreen support is enabled.
UMS	USB Mass Storage support is enabled.

(continues on next page)

(continued from previous page)

USB	USB support is enabled.
USBHID	USB Human Interface Device support is enabled.
V4L	Video For Linux support is enabled.
VMMIO	Driver for memory mapped virtio devices is enabled.
VGA	The VGA console has been enabled.
VT	Virtual terminal support is enabled.
WDT	Watchdog support is enabled.
XT	IBM PC/XT MFM hard disk support is enabled.
X86-32	X86-32, aka i386 architecture is enabled.
X86-64	X86-64 architecture is enabled. More X86-64 boot options can be found in Documentation/x86/x86_64/boot-options.rst.
X86	Either 32-bit or 64-bit x86 (same as X86-32+X86-64)
X86_UV	SGI UV support is enabled.
XEN	Xen support is enabled

In addition, the following text indicates that the option:

BUGS=	Relates to possible processor bugs on the said processor.
KNL	Is a kernel start-up parameter.
BOOT	Is a boot loader parameter.

Parameters denoted with BOOT are actually interpreted by the boot loader, and have no meaning to the kernel directly. Do not modify the syntax of boot loader parameters without extreme need or coordination with <Documentation/x86/boot.rst>.

There are also arch-specific kernel-parameters not documented here. See for example <Documentation/x86/x86\_64/boot-options.rst>.

Note that ALL kernel parameters listed below are CASE SENSITIVE, and that a trailing = on the name of any parameter states that that parameter will be entered as an environment variable, whereas its absence indicates that it will appear as a kernel argument readable via /proc/cmdline by programs running once the system is up.

The number of kernel parameters is not limited, but the length of the complete command line (parameters including spaces etc.) is limited to a fixed number of characters. This limit depends on the architecture and is between 256 and 4096 characters. It is defined in the file ./include/asm/setup.h as COMMAND\_LINE\_SIZE.

Finally, the [KMG] suffix is commonly described after a number of kernel parameter values. These 'K', 'M', and 'G' letters represent the `_binary_multipliers` 'Kilo', 'Mega', and 'Giga', equaling  $2^{10}$ ,  $2^{20}$ , and  $2^{30}$  bytes respectively. Such letter suffixes can also be entirely omitted:

```
acpi= [HW,ACPI,X86,ARM64]
      Advanced Configuration and Power Interface
      Format: { force | on | off | strict | noirq
→ | rsdt |
           copy_dsdt }
      force -- enable ACPI if default was off
      on -- enable ACPI but allow fallback to DT
→ [arm64]
```

off -- disable ACPI if default was on  
 noirq -- do not use ACPI for IRQ routing  
 strict -- Be less tolerant of platforms

→ that are not strictly ACPI specification

→ compliant.

rsdt -- prefer RSDT over (default) XSDT  
 copy\_dsdt -- copy DSDT to memory  
 For ARM64, ONLY "acpi=off", "acpi=on" or

→ "acpi=force" are available

See also Documentation/power/runtime\_pm.rst,

→ pci=noacpi

```

acpi_apic_instance=    [ACPI, IOAPIC]
                       Format: <int>
                       2: use 2nd APIC table, if available
                       1,0: use 1st APIC table
                       default: 0
  
```

acpi\_backlight= [HW,ACPI]
 { vendor | video | native | none }
 If set to vendor, prefer vendor-specific

→ driver (e.g. thinkpad\_acpi, sony\_acpi, etc.)

→ instead of the ACPI video.ko driver.

→ driver. If set to video, use the ACPI video.ko

→ backlight mode. If set to native, use the device's native

→ interface. If set to none, disable the ACPI backlight

```

acpi_force_32bit_fadt_addr
                       force FADT to use 32 bit addresses rather
  
```

→ than the 64 bit X\_\* addresses. Some firmware have

→ broken 64 bit addresses for force ACPI ignore these

→ and use the older legacy 32 bit addresses.

```

acpica_no_return_repair [HW, ACPI]
                       Disable AML predefined validation mechanism
                       This mechanism can repair the evaluation
  
```

→ result to make the return objects more ACPI specification

→ compliant. This option is useful for developers to

→ identify the root cause of an AML interpreter issue when  
→ the issue has something to do with the repair  
→ mechanism.

```
acpi.debug_layer= [HW,ACPI,ACPI_DEBUG]
acpi.debug_level= [HW,ACPI,ACPI_DEBUG]
```

Format: <int>  
CONFIG\_ACPI\_DEBUG must be enabled to

→ produce any ACPI debug output. Bits in debug\_layer  
→ correspond to a \_COMPONENT in an ACPI source file, e.g.,  
#define \_COMPONENT ACPI\_PCI\_COMPONENT  
Bits in debug\_level correspond to a level in  
ACPI\_DEBUG\_PRINT statements, e.g.,  
ACPI\_DEBUG\_PRINT((ACPI\_DB\_INFO, ...  
The debug\_level mask defaults to "info".

→ See Documentation/firmware-guide/acpi/debug.rst  
→ for more information about debug layers and levels.

```
Enable processor driver info messages:
acpi.debug_layer=0x20000000
Enable PCI/PCI interrupt routing info
```

→ messages:  
acpi.debug\_layer=0x400000  
Enable AML "Debug" output, i.e., stores to  
→ the Debug object while interpreting AML:

```
acpi.debug_layer=0xffffffff acpi.debug_
→ level=0x2
Enable all messages related to ACPI
```

```
→ hardware:
acpi.debug_layer=0x2 acpi.debug_
→ level=0xffffffff
```

Some values produce so much output that the  
→ system is unusable. The "log\_buf\_len" parameter may  
→ be useful if you need to capture more output.

```
acpi_enforce_resources= [ACPI]
{ strict | lax | no }
Check for resource conflicts between native
```

→ drivers and ACPI OperationRegions (SystemIO and  
→ SystemMemory

only). IO ports and memory declared in ACPI  
 might be used by the ACPI subsystem in arbitrary AML  
 code and can interfere with legacy drivers.  
 strict (default): access to resources  
 claimed by ACPI is denied; legacy drivers trying to access  
 reserved resources will fail to bind to device using  
 them.  
 lax: access to resources claimed by ACPI is  
 allowed; legacy drivers trying to access reserved  
 resources will bind successfully but a warning  
 message is logged.  
 reserved, no: ACPI OperationRegions are not marked as  
 no further checks are performed.

`acpi_force_table_verification` [HW,ACPI]  
 Enable table checksum verification during  
 early stage.  
 By default, this is disabled due to x86  
 early mapping size limitation.

`acpi_irq_balance` [HW,ACPI]  
 ACPI will balance active IRQs  
 default in APIC mode

`acpi_irq_nobalance` [HW,ACPI]  
 ACPI will not move active IRQs (default)  
 default in PIC mode

`acpi_irq_isa=` [HW,ACPI] If `irq_balance`, mark listed IRQs  
 used by ISA  
 Format: `<irq>,<irq>...`

`acpi_irq_pci=` [HW,ACPI] If `irq_balance`, clear listed IRQs  
 for use by PCI  
 Format: `<irq>,<irq>...`

`acpi_mask_gpe=` [HW,ACPI]  
 Due to the existence of `_Lxx/_Exx`, some  
 GPEs triggered by unsupported hardware/firmware features  
 can result in GPE floodings that cannot be automatically

↳disabled by the GPE dispatcher.  
This facility can be used to prevent such

↳uncontrolled GPE floodings.  
Format: <byte>

`acpi_no_auto_serialize` [HW,ACPI]  
Disable auto-serialization of AML methods  
AML control methods that contain the

↳opcodes to create named objects will be marked as

↳"Serialized" by the auto-serialization feature.  
This feature is enabled by default.  
This option allows to turn off the feature.

`acpi_no_memhotplug` [ACPI] Disable memory hotplug. Useful

↳for kdump kernels.

`acpi_no_static_ssdt` [HW,ACPI]  
Disable installation of static SSDTs at

↳early boot time By default, SSDTs contained in the RSDT/  
↳XSDT will be installed automatically and they will

↳appear under `/sys/firmware/acpi/tables`.  
This option turns off this feature.  
Note that specifying this option does not

↳affect dynamic table installation which will

↳install SSDT tables to `/sys/firmware/acpi/tables/dynamic`.

`acpi_no_watchdog` [HW,ACPI,WDT]  
Ignore the ACPI-based watchdog interface

↳(WDAT) and let a native driver control the watchdog device

↳instead.

`acpi_rsdp=` [ACPI,EFI,KEXEC]  
Pass the RSDP address to the kernel, mostly

↳used on machines running EFI runtime service to

↳boot the second kernel for kdump.

`acpi_os_name=` [HW,ACPI] Tell ACPI BIOS the name of the OS  
Format: To spoof as Windows 98: ="Microsoft

→Windows"

→acpi\_rev\_override [ACPI] Override the \_REV object to return  
 →5 (instead of 2 which is mandated by ACPI 6) as the  
 →supported ACPI specification revision (when using this  
 →switch, it may be necessary to carry out a cold reboot \_  
 →twice\_ in a row to make it take effect on the platform  
 →firmware).

→acpi\_osi= [HW,ACPI] Modify list of supported OS  
 →interface strings

```
acpi_osi="string1"      # add string1
acpi_osi="!string2"   # remove string2
acpi_osi=!*           # remove all strings
acpi_osi=!            # disable all
                        strings
acpi_osi=!!           # enable all
                        strings
acpi_osi=              # disable all
                        strings
```

→built-in OS vendor

→built-in OS vendor

→strings

→with single or

→specific OS

→can only

→strings, thus

→feature group

→vendor strings,

→command line

→when one do not

→strings which

→is equivalent

→', they all

'acpi\_osi=!' can be used in combination  
 multiple 'acpi\_osi="string1"' to support  
 vendor string(s). Note that such command  
 affect the default state of the OS vendor  
 it cannot affect the default state of the  
 strings and the current state of the OS  
 specifying it multiple times through kernel  
 is meaningless. This command is useful  
 care about the state of the feature group  
 should be controlled by the OSPM.

Examples:

1. 'acpi\_osi=! acpi\_osi="Windows 2000"'

to 'acpi\_osi="Windows 2000" acpi\_osi=!

can make `'_OSI("Windows 2000")'` TRUE.

↳with other  
↳will not  
↳such command can  
↳specifying it  
↳is also

`'acpi_osi='` cannot be used in combination  
`'acpi_osi='` command lines, the `_OSI` method  
exist in the ACPI namespace. NOTE that  
only affect the `_OSI` support state, thus  
multiple times through kernel command line  
meaningless.

Examples:  
1. `'acpi_osi='` can make `'CondRefOf(_OSI, Local1)'`  
FALSE.

↳with single or  
↳specific  
↳affect the  
↳and the  
↳multiple times  
↳But it may  
↳a string if  
↳This command  
↳state of the  
↳related to

`'acpi_osi=!*'` can be used in combination  
multiple `'acpi_osi="string1"'` to support  
string(s). Note that such command can  
current state of both the OS vendor strings  
feature group strings, thus specifying it  
through kernel command line is meaningful.   
still not able to affect the final state of  
there are quirks related to this string.   
is useful when one want to control the  
feature group strings to debug BIOS issues  
the OSPM features.

Examples:  
1. `'acpi_osi="Module Device" acpi_osi=!*'`  
`'_OSI("Module Device")'` FALSE.  
2. `'acpi_osi=!* acpi_osi="Module Device"'`  
`'_OSI("Module Device")'` TRUE.  
3. `'acpi_osi=! acpi_osi=!* acpi_`  
equivalent to  
`'acpi_osi=!* acpi_osi=! acpi_`

↳can make  
↳can make  
↳`osi="Windows 2000"` is  
↳`osi="Windows 2000"`

```

and
'acpi_osi=!* acpi_osi="Windows 2000"
acpi_osi=!',
they all will make '_OSI("Windows
2000")' TRUE.

acpi_pm_good [X86]
Override the pmtimer bug detection: force
the kernel
to assume that this machine's pmtimer
latches its value
and always returns good values.

acpi_sci= [HW,ACPI] ACPI System Control Interrupt
trigger mode
Format: { level | edge | high | low }

acpi_skip_timer_override [HW,ACPI]
Recognize and ignore IRQ0/pin2 Interrupt
Override.
For broken nForce2 BIOS resulting in XT-PIC
timer.

acpi_sleep= [HW,ACPI] Sleep options
Format: { s3_bios, s3_mode, s3_beeper, s4_
nohwsig,
old_ordering, nonvs, sci_force_
enable, nobl }
See Documentation/power/video.rst for
information on
s3_bios and s3_mode.
s3_beeper is for debugging; it makes the PC's
speaker beep
as soon as the kernel's real-mode entry
point is called.
s4_nohwsig prevents ACPI hardware signature
from being
used during resume from hibernation.
old_ordering causes the ACPI 1.0 ordering
of the _PTS
control method, with respect to putting
devices into
low power states, to be enforced (the ACPI
2.0 ordering
of _PTS is used by default).
nonvs prevents the kernel from saving/
restoring the
ACPI NVS memory during suspend/hibernation.
and resume.
sci_force_enable causes the kernel to set
SCI_EN directly

```

↪ACPI spec,  
↪it).  
↪systems known to  
↪respect to system  
↪wisely).

        acpi\_use\_timer\_override [HW,ACPI]  
↪NF5 boards  
↪have HPET

        Use timer override. For some broken Nvidia,  
that require a timer override, but don't

        add\_efi\_memmap [EFI; X86] Include EFI memory map in  
kernel's map of available physical RAM.

        agp= [AGP]  
        { off | try\_unsupported }  
↪chipsets  
↪corruption)

        off: disable AGP support  
        try\_unsupported: try to drive unsupported  
        (may crash computer or cause data

        ALSA [HW,ALSA]  
↪rst

        See Documentation/sound/alsa-configuration.

        alignment= [KNL,ARM]  
↪handler  
↪warnings,  
↪segfault.

        Allow the default userspace alignment fault  
behaviour to be specified. Bit 0 enables  
bit 1 enables fixups, and bit 2 sends a

        align\_va\_addr= [X86-64]  
↪[14:12] when  
↪This option  
↪on AMD F15h  
↪for a  
↪vary highly in

        Align virtual addresses by clearing slice  
allocating a VMA at process creation time.  
gives you up to 3% performance improvement  
machines (where it is enabled by default)  
CPU-intensive style benchmark, and it can

a microbenchmark depending on workload and `u`  
`→compiler.`

`32:` only for 32-bit processes  
`64:` only for 64-bit processes  
`on:` enable for both 32- and 64-bit processes  
`off:` disable for both 32- and 64-bit `u`

`→processes`

`alloc_snapshot` [FTRACE]  
 Allocate the ftrace snapshot buffer on boot `u`  
`→up when the`  
`→debugging`  
`→boot up, and`  
`→as it needs`  
`→allowed.`

`amd_iommu=` [HW,X86-64]  
 Pass parameters to the AMD IOMMU driver in `u`  
`→the system.`

Possible values are:  
`fullflush` - enable flushing of IO/TLB `u`  
`→entries when`  
`→they are`  
`→reused, which`  
`→found in`  
`→for all`  
`→is not`  
`→isolation`  
`→This option`

`off` - do not initialize any AMD IOMMU `u`  
`→the system`  
`force_isolation` - Force device isolation `u`  
`→devices. The IOMMU driver`  
`→allowed anymore to lift`  
`→requirements as needed.`  
`→does not override iommu=pt`

`amd_iommu_dump=` [HW,X86-64]  
 Enable AMD IOMMU driver option to dump the `u`  
`→ACPI table`  
`→AMD IOMMU`  
`→driver will print ACPI tables for AMD IOMMU` `u`

↳during IOMMU initialization.

amd\_iommu\_intr= [HW,X86-64]  
Specifies one of the following AMD IOMMU

↳interrupt remapping modes:  
legacy - Use legacy interrupt remapping

↳mode. vpic - Use virtual APIC mode, which

↳allows IOMMU to inject interrupts directly

↳into guest. This mode requires kvm-amd.

↳avic=1. (Default when IOMMU HW support

↳is present.)

amijoy.map= [HW,JOY] Amiga joystick support  
Map of devices attached to JOY0DAT and

↳JOY1DAT  
Format: <a>,<b>  
See also Documentation/input/joydev/

↳joystick.rst

analog.map= [HW,JOY] Analog joystick and gamepad support  
Specifies type or capabilities of an analog

↳joystick  
connected to one of 16 gameports  
Format: <type1>,<type2>,..<type16>

apc= [HW,SPARC]  
Power management functions (SPARCstation-4/

↳5 + deriv.)  
Format: noidle  
Disable APC CPU standby support.

↳SPARCstation-Fox does  
not play well with APC CPU idle - disable

↳it if you have  
APC and your system crashes randomly.

apic= [APIC,X86] Advanced Programmable Interrupt

↳Controller  
Change the output verbosity while booting  
Format: { quiet (default) | verbose | debug

↳}  
Change the amount of debugging information

↳output  
when initialising the APIC and IO-APIC

↳components.  
For X86-32, this can also be used to

↪ specify an APIC driver name.  
 Format: apic=driver\_name  
 Examples: apic=bigsm

apic\_extnmi= [APIC,X86] External NMI delivery setting  
 Format: { bsp (default) | all | none }  
 bsp: External NMI is delivered only to CPU ↵  
 ↪ 0  
 ↪ CPUs as a backup of CPU 0  
 ↪ This is none: External NMI is masked for all CPUs. ↵  
 ↪ won't be useful so that a dump capture kernel ↵  
 ↪ shot down by NMI

autoconf= [IPV6]  
 See Documentation/networking/ipv6.rst.

↪ Controller show\_lapic= [APIC,X86] Advanced Programmable Interrupt ↵  
 ↪ the maximal Limit apic dumping. The parameter defines ↵  
 ↪ is possible number of local apics being dumped. Also it ↵  
 ↪ here. to set it to "all" by meaning -- no limit ↵  
 Format: { 1 (default) | 2 | ... | all }.  
 The parameter valid if only apic=debug or apic=verbose is specified.  
 Example: apic=debug show\_lapic=all

apm= [APM] Advanced Power Management  
 See header of arch/x86/kernel/apm\_32.c.

↪ mem-mapped) cards arcrimi= [HW,NET] ARCnet - "RIM I" (entirely ↵  
 Format: <io>,<irq>,<nodeID>

ataflop= [HW,M68k]

atarimouse= [HW,MOUSE] Atari Mouse

↪ RapidAccess, atkbd.extra= [HW] Enable extra LEDs and keys on IBM ↵  
 ↪ EzKey and similar keyboards

atkbd.reset= [HW] Reset keyboard during initialization

atkbd.set=	[HW] Select keyboard code set Format: <int> (2 = AT (default), 3 = PS/2)
↳similar atkbd.scroll=	[HW] Enable scroll wheel on MS Office and keyboards
↳mode atkbd.softraw=	[HW] Choose between synthetic and real raw Format: <bool> (0 = real, 1 = synthetic ↳(default))
atkbd.softrepeat=	[HW] Use software keyboard repeat
audit=	[KNL] Enable the audit sub-system Format: { "0"   "1"   "off"   "on" } 0   off - kernel audit is disabled and can ↳not be enabled until the next reboot unset - kernel audit is initialized but ↳disabled and will be fully enabled by the userspace ↳auditd. 1   on - kernel audit is initialized and ↳partially enabled, storing at most audit_backlog_ ↳limit messages in RAM until it is fully ↳enabled by the userspace auditd. Default: unset
audit_backlog_limit=	[KNL] Set the audit queue size limit. Format: <int> (must be >=0) Default: 64
↳default bau=	[X86_UV] Enable the BAU on SGI UV. The behavior is to disable the BAU (i.e. bau=0). Format: { "0"   "1" } 0 - Disable the BAU. 1 - Enable the BAU. unset - Disable the BAU.
baycom_epp=	[HW,AX25] Format: <io>,<mode>
baycom_par=	[HW,AX25] BayCom Parallel Port AX.25 Modem Format: <io>,<mode> See header of drivers/net/hamradio/baycom_

`par.c.`  
`baycom_ser_fdx= [HW,AX25]`  
 BayCom Serial Port AX.25 Modem (Full Duplex  
`Mode)`  
 Format: `<io>,<irq>,<mode>[,<baud>]`  
 See header of `drivers/net/hamradio/baycom_`  
`ser_fdx.c.`  
`baycom_ser_hdx= [HW,AX25]`  
 BayCom Serial Port AX.25 Modem (Half Duplex  
`Mode)`  
 Format: `<io>,<irq>,<mode>`  
 See header of `drivers/net/hamradio/baycom_`  
`ser_hdx.c.`  
`blkdevparts=` Manual partition parsing of block device(s)  
`for` embedded devices based on command line  
`input.` See `Documentation/block/cmdline-partition.`  
`rst`  
`boot_delay=` Milliseconds to delay each `printk` during  
`boot.` Values larger than 10 seconds (10000) are  
`changed to` no delay (0).  
 Format: integer  
`bootconfig` [KNL]  
`to an initrd` Extended command line options can be added  
`it.` and this will cause the kernel to look for  
 See `Documentation/admin-guide/bootconfig.rst`  
`bert_disable` [ACPI]  
 Disable BERT OS support on buggy BIOSes.  
`bgrt_disable` [ACPI][X86]  
 Disable BGRT to avoid flickering OEM logo.  
`bttv.card=` [HW,V4L] `bttv` (bt848 + bt878 based grabber  
`cards)`  
`bttv.radio=` Most important `insmod` options are available  
`as` kernel args too.  
`bttv.pll=` See `Documentation/admin-guide/media/bttv.rst`  
`bttv.tuner=`

`bulk_remove=off` [PPC] This parameter disables the use of the pSeries firmware feature for flushing multiple hptentries at a time.

`c101=` [NET] Moxa C101 synchronous serial card

`cachesize=` [BUGS=X86-32] Override level 2 CPU cache size detection. Sometimes CPU hardware bugs make them report the cache size incorrectly. The kernel will attempt work arounds to fix known problems, but for some CPUs it is not possible to determine what the correct size should be. This option provides an override for these situations.

`carrier_timeout=` [NET] Specifies amount of time (in seconds) that the kernel should wait for a network carrier. By default it waits 120 seconds.

`ca_keys=` [KEYS] This parameter identifies a specific key(s) on the system trusted keyring to be used for certificate trust validation.  
format: { id:<keyid> | builtin }

`cca=` [MIPS] Override the kernel pages' cache coherency algorithm. Accepted values range from 0 to 7 inclusive. See `arch/mips/include/asm/pgtable-bits.h` for platform specific values (SB1, Loongson3 and others).

`ccw_timeout_log` [S390] See `Documentation/s390/common_io.rst` for details.

`cgroup_disable=` [KNL] Disable a particular controller

Format: {name of the controller(s) to

disable}

The effects of cgroup\_disable=foo are:

- foo isn't auto-mounted if you mount all
- foo isn't visible as an individually
- foo isn't visible as an individually

cgroups in a single hierarchy

mountable subsystem

with this and {Currently only "memory" controller deal

usage. So cut the overhead, others just disable the

worthy} only cgroup\_disable=memory is actually

cgroup\_no\_v1= [KNL] Disable cgroup controllers and named

hierarchies in v1

Format: { { controller | "all" | "named" }  
[, { controller | "all" | "named" }  
... ] }

Like cgroup\_disable, but only applies to

cgroup v1; the blacklisted controllers remain

available in cgroup2. "all" blacklists all controllers and

"named" disables named mounts. Specifying both "all" and

"named" disables all v1 hierarchies.

cgroup.memory= [KNL] Pass options to the cgroup memory

controller.

Format: <string>

nosocket -- Disable socket memory

accounting. nokmem -- Disable kernel memory accounting.

checkreqprot [SELINUX] Set initial checkreqprot flag

value.

Format: { "0" | "1" }

See security/selinux/Kconfig help text.

0 -- check protection applied by kernel

(includes any implied execute protection).

1 -- check protection requested by

application. Default value is set via a kernel config

option. Value can be changed at runtime via

/sys/fs/selinux/checkreqprot.

Setting `checkreqprot` to 1 is deprecated.

`cio_ignore=` [S390]  
See [Documentation/s390/common\\_io.rst](#) for details.

`clk_ignore_unused` [CLK]  
Prevents the clock framework from automatically gating clocks that have not been explicitly enabled by a Linux device driver but are enabled in hardware at reset or by the bootloader/firmware. Note that this does not force such clocks to be always-on nor does it reserve those clocks in any way. This parameter is useful for debug and development, but should not be needed on a platform with proper driver support. For more information, see [Documentation/driver-api/clk.rst](#).

`clock=` [BUGS=X86-32, HW] `gettimeofday` `clocksource` `override`. [Deprecated]  
Forces specified `clocksource` (if available) to be used when calculating `gettimeofday()`. If specified `clocksource` is not available, it defaults to PIT.  
Format: { pit | tsc | cyclone | pmtmr }

`clocksource=` Override the default `clocksource`  
Format: <string>  
Override the default `clocksource` and use the `clocksource` with the name specified.  
Some `clocksource` names to choose from, depending on the platform:  
[all] `jiffies` (this is the base, fallback `clocksource`)  
[ACPI] `acpi_pm`  
[ARM] `imx_timer1`, `OSTS`, `netx_timer`, `mpu_timer2`, `pxa_timer`, `timer3`, `32k_counter`, `timer0_1`

[X86-32] pit,hpet,tsc;  
scx200\_hrt on Geode; cyclone on IBM

→x440

[MIPS] MIPS  
[PARISC] cr16  
[S390] tod  
[SH] SuperH  
[SPARC64] tick  
[X86-64] hpet,tsc

clocksource.arm\_arch\_timer.evtstrm=  
[ARM,ARM64]  
Format: <bool>  
Enable/disable the eventstream feature of

→the ARM  
→WFE-based polling  
→production

architected timer so that code using  
loops can be debugged more effectively on  
systems.

clearcpuid=BITNUM [X86]  
Disable CPUID feature X for the kernel. See  
arch/x86/include/asm/cpufeatures.h for the

→valid bit  
→not necessarily  
→specific  
→directly  
→anything  
→from  
→cpuinfo.  
→you disable

numbers. Note the Linux specific bits are  
stable over kernel options, but the vendor  
ones should be.  
Also note that user programs calling CPUID  
or using the feature without checking  
will still see it. This just prevents it  
being used by the kernel or shown in /proc/  
Also note the kernel might malfunction if  
some critical bits.

cma=nn[MG]@[start[MG] [-end[MG]]]  
[ARM,X86,KNL]  
Sets the size of kernel global memory area

→for  
→optionally the  
→address range of

contiguous memory allocations and  
placement constraint by the physical  
memory allocations. A value of 0 disables

→CMA  
altogether. For more information, see  
include/linux/dma-contiguous.h

    cmo\_free\_hint= [PPC] Format: { yes | no }  
Specify whether pages are marked as being  
→inactive  
when they are freed. This is used in CMO  
→environments  
to determine OS memory pressure for page  
→stealing by  
a hypervisor.  
Default: yes

    coherent\_pool=nn[KMG] [ARM,KNL]  
Sets the size of memory pool for coherent,  
→atomic dma  
allocations, by default set to 256K.

    com20020= [HW,NET] ARCnet - COM20020 chipset  
Format:  
→<timeout>]]]]  
<io>[,<irq>[,<nodeID>[,<backplane>[,<ckp>[,

    com90io= [HW,NET] ARCnet - COM90xx chipset  
→(IO-mapped buffers)  
Format: <io>[,<irq>]

    com90xx= [HW,NET]  
ARCnet - COM90xx chipset (memory-mapped  
→buffers)  
Format: <io>[,<irq>[,<memstart>]]

    condev= [HW,S390] console device  
conmode=

    console= [KNL] Output console device and options.  
tty<n> Use the virtual console device <n>.  
ttyS<n>[,options]  
ttyUSB0[,options]  
Use the specified serial port. The options  
→are of  
the form "bbbbpnf", where "bbbb" is the  
→baud rate,  
"p" is parity ("n", "o", or "e"), "n" is  
→number of  
bits, and "f" is flow control ("r" for RTS  
→or  
omit it). Default is "9600n8".

See Documentation/admin-guide/  
 serial-console.rst for more  
 information. See  
 Documentation/networking/netconsole.rst for  
 an alternative.

```
uart[8250],io,<addr>[,options]
uart[8250],mmio,<addr>[,options]
uart[8250],mmio16,<addr>[,options]
uart[8250],mmio32,<addr>[,options]
uart[8250],0x<addr>[,options]
```

Start an early, polled-mode console on the  
 8250/16550  
 address,  
 either 8-bit  
 is assumed  
 specified in  
 if unspecified,  
 is for  
 but a braille  
 instance

```
hvc<n> Use the hypervisor console device <n>. This  

  both Xen and PowerPC hypervisors.
```

If the device connected to the port is not a TTY  
 device, prepend "brl," before the device type, for  
 instance

```
console=brl,ttyS0
```

For now, only VisioBraille is supported.

```
console_msg_format=
  [KNL] Change console messages format
  default
  By default we print messages on consoles in
  "[time stamp] text\n" format (time stamp
  may not be
  printed, depending on CONFIG_PRINTK_TIME or
  `printk_time' param).
  syslog
  Switch to syslog format: "<%u>[time stamp]
  text\n"
```

`loglevel`  
by `syslog()`  
or to reading IOW, each message will have a facility and prefix. The format is similar to one used by `syslog()`, or to executing `"dmesg -S --raw"` from `/proc/kmsg`.

`consoleblank=`  
timeout in [KNL] The console blank (screen saver) seconds. A value of 0 disables the blank timer. Defaults to 0.

`coredump_filter=`  
[KNL] Change the default value for `/proc/<pid>/coredump_filter`. See also `Documentation/filesystems/proc.rst`.

`coresight_cpu_debug.enable`  
[ARM,ARM64]  
Format: `<bool>`  
Enable/disable the CPU sampling based debugging.  
0: default value, disable debugging  
1: enable debugging at boot time

`cpuidle.off=1` [CPU\_IDLE]  
disable the cpuidle sub-system

`cpuidle.governor=`  
[CPU\_IDLE] Name of the cpuidle governor to use.

`cpufreq.off=1` [CPU\_FREQ]  
disable the cpufreq sub-system

`cpu_init_udelay=N`  
[X86] Delay for N microsec between assert and de-assert  
delay occurs  
resume from suspend.  
of APIC INIT to start processors. This delay occurs on every CPU online, such as boot, and resume from suspend.  
Default: 10000

`cpcihp_generic=` [HW,PCI] Generic port I/O CompactPCI driver  
Format:  
`<first_slot>,<last_slot>,<port>,<enum_bit>[,<debug>]`

```
crashkernel=size[KMG][@offset[KMG]]
[KNL] Using kexec, Linux can switch to a
↳ 'crash kernel'
↳ physical
↳ that kernel
↳ suitable offset
↳ first, and
↳ '@offset'
hasn't been specified.
See Documentation/admin-guide/kdump/kdump.
↳ rst for further details.
```

```
crashkernel=range:size1[,range2:size2,...][@offset]
[KNL] Same as above, but depends on the
↳ memory
↳ is
start-[end] where start and end are both
a memory unit (amount[KMG]). See also
Documentation/admin-guide/kdump/kdump.rst
↳ for an example.
```

```
crashkernel=size[KMG],high
[KNL, x86_64] range could be above 4G.
↳ Allow kernel
↳ so could
↳ installed.
↳ below 4G, if
available.
It will be ignored if crashkernel=X is
↳ specified.
```

```
crashkernel=size[KMG],low
[KNL, x86_64] range under 4G. When
↳ crashkernel=X,high
↳ memory region
↳ system
↳ swiotlb
↳ enough extra
is passed, kernel could allocate physical
above 4G, that cause second kernel crash on
that require some amount of low memory, e.g.
requires at least 64M+32K low memory, also
```

low memory is needed to make sure DMA buffers for 32-bit devices won't run out. Kernel would try to allocate at least 256M below 4G automatically. This one let user to specify own low range under 4G for second kernel instead. 0: to disable low allocation. It will be ignored when crashkernel=X,high is not used or memory reserved is below 4G.

cryptomgr.notests [KNL] Disable crypto self-tests

cs89x0\_dma= [HW,NET]  
Format: <dma>

cs89x0\_media= [HW,NET]  
Format: { rj45 | aui | bnc }

dasd= [HW,NET]  
See header of drivers/s390/block/dasd\_devmap.c.

db9.dev[2|3]= [HW,JOY] Multisystem joystick support via parallel port (one device per port)  
Format: <port#>,<type>  
See also Documentation/input/devices/joystick-parport.rst

ddebug\_query= [KNL,DYNAMIC\_DEBUG] Enable debug messages at early boot time. See Documentation/admin-guide/dynamic-debug-howto.rst for details. Deprecated, see dyndbg.

debug [KNL] Enable kernel debugging (events log level).

debug\_boot\_weak\_hash [KNL] Enable printing [hashed] pointers early in the boot sequence. If enabled, we use a weak hash instead of siphash to hash pointers. Use this option if you are seeing instances of '(\_\_ptrval\_\_)' and

↪ need to see a value (hashed pointer) instead.

↪ Cryptographically insecure, please do not use on production kernels.

```

debug_locks_verbose=
    [KNL] verbose self-tests
    Format=<0|1>
    Print debugging info while doing the
  
```

↪ locking API self-tests. We default to 0 (no extra messages),

↪ setting it to 1 will print a lot more information -

↪ normally only useful to kernel developers.

```

debug_objects    [KNL] Enable object debugging
no_debug_objects [KNL] Disable object debugging
  
```

↪ this

```

debug_guardpage_minorder=
    [KNL] When CONFIG_DEBUG_PAGEALLOC is set,
  
```

↪ pages that will be intentionally kept free (and hence

↪ protected) by the buddy allocator. Bigger value increase the

↪ probability of catching random memory corruption, but

↪ reduce the amount of memory for normal system use. The

↪ maximum possible value is MAX\_ORDER/2. Setting

↪ this parameter to 1 or 2 should be enough to identify most

↪ random memory corruption problems caused by bugs

↪ in kernel or driver code when a CPU writes to (or reads

↪ from) a random memory location. Note that there

↪ exists a class of memory corruptions problems caused by

↪ buggy H/W or F/W or by drivers badly programming DMA

↪ (basically when memory is written at bus level and the CPU

↳MMU is bypassed) which are not detectable by CONFIG\_DEBUG\_PAGEALLOC, hence this option, ↳

↳will not help tracking down these problems.

debug\_pagealloc= [KNL] When CONFIG\_DEBUG\_PAGEALLOC is set, ↳

↳this parameter enables the feature at boot time. By ↳

↳default, it is disabled and the system will work mostly ↳

↳the same as a kernel built without CONFIG\_DEBUG\_PAGEALLOC. Note: to get most of debug\_pagealloc error, ↳

↳reports, it's useful to also enable the page\_owner, ↳

↳functionality. on: enable the feature

debugpat [X86] Enable PAT debugging

decnet.addr= [HW,NET]  
Format: <area>[,<node>]  
See also Documentation/networking/decnet.

↳rst.

default\_hugepagesz= [HW] The size of the default HugeTLB page. ↳

↳This is the size represented by the legacy /proc/ ↳

↳hugepages APIs. In addition, this is the default, ↳

↳hugetlb size used for shmget(), mmap() and mounting, ↳

↳hugetlbfs filesystems. If not specified, defaults to, ↳

↳the architecture's default huge page size. ↳

↳Huge page sizes are architecture dependent. See also Documentation/admin-guide/mm/hugetlbpage.

↳rst. Format: size[KMG]

deferred\_probe\_timeout= [KNL] Debugging option to set a timeout in, ↳

↳seconds for deferred probe to give up waiting on, ↳

↳dependencies to probe. Only specific dependencies, ↳

↪(subsystems or drivers) that have opted in will be ignored.  
 ↪ A timeout of 0 will timeout at the end of initcalls. This  
 ↪option will also dump out devices still on the deferred  
 ↪probe list after retrying.

dfltcc= [HW,S390]  
 Format: { on | off | def\_only | inf\_only |  
 ↪always }  
 ↪compression on on: s390 zlib hardware support for  
 ↪(default) level 1 and decompression  
 ↪deflate off: No s390 zlib hardware support  
 ↪inflate def\_only: s390 zlib hardware support for  
 ↪selected compression only (compression on level 1)  
 ↪support (used for debugging) inf\_only: s390 zlib hardware support for  
 always: only (decompression)  
 Same as 'on' but ignores the  
 level always using hardware

dhash\_entries= [KNL]  
 Set number of hash buckets for dentry cache.

disable\_ltb\_segments [PPC]  
 ↪segments. This Disables the use of 1TB hash page table  
 ↪segments which causes the kernel to fall back to 256MB  
 ↪require an SLB can be useful when debugging issues that  
 miss to occur.

stress\_slb [PPC]  
 ↪and flushes Limits the number of kernel SLB entries,  
 ↪faults them frequently to increase the rate of SLB  
 on kernel addresses.

disable= [IPV6]  
 See Documentation/networking/ipv6.rst.

hardened\_usercopy=

`↳whether` [KNL] Under `CONFIG_HARDENED_USERCOPY`,  
hardening is enabled for this boot. Hardened  
usercopy checking is used to protect the  
`↳kernel` from reading or writing beyond known memory  
allocation boundaries as a proactive defense  
against bounds-checking flaws in the  
`↳kernel's` `copy_to_user()/copy_from_user()` interface.  
on Perform hardened usercopy checks (default).  
off Disable hardened usercopy checks.

`disable_radix` [PPC]  
Disable RADIX MMU mode on POWER9

`disable_tlbie` [PPC]  
Disable TLBIE instruction. Currently does  
`↳not work` with KVM, with HASH MMU, or with coherent  
`↳accelerators.`

`disable_cpu_apicid=` [X86,APIC,SMP]  
Format: `<int>`  
The number of initial APIC ID for the  
corresponding CPU to be disabled at boot,  
mostly used for the kdump 2nd kernel to  
disable BSP to wake up multiple CPUs without  
causing system reset or hang due to sending  
INIT from AP to BSP.

`perf_v4_pmi=` [X86,INTEL]  
Format: `<bool>`  
Disable Intel PMU counter freezing feature.  
The feature only exists starting from  
Arch Perfmon v4 (Skylake and newer).

`disable_ddw` [PPC/PSERIES]  
Disable Dynamic DMA Window support. Use  
`↳this if` to workaround buggy firmware.

`disable_ipv6=` [IPV6]  
See Documentation/networking/ipv6.rst.

`disable_mtrr_cleanup` [X86]  
The kernel tries to adjust MTRR layout from  
`↳continuous` to discrete, to make X server driver able  
`↳to add WB` entry later. This parameter disables that.

`disable_mtrr_trim` [X86, Intel and AMD only]  
 By default the kernel will trim any uncacheable memory out of your available memory pool based on MTRR settings. This parameter disables that behavior, possibly causing your machine to run very slowly.

`disable_timer_pin_1` [X86]  
 Disable PIN 1 of APIC timer  
 Can be useful to work around chipset bugs.

`dis_ucode_ldr` [X86] Disable the microcode loader.

`dma_debug=off` If the kernel is compiled with `DMA_API_DEBUG` support, this option disables the debugging code at boot.

`dma_debug_entries=<number>`  
 This option allows to tune the number of preallocated entries for DMA-API debugging code. One entry is required per DMA-API allocation. Use this if the DMA-API debugging code disables itself because the architectural default is too low.

`dma_debug_driver=<driver_name>`  
 With this option the DMA-API debugging filter feature can be enabled at boot time. Just pass the driver to filter for as the parameter. The filter can be disabled or changed to another driver later using `sysfs`.

`driver_async_probe=` [KNL]  
 List of driver names to be probed asynchronously.  
 Format: `<driver_name1>,<driver_name2>...`

`drm.edid_firmware=[<connector>:]<file>[, [<connector>:]<file>]`

Broken monitors, graphic adapters, KVMs and panels may send no or incorrect EDID data. This parameter allows to specify an EDID in the `/lib/firmware` directory that are Generic built-in EDID data sets are used, `edid/1024x768.bin`, `edid/1280x1024.bin`, `edid/1680x1050.bin`, or `edid/1920x1080.bin` and no file with the same name exists. Instructions how to build your own EDID are available in `Documentation/admin-guide/edid`. An EDID data set will only be used for a particular connector, if its name and a colon are prepended to the EDID name. Each connector may use a unique EDID set by separating the files with a comma. An EDID data set with no connector name will be used for any connectors not explicitly specified.

```
dscc4.setup= [NET]
dt_cpu_ftrs= [PPC]
Format: {"off" | "known"}
Control how the dt_cpu_ftrs device-tree is used for CPU feature discovery and setup (if it exists).
off: Do not use it, fall back to legacy cpu table.
known: Do not pass through unknown features to guests or userspace, only those that the kernel is aware of.
```

```
dump_apple_properties [X86]
Dump name and content of EFI device properties on x86 Macs. Useful for driver authors to
```

- ↪determine what data is available or for
- ↪reverse-engineering.
- dyndbg[="val"] [KNL,DYNAMIC\_DEBUG]
  - module.dyndbg[="val"]

Enable debug messages at boot time. See Documentation/admin-guide/
- ↪dynamic-debug-howto.rst for details.
- nopku [X86] Disable Memory Protection Keys CPU

↪feature found in some Intel CPUs.
- module.async\_probe [KNL]

Enable asynchronous probe on this module.
- early\_ioremap\_debug [KNL]

Enable debug messages in early\_ioremap

↪support. This is useful for tracking down temporary early

↪mappings which are not unmapped.
- earlycon= [KNL] Output early console device and

↪options.

When used with no options, the early

↪console is determined by stdout-path property in

↪device tree's chosen node or the ACPI SPCR table if

↪supported by the platform.
- cdns,<addr>[,options]

Start an early, polled-mode console on a

↪Cadence (xuartps) serial port at the specified

↪address. Only supported option is baud rate. If baud rate

↪is not specified, the serial port must already be

↪setup and configured.
- uart[8250],io,<addr>[,options]
  - uart[8250],mmio,<addr>[,options]
  - uart[8250],mmio32,<addr>[,options]
  - uart[8250],mmio32be,<addr>[,options]

`uart[8250],0x<addr>[,options]`  
Start an early, polled-mode console on the  
→8250/16550  
UART at the specified I/O port or MMIO  
→address.  
MMIO inter-register address stride is  
→either 8-bit  
(mmio) or 32-bit (mmio32 or mmio32be).  
If none of [io|mmio|mmio32|mmio32be], <addr>  
→ is assumed  
to be equivalent to 'mmio'. 'options' are  
→specified  
in the same format described for  
→"console=ttyS<n>"; if  
unspecified, the h/w is not initialized.

`pl011,<addr>`  
`pl011,mmio32,<addr>`  
Start an early, polled-mode console on a  
→pl011 serial  
port at the specified address. The pl011  
→serial port  
must already be setup and configured.  
→Options are not  
yet supported. If 'mmio32' is specified,  
→then only  
the driver will use only 32-bit accessors  
→to read/write  
the device registers.

`meson,<addr>`  
Start an early, polled-mode console on a  
→meson serial  
port at the specified address. The serial  
→port must  
already be setup and configured. Options  
→are not yet  
supported.

`msm_serial,<addr>`  
Start an early, polled-mode console on an  
→msm serial  
port at the specified address. The serial  
→port  
must already be setup and configured.  
→Options are not  
yet supported.

`msm_serial_dm,<addr>`  
Start an early, polled-mode console on an  
→msm serial

- dm port at the specified address. The serial port must already be setup and configured. Options are not yet supported.
- `owl,<addr>`  
Start an early, polled-mode console on a serial port of an Actions Semi SoC, such as S500 or S900, at the specified address. The serial port must already be setup and configured. Options are not yet supported.
- `rda,<addr>`  
Start an early, polled-mode console on a serial port of an RDA Micro SoC, such as RDA8810PL, at the specified address. The serial port must already be setup and configured. Options are not yet supported.
- `sbi`  
Use RISC-V SBI (Supervisor Binary Interface) for early console.
- `smh` Use ARM semihosting calls for early console.
- `s3c2410,<addr>`  
`s3c2412,<addr>`  
`s3c2440,<addr>`  
`s3c6400,<addr>`  
`s5pv210,<addr>`  
`exynos4210,<addr>`  
Use early console provided by serial driver on Samsung SoCs, requires selecting proper type and a correct base address of the selected UART port. The serial port must already be setup and configured. Options are not yet supported.
- `lantiq,<addr>`  
Start an early, polled-mode console on a

↳lantiq serial (lqasc) port at the specified address. The  
↳serial port must already be setup and configured.  
↳Options are not yet supported.

↳UART driver `lpuart,<addr>`  
↳processors. `lpuart32,<addr>`  
Use early console provided by Freescale LP  
↳the serial found on Freescale Vybrid and QorIQ LS1021A  
A valid base address must be provided, and  
port must already be setup and configured.

↳console on the `ec_imx21,<addr>`  
↳address. The UART `ec_imx6q,<addr>`  
Start an early, polled-mode, output-only  
Freescale i.MX UART at the specified  
must already be setup and configured.

↳setup `ar3700_uart,<addr>`  
↳supported. Start an early, polled-mode console on the  
Armada 3700 serial port at the specified  
address. The serial port must already be  
and configured. Options are not yet

↳Qualcomm `qcom_geni,<addr>`  
↳at the Start an early, polled-mode console on a  
Generic Interface (GENI) based serial port  
↳already be specified address. The serial port must  
↳supported. setup and configured. Options are not yet

↳the EFI `efifb,[options]`  
↳On cache Start an early, unaccelerated console on  
↳memory for the framebuffer (if available).  
coherent non-x86 systems that use system  
the framebuffer, pass the 'ram' option so

↳that it is mapped with the correct attributes.

```

linflex,<addr>
Use early console provided by Freescale
↳LINFlexD UART serial driver for NXP S32V234 SoCs. A valid
↳base address must be provided, and the serial
↳port must already be setup and configured.

earlyprintk= [X86,SH,ARM,M68k,S390]
earlyprintk=vga
earlyprintk=sclp
earlyprintk=xen
earlyprintk=serial[,ttySn[,baudrate]]
earlyprintk=serial[,0x...[,baudrate]]
earlyprintk=ttySn[,baudrate]
earlyprintk=dbgp[debugController#]
earlyprintk=pciserial[,force],bus:device.
↳function[,baudrate] earlyprintk=xdbc[xhciController#]

earlyprintk is useful when the kernel
↳crashes before the normal console is initialized. It is
↳not enabled by default because it has some cosmetic
↳problems.

Append ",keep" to not disable it when the
↳real console takes over.

Only one of vga, efi, serial, or usb debug
↳port can be used at a time.

Currently only ttyS0 and ttyS1 may be
↳specified by name. Other I/O ports may be explicitly
↳specified on some architectures (x86 and arm at
↳least) by replacing ttySn with an I/O port address,
↳like this:

        earlyprintk=serial,0x1008,115200
You can find the port for a given device in
/proc/tty/driver/serial:
        2: uart:ST16650V2 port:00001008

```

→irq:18 ...

→is not

→overwritten by

→guests.

→use of a

→of the

edac\_report=

→overridden

→module.

→EDAC.

→W event.

ekgdboc=

→debugging

→with

→parameter

→available

→debugging

→instead.

edd=

Interaction with the standard serial driver,  
very good.

The VGA and EFI output is eventually,  
the real console.

The xen output can only be used by Xen PV,  
guests.

The sclp output can only be used on s390.

The optional "force" to "pciserial" enables,  
PCI device even when its classcode is not,  
UART class.

[HW,EDAC] Control how to report EDAC event  
Format: {"on" | "off" | "force"}  
on: enable EDAC to report H/W event. May be,  
by other higher priority error reporting,  
off: disable H/W event reporting through,  
force: enforce the use of EDAC to report H/  
default: on.

[X86,KGDB] Allow early kernel console,  
ekgdboc=kbd

This is designed to be used in conjunction,  
the boot argument: earlyprintk=vga

This parameter works in place of the kgdboc,  
but can only be used if the backing tty is,  
very early in the boot process. For early,  
via a serial port see kgboc\_earlycon,  
instead.

[EDD]

```

Format: {"off" | "on" | "skip[mbr]"}

efi= [EFI]
Format: { "old_map", "nochunk", "noruntime",
↳ "debug", "nosoftreserve", "disable_early_
↳pci_dma", "no_disable_early_pci_dma" }
old_map [X86-64]: switch to the old_
↳ioremap-based EFI runtime services mapping. [Needs CONFIG_X86_
↳UV=y nochunk: disable reading files in "chunks"
↳in the EFI boot stub, as chunking can cause problems
↳with some firmware implementations.
noruntime : disable EFI runtime services
↳support debug: enable misc debug output
↳Purpose) nosoftreserve: The EFI_MEMORY_SP (Specific
↳the attribute may cause the kernel to reserve
↳this memory range for a memory mapping driver to
↳base type claim. Specify efi=nosoftreserve to disable
↳RAM"). reservation and treat the memory by its
↳busmaster bit on all (i.e. EFI_CONVENTIONAL_MEMORY / "System
↳busmaster bit set disable_early_pci_dma: Disable the
↳stub PCI bridges while in the EFI boot stub
no_disable_early_pci_dma: Leave the
on all PCI bridges while in the EFI boot

efi_no_storage_paranoid [EFI; X86]
↳50% of Using this parameter you can use more than
↳parameter only if your efi variable storage. Use this
↳sane gc and you are really sure that your UEFI does
↳brick. fulfills the spec otherwise your board may

efi_fake_mem= nn[KMG]@ss[KMG]:aa[,nn[KMG]@ss[KMG]:aa,..]
↳[EFI; X86]

```

↪range by Add arbitrary attribute to specific memory, updating original EFI memory map. Region of memory which aa attribute is from ss to ss+nn.

↪added to is

↪2G@0x10a0000000:0x10000 If efi\_fake\_mem=2G@4G:0x10000, is specified, EFI\_MEMORY\_MORE\_RELIABLE(0x10000) attribute is added to range 0x100000000-0x180000000 and 0x10a0000000-0x1120000000.

↪the If efi\_fake\_mem=8G@9G:0x40000 is specified, EFI\_MEMORY\_SP(0x40000) attribute is added to range 0x240000000-0x43fffffff.

↪of EFI memmap Using this parameter you can do debugging related features. For example, you can do Address Range Mirroring feature even if your box doesn't support it, or mark specific memory as "soft reserved".

↪contains an `efivar_ssdt=` [EFI; X86] Name of an EFI variable that is to be dynamically loaded by Linux. If there are multiple variables with the same name but with different vendor GUIDs, all of them will be loaded. See Documentation/admin-guide/acpi/ssdt-overlays.rst for details.

↪eisa\_irq\_edge= [PARISC,HW] See header of drivers/parisc/eisa.c.

↪elanfreq= [X86-32] See comment before function elanfreq\_setup() in arch/x86/kernel/cpu/cpufreq/elanfreq.c.

↪elfcorehdr=[size[KMG]@]offset[KMG] [IA64,PPC,SH,X86,S390]

↪kernel core Specifies physical address of start of  
 ↪Generally image elf header and optionally the size.  
 ↪capture kernel. kexec loader will pass this option to  
 ↪rst for details. See Documentation/admin-guide/kdump/kdump.

enable\_mtrr\_cleanup [X86]  
 ↪continuous The kernel tries to adjust MTRR layout from  
 ↪to add WB to discrete, to make X server driver able  
 entry later. This parameter enables that.

enable\_timer\_pin\_1 [X86]  
 ↪default. Enable PIN 1 of APIC timer  
 Can be useful to work around chipset bugs  
 (in particular on some ATI chipsets).  
 The kernel tries to set a reasonable

enforcing [SELINUX] Set initial enforcing status.  
 Format: {"0" | "1"}  
 See security/selinux/Kconfig help text.  
 0 -- permissive (log only, no denials).  
 1 -- enforcing (deny and log).  
 Default value is 0.  
 Value can be changed at runtime via  
 /sys/fs/selinux/enforce.

↪(ERST) erst\_disable [ACPI]  
 Disable Error Record Serialization Table  
 support.

↪option, which ether= [HW,NET] Ethernet cards parameters  
 ↪for details. This option is obsoleted by the "netdev="

↪regardless of evm= [EVM]  
 Format: { "fix" }  
 Permit 'security.evm' to be updated  
 current integrity status.

failslab=  
 fail\_page\_alloc=

`fail_make_request=[KNL]`  
General fault injection mechanism.  
Format: `<interval>,<probability>,<space>,<times>`  
See also `Documentation/fault-injection/`.

`floppy=` [HW]  
See `Documentation/admin-guide/blockdev/floppy.rst`.

`force_pal_cache_flush`  
[IA-64] Avoid `check_sal_cache_flush` which may hang on buggy `SAL_CACHE_FLUSH` implementations. Using this parameter will force `ia64_sal_cache_flush` to call `ia64_pal_cache_flush` instead of `SAL_CACHE_FLUSH`.

`forcepae` [X86-32]  
Forcefully enable Physical Address Extension (PAE). Many Pentium M systems disable PAE but may have a functionally usable PAE implementation. Warning: use of this parameter will taint the kernel and may cause unknown problems.

`ftrace=[tracer]`  
[FTRACE] will set and start the specified tracer as early as possible in order to facilitate early boot debugging.

`ftrace_dump_on_oops[=orig_cpu]`  
[FTRACE] will dump the trace buffers on oops. If no parameter is passed, `ftrace` will dump buffers of all CPUs, but if you pass `orig_cpu`, it will dump only the buffer of the CPU that triggered the oops.

`ftrace_filter=[function-list]`  
[FTRACE] Limit the functions traced by the function tracer at boot up. `function-list` is a comma

↪ separated list of functions. This list can be changed  
 ↪ at run time by the `set_ftrace_filter` file in the  
 ↪ `debugfs` tracing directory.

`ftrace_notrace=[function-list]`  
 [FTRACE] Do not trace the functions  
 ↪ specified in `function-list`. This list can be changed at  
 ↪ run time by the `set_ftrace_notrace` file in the  
 ↪ `debugfs` tracing directory.

`ftrace_graph_filter=[function-list]`  
 [FTRACE] Limit the top level callers  
 ↪ functions traced by the function graph tracer at boot up.  
 ↪ `function-list` is a comma separated list of  
 ↪ functions that can be changed at run time by the  
 ↪ `set_graph_function` file in the `debugfs`  
 ↪ tracing directory.

`ftrace_graph_notrace=[function-list]`  
 [FTRACE] Do not trace from the functions  
 ↪ specified in `function-list`. This list is a comma  
 ↪ separated list of functions that can be changed at run time  
 ↪ by the `set_graph_notrace` file in the `debugfs`  
 ↪ tracing directory.

`ftrace_graph_max_depth=<uint>`  
 [FTRACE] Used with the function graph  
 ↪ tracer. This is the max depth it will trace into a function.  
 ↪ This value can be changed at run time by the `max_graph_`  
 ↪ `depth` file in the `tracefs` tracing directory. default:  
 ↪ 0 (no limit)

`fw_devlink=`  
 ↪ and supplier [KNL] Create device links between consumer  
 ↪ the devices by scanning the firmware to infer  
 ↪ consumer/supplier relationships. This

↪ feature is especially useful when drivers are loaded.  
↪ as modules as it ensures proper ordering of tasks like  
↪ device probing (suppliers first, then consumers), supplier  
↪ boot state clean up (only after all consumers have  
↪ probed), suspend/resume & runtime PM (consumers  
↪ first, then suppliers).  
Format: { off | permissive | on | rpm }  
↪ firmware info. off -- Don't create device links from  
↪ firmware info permissive -- Create device links from  
↪ state clean but use it only for ordering boot  
↪ up (sync\_state() calls).  
↪ info and use it on -- Create device links from firmware  
↪ ordering. to enforce probe and suspend/resume  
↪ runtime PM. rpm -- Like "on", but also use to order

```
gamecon.map[2|3]=  
↪ PSX pad [HW,JOY] Multisystem joystick and NES/SNES/  
↪ per port) support via parallel port (up to 5 devices  
↪ <pad5> Format: <port#>,<pad1>,<pad2>,<pad3>,<pad4>,  
↪ joystick-parport.rst See also Documentation/input/devices/
```

```
gamma= [HW,DRM]  
gart_fix_e820= [X86_64] disable the fix e820 for K8 GART  
Format: off | on  
default: on  
gcov_persist= [GCOV] When non-zero (default), profiling  
↪ data for kernel modules is saved and remains  
↪ accessible via debugfs, even when the module is unloaded/  
↪ reloaded. When zero, profiling data is discarded and  
↪ associated
```

debugfs files are removed at module unload.

`time.`

`goldfish` [X86] Enable the goldfish android emulator.

`platform.` Don't use this when you are not running on the android emulator.

`the`

`gpt` [EFI] Forces disk with valid GPT signature.

`but` invalid Protective MBR to be treated as GPT.

`If the` primary GPT is corrupted, it enables the backup/alternate GPT to be used instead.

`backup/alternate`

`grcan.enable0=` [HW] Configuration of physical interface 0. Determines the "Enable 0" bit of the configuration register.

Format: 0 | 1  
Default: 0

`grcan.enable1=` [HW] Configuration of physical interface 1. Determines the "Enable 0" bit of the configuration register.

Format: 0 | 1  
Default: 0

`grcan.select=` [HW] Select which physical interface to use. Format: 0 | 1  
Default: 0

`grcan.txsize=` [HW] Sets the size of the tx buffer. Format: <unsigned int> such that (txsize & ~0x1fffc0) == 0. Default: 1024

`grcan.rxsize=` [HW] Sets the size of the rx buffer. Format: <unsigned int> such that (rxsize & ~0x1fffc0) == 0. Default: 1024

`gpio-mockup.gpio_mockup_ranges` [HW] Sets the ranges of gpiochip of for this device. Format: <start1>,<end1>,<start2>,<end2>...

`hardlockup_all_cpu_backtrace=` [KNL] Should the hard-lockup detector generate backtraces on all cpus. Format: 0 | 1

hashdist= [KNL,NUMA] Large hashes allocated during boot are distributed across NUMA nodes. Defaults on for 64-bit NUMA, off otherwise. Format: 0 | 1 (for off | on)

hcl= [IA-64] SGI's Hardware Graph compatibility layer

hd= [EIDE] (E)IDE hard drive subsystem geometry Format: <cyl>,<head>,<sect>

hest\_disable [ACPI] Disable Hardware Error Source Table (HEST) support; corresponding firmware-first mode error processing logic will be disabled.

highmem=nn[KMG] [KNL,BOOT] forces the highmem zone to have an exact size of <nn>. This works even on boxes that have no highmem otherwise. This also works to reduce highmem size on bigger boxes.

highres= [KNL] Enable/disable high resolution timer mode. Valid parameters: "on", "off" Default: "on"

hlt [BUGS=ARM,SH]

hpet= [X86-32,HPET] option to control HPET usage Format: { enable (default) | disable | verbose } disable: disable HPET and use PIT instead force: allow force enabled of undocumented chips (ICH4, VIA, nVidia) verbose: show contents of HPET registers during setup

hpet\_mmap= [X86, HPET\_MMAP] Allow userspace to mmap HPET registers. Default set by CONFIG\_HPET\_MMAP\_DEFAULT.

<p>hugetlb_cma= →allocation</p> <p>→allocate gigantic →enabled, the →is skipped.</p>	<p>[HW] The size of a cma area used for of gigantic hugepages. Format: nn[KMGTPe]</p> <p>Reserve a cma area of given size and hugepages using the cma allocator. If boot-time allocation of gigantic hugepages</p>
<p>hugepages= →boot. →specifies →allocated. →the command →allocate for →rst.</p>	<p>[HW] Number of HugeTLB pages to allocate at If this follows hugepagesz (below), it the number of pages of hugepagesz to be If this is the first HugeTLB parameter on line, it specifies the number of pages to the default huge page size. See also Documentation/admin-guide/mm/hugetlbpage. Format: &lt;integer&gt;</p>
<p>hugepagesz= →is used in →allocate huge →once for →sizes are →rst.</p>	<p>[HW] The size of the HugeTLB pages. This conjunction with hugepages (above) to pages of a specific size at boot. The pair hugepagesz=X hugepages=Y can be specified each supported huge page size. Huge page architecture dependent. See also Documentation/admin-guide/mm/hugetlbpage. Format: size[KMG]</p>
<p>hung_task_panic= →generate panics. →when a</p>	<p>[KNL] Should the hung task detector Format: 0   1</p> <p>A value of 1 instructs the kernel to panic hung task is detected. The default value is</p>

↳controlled by the CONFIG\_BOOTPARAM\_HUNG\_TASK\_PANIC\_

↳build-time option. The value selected by this boot\_

↳parameter can be changed later by the kernel.hung\_task\_

↳panic sysctl.

hvc\_iucv= [S390] Number of z/VM IUCV hypervisor\_

↳console (HVC) terminal devices. Valid values: 0..8

hvc\_iucv\_allow= [S390] Comma-separated list of z/VM user\_

↳IDs. If specified, z/VM IUCV HVC accepts\_

↳connections from listed z/VM user IDs only.

hv\_nopvspin [X86,HYPER\_V] Disables the paravirt\_

↳spinlock optimizations which allow the hypervisor to\_

↳'idle' the guest on lock contention.

keep\_bootcon [KNL] Do not unregister boot console at start.\_

↳This is only useful for debugging when something happens\_

↳in the window between unregistering the boot console and\_

↳initializing the real console.

i2c\_bus= [HW] Override the default board specific\_

↳I2C bus speed or register an additional I2C bus\_

↳that is not registered from board\_

↳initialization code. Format:  
<bus\_id>,<clkrate>

i8042.debug [HW] Toggle i8042 debug mode

i8042.unmask\_kbd\_data [HW] Enable printing of interrupt data from\_

↳the KBD port (disabled by default, and as a\_

↳pre-condition requires that i8042.debug=1 be enabled)

i8042.direct [HW] Put keyboard port into non-translated\_

↳mode

i8042.dumbkbd [HW] Pretend that controller can only read\_

```

↳data from
        keyboard and cannot control its state
        (Don't attempt to blink the leds)
        i8042.noaux [HW] Don't check for auxiliary (== mouse)
↳port
        i8042.nokbd [HW] Don't check/create keyboard port
        i8042.noloop [HW] Disable the AUX Loopback command while
↳probing
        for the AUX port
        i8042.nomux [HW] Don't check presence of an active
↳multiplexing
        controller
        i8042.nopnp [HW] Don't use ACPIpnp / PnPBIOS to
↳discover KBD/AUX
        controllers
        i8042.timeout [HW] Ignore timeout condition signalled by
↳controller
        i8042.reset [HW] Reset the controller during init,
↳cleanup and
        suspend-to-ram transitions, only
↳during s2r
        transitions, or never reset
        Format: { 1 | Y | y | 0 | N | n }
        1, Y, y: always reset controller
        0, N, n: don't ever reset controller
        Default: only on s2r transitions on x86;
↳most other
        architectures force reset to be always
↳executed
        i8042.unlock [HW] Unlock (ignore) the keylock
        i8042.kbdrset [HW] Reset device connected to KBD port
        i810= [HW,DRM]
        i8k.ignore_dmi [HW] Continue probing hardware even if DMI
↳data
        indicates that the driver is running on
↳unsupported
        hardware.
        i8k.force [HW] Activate i8k driver even if SMM BIOS
↳signature
        does not match list of supported models.
        i8k.power_status [HW] Report power status in /proc/i8k
        (disabled by default)
        i8k.restricted [HW] Allow controlling fans only if SYS_
↳ADMIN
        capability is set.
        i915.invert_brightness=
        [DRM] Invert the sense of the variable that

```

→ is used to set the brightness of the panel backlight. `▬`  
→ Normally a brightness value of 0 indicates backlight `▬`  
→ switched off, and the maximum of the brightness value `▬`  
→ sets the backlight to maximum brightness. If this parameter is `▬`  
→ set to 0 (default) and the machine requires it, or `▬`  
→ this parameter is set to 1, a brightness value of 0 sets `▬`  
→ the backlight to maximum brightness, and the maximum of `▬`  
→ the brightness value switches the backlight off.  
-1 -- never invert brightness  
0 -- machine default  
1 -- force brightness inversion

`icn= [HW,ISDN]`  
Format: <io>[,<membase>[,<icn\_id>[,<icn\_id2>  
→ ]]]

`ide-core.nodma= [HW] (E)IDE subsystem`  
Format: =0.0 to prevent dma on hda, =0.1 `▬`  
→ hdb =1.0 hdc `▬`  
→ .vlb\_clock .pci\_clock .noflush .nohpa . `▬`  
→ noprobe .nowerr `▬`  
→ options `▬`  
→ .cdrom .chs .ignore\_cable are additional `▬`  
→ See Documentation/ide/ide.rst.

`ide-generic.probe-mask= [HW] (E)IDE subsystem`  
Format: <int>  
Probe mask for legacy ISA IDE ports. `▬`  
→ Depending on platform up to 6 ports are supported, `▬`  
→ enabled by setting corresponding bits in the mask to 1. `▬`  
→ The default value is 0x0, which has a special `▬`  
→ meaning. On systems that have PCI, it triggers `▬`  
→ scanning the PCI bus for the first and the second port, `▬`  
→ which are then probed. On systems without PCI, `▬`  
→ the value of 0x0 enables probing the two first ports `▬`  
→ as if it

was 0x3.

```

ide-pci-generic.all-generic-ide [HW] (E)IDE subsystem
Claim all unknown PCI IDE storage
↳ controllers.

idle= [X86]
Format: idle=poll, idle=halt, idle=nomwait
Poll forces a polling idle loop that can
↳ slightly
↳ CPU, but
↳ run hot.
Not recommended.
idle=halt: Halt is forced to be used for
↳ CPU idle.
In such case C2/C3 won't be used again.
idle=nomwait: Disable mwait for CPU C-states

ieee754= [MIPS] Select IEEE Std 754 conformance mode
Format: { strict | legacy | 2008 | relaxed }
Default: strict

↳ execution
↳ supported by
↳ the value
↳ by each
↳ permitted to
↳ the 2008 NaN
encoding
↳ supported
↳ supported
↳ whether

```

Choose which programs will be accepted for  
based on the IEEE 754 NaN encoding(s)  
the FPU and the NaN encoding requested with  
of an ELF file header flag individually set  
binary. Hardware implementations are  
support either or both of the legacy and  
encoding mode.

Available settings are as follows:

```

strict  accept binaries that request a NaN
        supported by the FPU
legacy  only accept legacy-NaN binaries, if
        by the FPU
2008    only accept 2008-NaN binaries, if
        by the FPU
relaxed accept any binaries regardless of
        supported by the FPU

```

↳ both NaN  
↳ or it has  
↳ settings of  
↳ accordingly,  
↳ legacy-NaN and  
↳ legacy-NaN only on  
↳ MIPS32 or  
↳ MIPS64 CPUs.

The FPU emulator is always able to support encodings, so if no FPU hardware is present, been disabled with 'nofpu', then the 'legacy' and '2008' strap the emulator, 'relaxed' straps the emulator for both 2008-NaN, whereas 'strict' enables legacy processors and both NaN encodings on MIPS64 CPUs.

↳ execution  
↳ encoding,

The setting for ABS.fmt/NEG.fmt instruction mode generally follows that for the NaN, except where unsupported by hardware.

ignore\_loglevel [KNL]  
↳ all/  
↳ debugging.  
↳ so users  
↳ loglevel.

Ignore loglevel setting - this will print / kernel messages to the console. Useful for We also add it as printk module parameter, could change it dynamically, usually by /sys/module/printk/parameters/ignore\_

ignore\_rlimit\_data  
↳ mappings,  
↳ changed via  
↳ data.

Ignore RLIMIT\_DATA setting for data, print warning at first misuse. Can be /sys/module/kernel/parameters/ignore\_

ihash\_entries= [KNL]  
Set number of hash buckets for inode cache.

ima\_appraise= [IMA] appraise integrity measurements  
↳ }  
Format: { "off" | "enforce" | "fix" | "log"  
default: "enforce"

ima\_appraise\_tcb [IMA] Deprecated. Use ima\_policy= instead.

↳files                   The builtin appraise policy appraises all  
                           owned by uid=0.

          ima\_canonical\_fmt [IMA]  
 ↳runtime                Use the canonical format for the binary  
                           measurements, instead of host native format.

          ima\_hash=  
 ↳sha384                 [IMA]  
                           Format: { md5 | sha1 | rmd160 | sha256 |  
                                   | sha512 | ... }  
                           default: "sha1"

↳defined                The list of supported hash algorithms is  
                           in crypto/hash\_info.h.

          ima\_policy=  
 ↳setup.                 [IMA]  
                           The builtin policies to load during IMA  
                           Format: "tcb | appraise\_tcb | secure\_boot |  
                                   fail\_securely"

↳exec'd, files         The "tcb" policy measures all programs  
                           mmap'd for exec, and all files opened with  
 ↳the read               mode bit set by either the effective uid  
 ↳(euid=0) or            uid=0.

↳integrity of         The "appraise\_tcb" policy appraises the  
                           all files owned by root.

↳integrity             The "secure\_boot" policy appraises the  
 ↳modules,              of files (eg. kexec kernel image, kernel  
 ↳signatures.            firmware, policy, etc) based on file

↳signature             The "fail\_securely" policy forces file  
 ↳mounted               verification failure also on privileged  
 ↳SIGNATURE             filesystems with the SB\_I\_UNVERIFIABLE\_  
                           flag.

`ima_tcb` [IMA] Deprecated. Use `ima_policy=` instead. Load a policy which meets the needs of the Trusted Computing Base. This means IMA will measure all programs exec'd, files mmap'd for exec, and all files opened for read by uid=0.

`ima_template=` [IMA] Select one of defined IMA measurements. Format: { "ima" | "ima-ng" | "ima-sig" } Default: "ima-ng"

`ima_template_fmt=` [IMA] Define a custom template format. Format: { "field1|...|fieldN" }

`ima.ahash_minsize=` [IMA] Minimum file size for asynchronous hash usage. Format: <min\_file\_size> Set the minimal file size for using asynchronous hash. If left unspecified, ahash usage is disabled. ahash performance varies for different data sizes on different crypto accelerators. This option can be used to achieve the best performance for a particular HW.

`ima.ahash_bufsize=` [IMA] Asynchronous hash buffer size. Format: <bufsize> Set hashing buffer size. Default: 4k. ahash performance varies for different chunk sizes on different crypto accelerators. This option can be used to achieve best performance for particular HW.

`init=` [KNL] Format: <full\_path> Run specified binary instead of /sbin/init as init process.

`initcall_debug` [KNL] Trace initcalls as they are executed.
 

- ↳ Useful for working out where the kernel is dying.
- ↳ during startup.

`initcall_blacklist=` [KNL] Do not execute a comma-separated
 

- ↳ list of initcall functions. Useful for debugging
- ↳ built-in modules and initcalls.

`initrd=` [BOOT] Specify the location of the initial
 

- ↳ ramdisk

`initrdmem=` [KNL] Specify a physical address and size
 

- ↳ from which to load the initrd. If an initrd is compiled
- ↳ in or specified in the bootparams, it takes
- ↳ priority over this setting.

 Format: `ss[KMG],nn[KMG]`  
 Default is 0, 0

`init_on_alloc=` [MM] Fill newly allocated pages and heap
 

- ↳ objects with zeroes.

 Format: 0 | 1  
 Default set by `CONFIG_INIT_ON_ALLOC_DEFAULT_`

- ↳ ON.

`init_on_free=` [MM] Fill freed pages and heap objects with
 

- ↳ zeroes.

 Format: 0 | 1  
 Default set by `CONFIG_INIT_ON_FREE_DEFAULT_`

- ↳ ON.

`init_pkru=` [x86] Specify the default memory protection
 

- ↳ keys rights register contents for all processes.
  - ↳ 0x55555554 by default (disallow access to all but pkey 0).
  - ↳ Can override in debugfs after boot.

`inport.irq=` [HW] Inport (ATI XL and Microsoft) busmouse
 

- ↳ driver

 Format: `<irq>`

`int_pln_enable` [x86] Enable power limit notification.  
↳interrupt

`integrity_audit=[IMA]`  
Format: { "0" | "1" }  
0 -- basic integrity auditing messages.  
↳(Default)  
1 -- additional integrity auditing messages.

`intel_iommu=` [DMAR] Intel IOMMU driver (DMAR) option  
on Enable intel iommu driver.  
off Disable intel iommu driver.

`igfx_off` [Default Off]  
By default, gfx is mapped as normal device.  
↳If a gfx device has a dedicated DMAR unit, the DMAR  
↳unit is bypassed by not enabling DMAR with this  
↳option. In this case, gfx device will use physical  
↳address for DMA.  
`forcedac` [x86\_64]  
↳look With this option iommu will not optimize to  
↳dual for io virtual address below 32-bit forcing  
↳supporting greater address cycle on pci bus for cards  
↳look than 32-bit addressing. The default is to  
↳available for translation below 32-bit and if not  
then look in the higher range.  
`strict` [Default Off]  
↳operation will With this option on every unmap\_single  
↳as opposed result in a hardware IOTLB flush operation  
to batching them for performance.

`sp_off` [Default Off]  
↳Intel IOMMU By default, super page will be supported if  
↳page will has the capability. With this option, super  
not be supported.  
`sm_on` [Default Off]  
↳even if the By default, scalable mode will be disabled

hardware advertises that it has support for `intel_idle` mode translation. With this option set, `intel_idle` will be used on hardware which claims to support it.

`tboot_noforce` [Default Off]  
Do not force the Intel IOMMU enabled under `tboot`. By default, `tboot` will force Intel IOMMU on, which could harm performance of some high-throughput devices like 40Gbit network cards, even if identity mapping is enabled. Note that using this option lowers the security provided by `tboot` because it makes the system vulnerable to DMA attacks.

`nobounce` [Default off]  
Disable bounce buffer for untrusted devices such as the Thunderbolt devices. This will treat the untrusted devices as the trusted ones, hence might expose security risks of DMA attacks.

`intel_idle.max_cstate=` [KNL,HW,ACPI,X86]  
0 disables `intel_idle` and fall back on `acpi_idle`.  
1 to 9 specify maximum depth of C-state.

`intel_pstate=` [X86]  
disable  
Do not enable `intel_pstate` as the default scaling driver for the supported processors.  
passive  
Use `intel_pstate` as a scaling driver, but configure it to work with generic `cpufreq` governors (instead of enabling its internal governor). This mode cannot be used along with the hardware-managed P-states (HWP) feature.  
force

```

    Enable intel_pstate on systems that
    prohibit it by default
    intel_pstate driver
    platform features, such
    that rely on ACPI
    OSPM and therefore
    does not work with
    intel_pstate driver
    instead of acpi-cpufreq.
    no_hwp
    Do not enable hardware P state control
    (HWP)
    if available.
    hwp_only
    Only load intel_pstate on systems which
    support
    hardware P state control (HWP) if
    available.
    support_acpi_ppc
    Enforce ACPI _PPC performance limits. If
    the Fixed ACPI
    Description Table, specifies preferred
    power management
    profile as "Enterprise Server" or
    "Performance Server",
    then this feature is turned on by default.
    per_cpu_perf_limits
    Allow per-logical-CPU P-State performance
    control limits using
    cpufreq sysfs interface

    intremap= [X86-64, Intel-IOMMU]
    on enable Interrupt Remapping (default)
    off disable Interrupt Remapping
    nosid disable Source ID checking
    no_x2apic_optout
    BIOS x2APIC opt-out request will be
    ignored

    nopost disable Interrupt Posting

    iomem= Disable strict checking of access to MMIO
    memory
    strict regions from userspace.
    relaxed
  
```

```

iommu=          [x86]
                off
                force
                noforce
                biomerge
                panic
                nopanic
                merge
                nomerge
                soft
                pt          [x86]
                nopt       [x86]
                nobypass   [PPC/POWERNV]
                Disable IOMMU bypass, using IOMMU for PCI
↳ devices.

iommu.strict=   [ARM64] Configure TLB invalidation behaviour
                Format: { "0" | "1" }
                0 - Lazy mode.
                Request that DMA unmap operations use
↳ deferred
                invalidation of hardware TLBs, for
↳ increased
                throughput at the cost of reduced device
↳ isolation.
                Will fall back to strict mode if not
↳ supported by
                the relevant IOMMU driver.
                1 - Strict mode (default).
                DMA unmap operations invalidate IOMMU
↳ hardware TLBs
                synchronously.

iommu.passthrough=
↳ IOMMU by default.
                [ARM64, X86] Configure DMA to bypass the
                Format: { "0" | "1" }
                0 - Use IOMMU translation for DMA.
                1 - Bypass the IOMMU for DMA.
                unset - Use value of CONFIG_IOMMU_DEFAULT_
↳ PASSTHROUGH.

io7=           [HW] I07 for Marvel based alpha systems
                See comment before marvel_specify_io7 in
                arch/alpha/kernel/core_marvel.c.

io_delay=      [X86] I/O delay method
                0x80
                Standard port 0x80 based delay
                0xed

```

↪some systems) Alternate port 0xed based delay (needed on ↪  
udelay Simple two microseconds delay  
none No delay

ip= [IP\_PNP]  
↪rst. See Documentation/admin-guide/nfs/nfsroot.

ipcmni\_extend [KNL] Extend the maximum number of unique ↪  
↪System V IPC identifiers from 32,768 to 16,777,216.

irqaffinity= [SMP] Set the default irq affinity mask  
↪above. The argument is a cpu list, as described ↪

irqchip.gicv2\_force\_probe= [ARM, ARM64]  
↪page Format: <bool>  
↪range Force the kernel to look for the second 4kB ↪  
of a GICv2 controller even if the memory ↪  
exposed by the device tree is too small.

irqchip.gicv3\_nolpi= [ARM, ARM64]  
↪of Force the kernel to ignore the availability ↪  
↪for system LPIs (and by consequence ITSs). Intended ↪  
↪thus want that use the kernel as a bootloader, and ↪  
↪setting up to let secondary kernels in charge of ↪  
LPIs.

irqchip.gicv3\_pseudo\_nmi= [ARM64]  
↪kernel. This Enables support for pseudo-NMIs in the ↪  
requires the kernel to be built with  
CONFIG\_ARM64\_PSEUDO\_NMI.

irqfixup [HW]  
↪handlers When an interrupt is not handled search all ↪  
↪broken for it. Intended to get systems with badly ↪

firmware running.

`irqpoll` [HW]  
 When an interrupt is not handled search all handlers for it. Also check all handlers each timer interrupt. Intended to get systems with badly broken firmware running.

`isapnp=` [ISAPNP]  
 Format: <RDP>,<reset>,<pci\_scan>,<verbosity>

`isolcpus=` [KNL,SMP,ISOL] Isolate a given set of CPUs from disturbance.  
 [Deprecated - use cpusets instead]  
 Format: [flag-list,]<cpu-list>

Specify one or more CPUs to isolate from disturbances specified in the flag list (default: domain):

`nohz`  
 Disable the tick when a single task runs. A residual 1Hz tick is offloaded to workqueues, which you need to affine to housekeeping through the global workqueue's affinity configured via the `/sys/devices/virtual/workqueue/cpumask` file, or by using the 'domain' flag described below.

NOTE: by default the global workqueue runs on all CPUs, so to protect individual CPUs the 'cpumask' file has to be configured manually after bootup.

`domain`  
 Isolate from the general SMP balancing and scheduling algorithms. Note that performing domain isolation this way is irreversible: it's not possible to bring back a CPU to the domains once isolated through `isolcpus`. It's strongly

→ scheduler load advised to use cpusets instead to disable,  
→ "balance" file. balancing through the "cpuset.sched\_load\_  
→ where CPUs can It offers a much more flexible interface,  
→ anytime. move in and out of an isolated set.

→ "isolated" CPU via You can move a process onto or off an  
→ value is the CPU affinity syscalls or cpuset.  
<cpu number> begins at 0 and the maximum,  
"number of CPUs in system - 1".

managed\_irq

→ interrupts Isolate from being targeted by managed,  
→ isolated which have an interrupt mask containing,  
→ is CPUs. The affinity of managed interrupts,  
→ changed via handled by the kernel and cannot be,  
the /proc/irq/\* interfaces.

→ effective This isolation is best effort and only,  
→ mask of a if the automatically assigned interrupt,  
→ housekeeping device queue contains isolated and,  
→ then such CPUs. If housekeeping CPUs are online,  
→ housekeeping CPU interrupts are directed to the,  
→ CPU so that IO submitted on the housekeeping,  
cannot disturb the isolated CPU.

→ isolated If a queue's affinity mask contains only,  
→ the CPUs then this parameter has no effect on,  
→ interrupts are interrupt routing decision, though,  
only delivered when tasks running on those  
isolated CPUs submit IO. IO submitted on  
housekeeping CPUs has no influence on,

→those queues.

The format of <cpu-list> is described above.

iucv= [HW,NET]

ivrs\_ioapic [HW,X86\_64]  
Provide an override to the IOAPIC-ID<->

→DEVICE-ID mapping provided in the IVRS ACPI table. For example, to map IOAPIC-ID decimal 10 to PCI device 00:14.0 write the parameter as:  
ivrs\_ioapic[10]=00:14.0

ivrs\_hpet [HW,X86\_64]  
Provide an override to the HPET-ID<->

→DEVICE-ID mapping provided in the IVRS ACPI table. For example, to map HPET-ID decimal 0 to PCI device 00:14.0 write the parameter as:  
ivrs\_hpet[0]=00:14.0

ivrs\_acpihid [HW,X86\_64]  
Provide an override to the ACPI-HID:UID<->

→DEVICE-ID mapping provided in the IVRS ACPI table. For example, to map UART-HID:UID AMD0020:0 to PCI device 00:14.5 write the parameter as:  
ivrs\_acpihid[00:14.5]=AMD0020:0

js= [HW,JOY] Analog joystick  
See Documentation/input/joydev/joystick.rst.

nokaslr [KNL]  
When CONFIG\_RANDOMIZE\_BASE is set, this  
→disables kernel and module base offset ASLR (Address  
→Space Layout Randomization).

kasan\_multi\_shot [KNL] Enforce KASAN (Kernel Address  
→Sanitizer) to print report on every invalid memory access.  
→Without this parameter KASAN will print report only for  
→the first invalid access.

keepinitrd [HW,ARM]

`kernelcore=` [KNL,X86,IA-64,PPC]  
Format: nn[KMGTPe] | nn% | "mirror"  
This parameter specifies the amount of memory usable by the kernel for non-movable allocations. The requested amount is spread evenly throughout all nodes in the system as ZONE\_NORMAL. The remaining memory is used for ZONE\_MOVABLE. In the event, a node is too small to have both ZONE\_NORMAL and ZONE\_MOVABLE, kernelcore memory will take priority and other nodes will have a larger ZONE\_MOVABLE. ZONE\_MOVABLE is used for the allocation of pages that may be reclaimed or moved by the page migration subsystem. Note that allocations like PTEs-from-HighMem and the Normal zone if it does not.

It is possible to specify the exact amount of memory in the form of "nn[KMGTPe]", a percentage of total system memory in the form of "nn%", or "mirror". If "mirror" option is specified, mirrored (reliable) memory is used for non-movable allocations and remaining memory is used for Movable pages. "nn[KMGTPe]", "nn%", and "mirror" are exclusive, so you cannot specify multiple forms.

`kgdbdbg=` [KGDB,Hw] kgdb over EHCI usb debug port.  
Format: <Controller#>[,poll interval]  
The controller # is the number of the ehci port as it is probed via PCI. The poll interval is optional and is the number seconds in

↳between  
 ↳you need  
 ↳kernel with  
 ↳When  
 ↳break into

each poll cycle to the debug port in case,  
 the functionality for interrupting the,  
 gdb or control-c on the dbgp connection.   
 not using this parameter you use sysrq-g to,  
 the kernel debugger.

kgdboc= [KGDB,HW] kgdb over consoles.  
 Requires a tty driver that supports console,  
 ↳polling,  
 ↳(non-usb).  
 ↳device>[,baud]

Serial only format: <serial\_device>[,baud]  
 keyboard only format: kbd  
 keyboard and serial format: kbd,<serial\_  
 ↳dev>[,baud]

Optional Kernel mode setting:  
 kms, kbd format: kms,kbd  
 kms, kbd and serial format: kms,kbd,<ser\_  
 ↳dev>[,baud]

kgdboc\_earlycon= [KGDB,HW]

↳read  
 ↳you can use  
 ↳backend  
 ↳Intended to  
 ↳which  
 ↳to.

↳specified  
 ↳the name of  
 ↳the tty  
 ↳value  
 ↳implements

If the boot console provides the ability to,  
 characters and can work in polling mode,  
 this parameter to tell kgdb to use it as a,  
 until the normal console is registered.  
 be used together with the kgdboc parameter,  
 specifies the normal console to transition,  
 ↳to.

The name of the early console should be,  
 as the value of this parameter. Note that,  
 the early console might be different than,  
 name passed to kgdboc. It's OK to leave the,  
 blank and the first boot console that,  
 read() will be picked.

`kgdbwait` [KGDB] Stop kernel execution and enter the kernel debugger at the earliest opportunity.

`kmac=` [MIPS] korina ethernet MAC address. Configure the RouterBoard 532 series on-chip Ethernet adapter MAC address.

`kmemleak=` [KNL] Boot-time kmemleak enable/disable  
Valid arguments: on, off  
Default: on  
Built with CONFIG\_DEBUG\_KMEMLEAK\_DEFAULT\_↵  
↵OFF=y,  
the default is off.

`kprobe_event=[probe-list]` [FTRACE] Add kprobe events and enable at ↵  
↵boot time.  
↵list of probe  
↵kprobe\_events  
↵delimited.  
↵read with  
The probe-list is a semicolon delimited ↵  
definitions. Each definition is same as ↵  
interface, but the parameters are comma ↵  
For example, to add a kprobe event on `vfs_`  
`arg1` and `arg2`, add to the command line;  
`kprobe_event=p,vfs_read,$arg1,$arg2`  
See also Documentation/trace/kprobetrace.  
↵rst "Kernel  
Boot Parameter" section.

`kpti=` [ARM64] Control page table isolation of user and kernel address spaces.  
Default: enabled on cores which need ↵  
↵mitigation.  
0: force disabled  
1: force enabled

`kvm.ignore_msrs=[KVM]` Ignore guest accesses to unhandled ↵  
↵MSRs.  
Default is 0 (don't ignore, but inject #GP)

`kvm.enable_vmware_backdoor=[KVM]` Support VMware backdoor PV ↵  
↵interface.  
Default is false (don't support).

`kvm.mmu_audit=` [KVM] This is a R/W parameter which allows ↵  
↵audit  
KVM MMU at runtime.

Default is 0 (off)

`kvm.nx_huge_pages=`  
 [KVM] Controls the software workaround for the  
 X86\_BUG\_ITLB\_MULTIHIT bug.  
 force : Always deploy workaround.  
 off : Never deploy workaround.  
 auto : Deploy workaround based on the presence of  
 X86\_BUG\_ITLB\_MULTIHIT.  
 Default is 'auto'.  
 If the software workaround is enabled for the host,  
 guests do need not to enable it for nested guests.

`kvm.nx_huge_pages_recovery_ratio=`  
 [KVM] Controls how many 4KiB pages are periodically zapped  
 back to huge pages. 0 disables the recovery, otherwise if  
 the value is N KVM will zap 1/Nth of the 4KiB pages every  
 minute. The default is 60.

`kvm-amd.nested=` [KVM,AMD] Allow nested virtualization in KVM/SVM.

Default is 1 (enabled)

`kvm-amd.npt=` [KVM,AMD] Disable nested paging for all guests.  
 (virtualized MMU)  
 Default is 1 (enabled) if in 64-bit or 32-bit PAE mode.

`kvm-arm.vgic_v3_group0_trap=`  
 [KVM,ARM] Trap guest accesses to GICv3 group-0  
 system registers

`kvm-arm.vgic_v3_group1_trap=`  
 [KVM,ARM] Trap guest accesses to GICv3 group-1  
 system registers

`kvm-arm.vgic_v3_common_trap=`  
 [KVM,ARM] Trap guest accesses to GICv3 common

system registers

`kvm-arm.vgic_v4_enable=`  
[KVM,ARM] Allow use of GICv4 for direct  
→ injection of LPIs.

`kvm-intel.ept=` [KVM,Intel] Disable extended page tables  
→ chips. (virtualized MMU) support on capable Intel  
Default is 1 (enabled)

`kvm-intel.emulate_invalid_guest_state=`  
→ guest states [KVM,Intel] Enable emulation of invalid  
Default is 0 (disabled)

`kvm-intel.flexpriority=`  
→ (TPR shadow). [KVM,Intel] Disable FlexPriority feature  
Default is 1 (enabled)

`kvm-intel.nested=`  
[KVM,Intel] Enable VMX nesting (nVMX).  
Default is 0 (disabled)

`kvm-intel.unrestricted_guest=`  
→ feature [KVM,Intel] Disable unrestricted guest  
→ capable (virtualized real and unpaged mode) on  
Intel chips. Default is 1 (enabled)

`kvm-intel.vmentry_lld_flush=`[KVM,Intel] Mitigation for L1  
→ Terminal Fault  
CVE-2018-3620.  
Valid arguments: never, cond, always  
always: L1D cache flush on every VMENTER.  
cond: Flush L1D on VMENTER only when the  
→ code between VMEXIT and VMENTER can leak host  
→ memory.  
never: Disables the mitigation  
Default is cond (do L1 cache flush in  
→ specific instances)

`kvm-intel.vpid=` [KVM,Intel] Disable Virtual Processor  
→ Identification

feature (tagged TLBs) on capable Intel chips.

Default is 1 (enabled)

`l1tf=`  
[X86] Control mitigation of the L1TF affected CPUs

unconditionally

The kernel PTE inversion protection is enabled and cannot be disabled.

full

Provides all available mitigations for the L1TF vulnerability. Disables SMT and enables all mitigations in the hypervisors, i.e. unconditional L1D flush.

via the sysfs interface is still possible after boot. Hypervisors will issue a warning when the first VM is started in a potentially insecure configuration, i.e. SMT enabled or L1D flush disabled.

full,force

Same as 'full', but disables SMT and L1D flush runtime control. Implies the 'nosmt=force' command line option. (i.e. sysfs control of SMT is disabled.)

flush

Leaves SMT enabled and enables the hypervisor mitigation, i.e. L1D flush.

via the sysfs interface is still possible after boot. Hypervisors will issue a

→warning  
when the first VM is started in a potentially insecure configuration, i.e. SMT enabled or L1D flush.

→disabled.

flush,nosmt

Disables SMT and enables the default hypervisor mitigation.

→via the  
→after  
→warning  
→disabled.

SMT control and L1D flush control via the sysfs interface is still possible after boot. Hypervisors will issue a warning when the first VM is started in a potentially insecure configuration, i.e. SMT enabled or L1D flush.

flush,nowarn

→will not  
→potentially

Same as 'flush', but hypervisors will not warn when a VM is started in a potentially insecure configuration.

off

→doesn't  
→available  
→hypervisor and

Disables hypervisor mitigations and does not emit any warnings. It also drops the swap size and RAM limit restriction on both hypervisor and bare metal.

Default is 'flush'.

→hw-vuln/l1tf.rst

For details see: Documentation/admin-guide/hw-vuln/l1tf.rst

l2cr= [PPC]

l3cr= [PPC]

→BIOS lapic [X86-32,APIC] Enable the local APIC even if disabled it.

`lapic=` [x86,APIC] "notscdeadline" Do not use TSC<sub>␣</sub>  
`→deadline` value for LAPIC timer one-shot<sub>␣</sub>  
`→implementation. Default` back to the programmable timer unit in the<sub>␣</sub>  
`→LAPIC.`

`lapic_timer_c2_ok` [X86,APIC] trust the local apic<sub>␣</sub>  
`→timer` in C2 power state.

`libata.dma=` [LIBATA] DMA control  
`→DMA` `libata.dma=0` Disable all PATA and SATA<sub>␣</sub>  
`→only` `libata.dma=1` PATA and SATA Disk DMA<sub>␣</sub>  
`→enables DMA` `libata.dma=2` ATAPI (CDROM) DMA only  
`libata.dma=4` Compact Flash DMA only  
Combinations also work, so `libata.dma=3`<sub>␣</sub>  
for disks and CDROMs, but not CFs.

`libata.ignore_hpa=` [LIBATA] Ignore HPA limit  
`→(default)` `libata.ignore_hpa=0` keep BIOS limits<sub>␣</sub>  
`→using full disk` `libata.ignore_hpa=1` ignore limits,<sub>␣</sub>

`libata.noacpi` [LIBATA] Disables use of ACPI in libata<sub>␣</sub>  
`→suspend/resume` when set.  
Format: <int>

`libata.force=` [LIBATA] Force configurations. The format<sub>␣</sub>  
`→is comma` separated list of "[ID:]VAL" where ID is  
`→numbers` PORT[.DEVICE]. PORT and DEVICE are decimal<sub>␣</sub>  
`→it matches` matching port, link or device. Basically,<sub>␣</sub>  
`→libata. If` the ATA ID string printed on console by<sub>␣</sub>  
`→and DEVICE` the whole ID part is omitted, the last PORT<sub>␣</sub>  
`→specified yet, the` values are used. If ID hasn't been<sub>␣</sub>  
`→and devices.` configuration applies to all ports, links<sub>␣</sub>  
If only DEVICE is omitted, the parameter<sub>␣</sub>

→applies to  
→it. DEVICE  
→or the  
→does not  
→selects the  
→force. As long  
→is allowed.  
→for 1.5Gbps.  
→or sata.  
→udma[0-7].  
→is also  
→support  
→configurations changing

the port and all links and devices behind,  
number of 0 either selects the first device,  
first fan-out link behind PMP device. It  
select the host link. DEVICE number of 15,  
host link and device attached to it.

The VAL specifies the configuration to,  
as there's no ambiguity shortcut notation,  
For example, both 1.5 and 1.5G would work,  
The following configurations can be forced.

- \* Cable type: 40c, 80c, short40c, unk, ign,  
Any ID with matching PORT is used.
- \* SATA link speed limit: 1.5Gbps or 3.0Gbps.
- \* Transfer mode: pio[0-7], mwdma[0-4] and,  
udma[/][16,25,33,44,66,100,133] notation,  
allowed.
- \* [no]ncq: Turn on or off NCQ.
- \* [no]ncqtrim: Turn off queued DSM TRIM.
- \* nohrst, nosrst, norst: suppress hard, soft  
and both resets.
- \* rstone: only attempt one reset during  
hot-unplug link recovery
- \* dump\_id: dump IDENTIFY data.
- \* atapi\_dmdir: Enable ATAPI DMADIR bridge,  
→support
- \* disable: Disable this device.

If there are multiple matching,  
→configurations changing  
the same attribute, the last one is used.

`memblock=debug` [KNL] Enable memblock debug messages.

`load_ramdisk=` [RAM] List of ramdisks to load from floppy  
 See Documentation/admin-guide/blockdev/  
`ramdisk.rst`.

`lockd.nlm_grace_period=P` [NFS] Assign grace period.  
 Format: <integer>

`lockd.nlm_tcpport=N` [NFS] Assign TCP port.  
 Format: <integer>

`lockd.nlm_timeout=T` [NFS] Assign timeout value.  
 Format: <integer>

`lockd.nlm_udpport=M` [NFS] Assign UDP port.  
 Format: <integer>

`lockdown=` [SECURITY]  
 { integrity | confidentiality }  
 Enable the kernel lockdown feature. If set,

`to`  
`userland to`  
`set to`  
`userland`  
`the kernel`  
 integrity, kernel features that allow  
 modify the running kernel are disabled. If  
 confidentiality, kernel features that allow  
 to extract confidential information from  
 are also disabled.

`locktorture.nreaders_stress=` [KNL]  
`kthreads.`  
`on the`  
 Set the number of locking read-acquisition  
 Defaults to being automatically set based  
 number of online CPUs.

`locktorture.nwriters_stress=` [KNL]  
`kthreads.`  
 Set the number of locking write-acquisition

`locktorture.onoff_holdoff=` [KNL]  
`testing.`  
 Set time (s) after boot for CPU-hotplug

`locktorture.onoff_interval=` [KNL]  
`or`  
 Set time (s) between CPU-hotplug operations,

zero to disable CPU-hotplug testing.

`locktorture.shuffle_interval= [KNL]`  
Set task-shuffle interval (jiffies). [↪](#)

[↪](#)Shuffling tasks allows some CPUs to go into [↪](#)

[↪](#)dyntick-idle mode during the locktorture test.

`locktorture.shutdown_secs= [KNL]`  
Set time (s) after boot system shutdown. [↪](#)

[↪](#)This is useful for hands-off automated testing.

`locktorture.stat_interval= [KNL]`  
Time (s) between statistics `printk()`s.

`locktorture.stutter= [KNL]`  
Time (s) to stutter testing, for example, specifying five seconds causes the test to [↪](#)

[↪](#)run for five seconds, wait for five seconds, and so [↪](#)

[↪](#)on. This tests the locking primitive's ability [↪](#)

[↪](#)to transition abruptly to and from idle.

`locktorture.torture_type= [KNL]`  
Specify the locking implementation to test.

`locktorture.verbose= [KNL]`  
Enable additional `printk()` statements.

`logibm.iirq= [HW,MOUSE]` Logitech Bus Mouse Driver  
Format: `<iirq>`

`loglevel=` All Kernel Messages with a loglevel smaller [↪](#)

[↪](#)than the console loglevel will be printed to the [↪](#)

[↪](#)console. It can also be changed with `klogd` or other [↪](#)

[↪](#)programs. The loglevels are defined as follows:

<a href="#">↪</a> taken immediately	0 (KERN_EMERG)	system is unusable
	1 (KERN_ALERT)	action must be <a href="#">↪</a>
	2 (KERN_CRIT)	critical conditions
	3 (KERN_ERR)	error conditions
	4 (KERN_WARNING)	warning conditions
	5 (KERN_NOTICE)	normal but <a href="#">↪</a>

↪ significant condition
 

6 (KERN_INFO)	informational
7 (KERN_DEBUG)	debug-level messages

↪ `log_buf_len=n[KMG]` Sets the size of the printk ring
   
 ↪ buffer,
   
 ↪ greater
   
 ↪ defined
   
 ↪ There is
   
 ↪ parameter
   
 ↪ depending on
   
 ↪ more details.
   
 in bytes. `n` must be a power of two and
   
 than the minimal size. The minimal size is
   
 by `LOG_BUF_SHIFT` kernel config parameter.
   
 also `CONFIG_LOG_CPU_MAX_BUF_SHIFT` config
   
 that allows to increase the default size
   
 the number of CPUs. See `init/Kconfig` for

↪ `logo.nologo` [FB] Disables display of the built-in Linux
   
 ↪ logo.
   
 ↪ space for
   
 ↪ debugging
   
 This may be used to provide more screen
   
 kernel log messages and is useful when
   
 kernel boot problems.

↪ `lp=0` [LP] Specify parallel ports to use, e.g,
   
 ↪ `lp=port[,port...]` `lp=none,parport0` (`lp0` not
   
 ↪ configured, `lp1` uses
   
 ↪ `lp=reset`
  
 ↪ disables the
   
 ↪ `lp=auto`
  
 ↪ can be
   
 ↪ causes
   
 ↪ parallel ports
   
 ↪ starting with
   
 ↪ 'none' to skip
   
 ↪ such as
   
 ↪ instead of a
   
 ↪ device IDs
   
 ↪ to see if
   
 first parallel port). '`lp=0`'
   
 printer driver. '`lp=reset`' (which
   
 specified in addition to the ports)
   
 attached printers to be reset. Using
   
`lp=port1,port2,...` specifies the
   
 to associate lp devices with,
   
`lp0`. A port specification may be
   
 that lp device, or a parport name
   
'`parport0`'. Specifying '`lp=auto`'
   
port specification list means that
   
from each port should be examined,

↪ attached; if  
↪ printer.  
↪ C.

↪ lpj=n [KNL]  
↪ thus avoiding Sets loops\_per\_jiffy to given constant,  
↪ to 250 ms per time-consuming boot-time autodetection (up  
↪ determine CPU). 0 enables autodetection (default). To  
↪ with normal the correct value for your kernel, boot  
↪ Note that autodetection and see what value is printed.  
↪ to all CPUs, on SMP systems the preset will be applied  
↪ CPUs need which is likely to cause problems if your  
↪ incorrect value significantly divergent settings. An  
↪ leading to will cause delays in the kernel to be wrong,  
↪ Although unpredictable I/O errors and other breakage.  
↪ damage your unlikely, in the extreme case this might  
hardware.

↪ ltpc= [NET]  
Format: <io>,<irq>,<dma>

↪ lsm.debug [SECURITY] Enable LSM initialization  
↪ debugging output.

↪ lsm=lsm1,...,lsmN [SECURITY] Choose order of LSM  
↪ initialization. This overrides CONFIG\_LSM, and the "security="   
↪ parameter.

↪ machvec= [IA-64] Force the use of a particular  
↪ machine-vector (machvec) in a generic kernel.  
Example: machvec=hpzx1

↪ machtype= [Loongson] Share the same kernel image file  
↪ between different

yeeloong laptop.  
 Example: machtype=lemote-yeeloong-2f-7inch

`max_addr=nn[KMG]` [KNL,B00T,ia64] All physical memory `greater` than or equal to this physical address is `ignored`.

`maxcpus=` [SMP] Maximum number of processors that an `SMP kernel` will bring up during bootup. `maxcpus=n : n` `>= 0` limits the kernel to bring up 'n' processors. `Surely after` bootup you can bring up the other plugged `cpu` by executing `online`". So `maxcpus` `equivalent to "nosmp"`, which also disables the IO APIC.

`max_loop=` [LOOP] The number of loop block devices `that get` (loop.max\_loop) unconditionally pre-created at init time. `The default` number is configured by `BLK_DEV_LOOP_MIN_COUNT`. Instead `number, loop` devices can be requested on-demand with the `/dev/loop-control` interface.

`mce` [X86-32] Machine Check Exception

`mce=option` [X86-64] See Documentation/x86/x86\_64/  
`boot-options.rst`

`md=` [HW] RAID subsystems devices and level  
 See Documentation/admin-guide/md.rst.

`mdacon=` [MDA]  
 Format: <first>,<last>  
 Specifies range of consoles to be captured `by the MDA`.

`mds=` [X86,INTEL]  
 Control mitigation for the `Micro-architectural Data Sampling (MDS) vulnerability`.

↪against CPU Certain CPUs are vulnerable to an exploit,  
↪information to a internal buffers which can forward,  
disclosure gadget under certain conditions.  
↪channel In vulnerable processors, the speculatively  
↪attacker does forwarded data can be used in a cache side  
attack, to access data to which the  
not have direct access.  
↪The This parameter controls the MDS mitigation.  
options are:  
↪vulnerable CPUs full - Enable MDS mitigation on  
↪disable full,nosmt - Enable MDS mitigation and  
SMT on vulnerable CPUs  
↪mitigation off - Unconditionally disable MDS  
↪prevented by On TAA-affected machines, mds=off can be  
↪vulnerabilities are an active TAA mitigation as both  
↪order to disable mitigated with the same mechanism so in  
↪async\_abort=off this mitigation, you need to specify tsx\_  
too.  
Not specifying this option is equivalent to  
mds=full.  
↪hw-vuln/mds.rst For details see: Documentation/admin-guide/  
mem=nn[KMG] [KNL,B00T] Force usage of a specific amount  
↪of memory Amount of memory to be used in cases as  
↪follows:  
1 for test;  
↪whole system memory; 2 when the kernel is not able to see the  
3 memory that lies after 'mem=' boundary is

↪excluded from the hypervisor, then assigned to KVM.  
 ↪guests.

↪together [X86] Work as limiting max address. Use  
 ↪space collisions. with memmap= to avoid physical address.  
 ↪at addresses Without memmap= PCI devices could be placed  
 belonging to unused RAM.

↪boot time since Note that this only takes effects during  
 ↪added after boot in above case 3, memory may need be hot  
 ↪sufficient. if system memory of hypervisor is not

mem=nopentium [BUGS=X86-32] Disable usage of 4MB pages  
 ↪for kernel memory.

memchunk=nn[KMG] [KNL,SH] Allow user to override the default  
 ↪size for per-device physically contiguous DMA  
 ↪buffers.

memhp\_default\_state=online/offline [KNL] Set the initial state for the memory  
 ↪hotplug onlining policy. If not specified, the  
 ↪default value is set according to the  
 CONFIG\_MEMORY\_HOTPLUG\_DEFAULT\_ONLINE kernel  
 ↪config option.  
 See Documentation/admin-guide/mm/  
 ↪memory-hotplug.rst.

memmap=exactmap [KNL,X86] Enable setting of an exact  
 ↪constructed based on E820 memory map, as specified by the user.  
 Such memmap=exactmap lines can be  
 ↪memmap=nn@ss BIOS output or other requirements. See the  
 option description.

memmap=nn[KMG]@ss[KMG] [KNL] Force usage of a specific region of

→memory. Region of memory to be used is from ss to

→ss+nn. If @ss[KMG] is omitted, it is equivalent to

→mem=nn[KMG], which limits max address to nn[KMG]. Multiple different regions can be specified, comma delimited. Example:

```
memmap=100M@2G,100M#3G,1G!1024G
```

memmap=nn[KMG]#ss[KMG]  
[KNL,ACPI] Mark specific memory as ACPI

→data. Region of memory to be marked is from ss to

→ss+nn.

```
memmap=nn[KMG]$ss[KMG]
```

[KNL,ACPI] Mark specific memory as reserved. Region of memory to be reserved is from ss

→to ss+nn. Example: Exclude memory from

→0x18690000-0x1869ffff

```
memmap=64K$0x18690000
or
memmap=0x10000$0x18690000
```

Some bootloaders may need an escape

→character before '\$', like Grub2, otherwise '\$' and the following

→number will be eaten.

```
memmap=nn[KMG]!ss[KMG]
```

[KNL,X86] Mark specific memory as protected. Region of memory to be used, from ss to

→ss+nn. The memory region may be marked as e820

→type 12 (0xc) and is NVDIMM or ADR memory.

```
memmap=<size>%<offset>-<oldtype>+<newtype>
```

[KNL,ACPI] Convert memory within the

→specified region from <oldtype> to <newtype>. If "-<oldtype>

→" is left out, the whole region will be marked as

→<newtype>, even if previously unavailable. If "+

→<newtype>" is left out, matching memory will be removed. Types

→are

- specified as e820 types, e.g., 1 = RAM, 2 = reserved,  
3 = ACPI, 12 = PRAM.
- `memory_corruption_check=0/1` [X86]  
Some BIOSes seem to corrupt the first 64k of memory when doing things like suspend/
- `resume.`  
Setting this option will scan the memory looking for corruption. Enabling this will both detect corruption and prevent the
- `kernel`  
from using the memory being corrupted. However, its intended as a diagnostic tool;
- `if`  
repeatable BIOS-originated corruption always affects the same memory, you can use `memmap=` to prevent the kernel from using that
- `memory.`
- `memory_corruption_check_size=size` [X86]  
By default it checks for corruption in the
- `low`  
64k, making this memory unavailable for
- `normal`  
use. Use this parameter to scan for corruption in more or less memory.
- `memory_corruption_check_period=seconds` [X86]  
By default it checks for corruption every 60 seconds. Use this parameter to check at
- `some`  
other rate. 0 disables periodic checking.
- `memtest=` [KNL,X86,ARM,PPC] Enable memtest  
Format: <integer>  
default : 0 <disable>  
Specifies the number of memtest passes to be performed. Each pass selects another test pattern from a given set of patterns.
- `Memtest`  
fills the memory with this pattern,
- `validates`  
memory contents and reserves bad memory regions that are detected.
- `mem_encrypt=` [X86-64] AMD Secure Memory Encryption (SME)
- `control`  
Valid arguments: on, off  
Default (depends on kernel configuration
- `option):`

```

on (CONFIG_AMD_MEM_ENCRYPT_ACTIVE_BY_
↪DEFAULT=y)
off (CONFIG_AMD_MEM_ENCRYPT_ACTIVE_BY_
↪DEFAULT=n)
mem_encrypt=on:          Activate SME
mem_encrypt=off:         Do not activate SME

Refer to Documentation/virt/kvm/
↪amd-memory-encryption.rst
for details on when memory encryption can
↪be activated.

mem_sleep_default=      [SUSPEND] Default system suspend
↪mode:
s2idle - Suspend-To-Idle
shallow - Power-On Suspend or equivalent
↪(if supported)
deep - Suspend-To-RAM or equivalent (if
↪supported)
See Documentation/admin-guide/pm/
↪sleep-states.rst.

meye.*=                 [HW] Set MotionEye Camera parameters
See Documentation/admin-guide/media/meye.
↪rst.

mfgpt_irq=              [IA-32] Specify the IRQ to use for the
Multi-Function General Purpose Timers on
↪AMD Geode
platforms.

mfgptfix                [X86-32] Fix MFGPT timers on AMD Geode
↪platforms when
the BIOS has incorrectly applied a
↪workaround. TinyBIOS
version 0.98 is known to be affected, 0.99
↪fixes the
problem by letting the user disable the
↪workaround.

mga=                    [HW,DRM]

min_addr=nn[KMG]       [KNL,B00T,ia64] All physical memory
↪below this
physical address is ignored.

mini2440=               [ARM,HW,KNL]
Format:[0..2][b][c][t]
Default: "0tb"
MINI2440 configuration specification:
0 - The attached screen is the 3.5" TFT

```

1 - The attached screen is the 7" TFT  
 2 - The VGA Shield is attached (1024x768)  
 Leaving out the screen size parameter will

↳not load  
 ↳left  
 ↳will be  
 ↳a GPIO  
 ↳using the  
 ↳support. The  
 ↳mainstream  
 ↳be found  
 ↳kernel at

mitigations=  
 ↳mitigations for  
 ↳curated,  
 ↳an  
 ↳options.

↳mitigations. This  
 ↳may also  
 ↳vulnerabilities.

↳PPC]  
 ↳PPC,S390,ARM64]  
 ↳[X86]

1 - The attached screen is the 7" TFT  
 2 - The VGA Shield is attached (1024x768)  
 Leaving out the screen size parameter will

the TFT driver, and the framebuffer will be

unconfigured.  
 b - Enable backlight. The TFT backlight pin

linked to the kernel VESA blanking code and

LED. This parameter is not necessary when

VGA shield.  
 c - Enable the s3c camera interface.  
 t - Reserved for enabling touchscreen

touchscreen support is not enabled in the

kernel as of 2.6.30, a preliminary port can

in the "bleeding edge" mini2440 support

<http://repo.or.cz/w/linux-2.6/mini2440.git>

[X86,PPC,S390,ARM64] Control optional

CPU vulnerabilities. This is a set of

arch-independent options, each of which is

aggregation of existing arch-specific

off

Disable all optional CPU

improves system performance, but it

expose users to several CPU

Equivalent to: nopti [X86,PPC]  
 kpti=0 [ARM64]  
 nospectre\_v1 [X86,  
 nobp=0 [S390]  
 nospectre\_v2 [X86,  
 spectre\_v2\_user=off

```

→disable=off [X86,PPC]
→[ARM64]
→[X86]
→pages=off [X86]

```

Exceptions:

```

→any effect on
→when
→pages=force.

```

auto (default)  
Mitigate all CPU vulnerabilities,  
enabled, even if it's vulnerable.   
users who don't want to be  
getting disabled across kernel  
have other ways of avoiding  
Equivalent to: (default behavior)

```

→but leave SMT
→This is for
→surprised by SMT
→upgrades, or who
→SMT-based attacks.

```

auto,nosmt  
Mitigate all CPU vulnerabilities,  
if needed. This is for users who  
be fully mitigated, even if it  
Equivalent to: l1tf=flush,nosmt  
mds=full,nosmt [X86]  
tsx\_async\_abort=full,

```

→disabling SMT
→always want to
→means losing SMT.
→[X86]
→nosmt [X86]

```

```

mminit_loglevel=
→this
→verbosity for

```

[KNL] When CONFIG\_DEBUG\_MEMORY\_INIT is set,  
parameter allows control of the logging  
the additional memory initialisation checks.

↪ A value of 0 disables mminit logging and a level of ↪  
 ↪4 will log everything. Information is printed at ↪  
 ↪KERN\_DEBUG so loglevel=8 may also need to be specified.

module.sig\_enforce [KNL] When CONFIG\_MODULE\_SIG is set, this ↪  
 ↪means that modules without (valid) signatures will ↪  
 ↪fail to load. Note that if CONFIG\_MODULE\_SIG\_FORCE is set,  
 ↪ that is always true, so this option does nothing.

module\_blacklist= [KNL] Do not load a comma-separated list ↪  
 ↪of modules. Useful for debugging problem ↪  
 ↪modules.

mousedev.tap\_time= [MOUSE] Maximum time between finger ↪  
 ↪touching and leaving touchpad surface for touch to be ↪  
 ↪considered a tap and be reported as a left button ↪  
 ↪click (for touchpads working in absolute mode only).  
 Format: <msecs>

mousedev.xres= [MOUSE] Horizontal screen resolution, used ↪  
 ↪for devices reporting absolute coordinates, such as ↪  
 ↪tablets  
 mousedev.yres= [MOUSE] Vertical screen resolution, used ↪  
 ↪for devices reporting absolute coordinates, such as ↪  
 ↪tablets

movablecore= [KNL,X86,IA-64,PPC]  
 Format: nn[KMGTPe] | nn%  
 This parameter is the complement to ↪  
 ↪kernelcore=, it specifies the amount of memory used for ↪  
 ↪migratable allocations. If both kernelcore and ↪  
 ↪movablecore is specified, then kernelcore will be at ↪  
 ↪\*least\* the specified value but may be more. If ↪  
 ↪movablecore on its

own is specified, the administrator must be careful that the amount of memory usable for all allocations is not too small.

`movable_node` [KNL] Boot-time switch to make hotpluggable memory NUMA nodes to be movable. This means that the memory of such nodes will be usable only for movable allocations which rules out almost all kernel allocations. Use with caution!

`MTD_Partition=` [MTD]  
Format: <name>,<region-number>,<size>,<offset>

`MTD_Region=` [MTD] Format:  
<name>,<region-number>[,<base>,<size>,<buswidth>,<altbuswidth>]

`mtdparts=` [MTD]  
See drivers/mtd/parsers/cmdlinepart.c

`multitce=off` [PPC] This parameter disables the use of the pSeries firmware feature for updating multiple TCE entries at a time.

`onenand.bdry=` [HW,MTD] Flex-OneNAND Boundary Configuration  
Format: [die0\_boundary][,<die0\_lock>][,<die1\_boundary>][,<die1\_lock>]  
boundary - index of last SLC block on Flex-OneNAND. The remaining blocks are configured as MLC blocks.  
lock - Configure if Flex-OneNAND boundary should be locked. Once locked, the boundary cannot be changed.  
1 indicates lock status, 0 indicates unlock status.

`mtdset=` [ARM]  
ARM/S3C2412 JIVE boot control

See arch/arm/mach-s3c2412/mach-jive.c

mtouchusb.raw\_coordinates=  
 ↳coordinates [HW] Make the MicroTouch USB driver use raw\_ ( 'y', default) or cooked coordinates ( 'n' )

mtrr\_chunk\_size=nn[KMG] [X86]  
 ↳continuous chunk used for mtrr cleanup. It is largest\_ that could hold holes aka. UC entries.

mtrr\_gran\_size=nn[KMG] [X86]  
 ↳mtrr block. Used for mtrr cleanup. It is granularity of\_ Default is 1. Large value could prevent small alignment\_ ↳from using up MTRRs.

mtrr\_spare\_reg\_nr=n [X86]  
 ↳entries number. Format: <integer>  
 ↳needs more. Range: 0,7 : spare reg number  
 Default : 1  
 Used for mtrr cleanup. It is spare mtrr\_ Set to 2 or more if your graphical card\_

n2=  
 ↳card [NET] SDL Inc. RISCom/N2 synchronous serial\_

netdev=  
 ↳<name> [NET] Network devices parameters  
 ↳mean Format: <irq>,<io>,<mem\_start>,<mem\_end>,  
 ↳driver source Note that mem\_start is often overloaded to\_ something different and driver-specific. This usage is only documented in each\_ file if at all.

nf\_conntrack.acct=  
 ↳accounting [NETFILTER] Enable connection tracking flow\_ 0 to disable accounting  
 1 to enable accounting  
 Default value is 0.

nfsaddr=  
 [NFS] Deprecated. Use ip= instead.

→rst. See Documentation/admin-guide/nfs/nfsroot.

nfsroot= [NFS] nfs root filesystem for disk-less  
→boxes. See Documentation/admin-guide/nfs/nfsroot.

→rst. See Documentation/admin-guide/nfs/nfsroot.

nfsrootdebug [NFS] enable nfsroot debugging messages.  
See Documentation/admin-guide/nfs/nfsroot.

→rst.

nfs.callback\_nr\_threads=  
[NFSv4] set the total number of threads  
→that the NFS client will assign to service NFSv4  
→callback requests.

nfs.callback\_tcpport=  
[NFS] set the TCP port on which the NFSv4  
→callback channel should listen.

nfs.cache\_getent=  
[NFS] sets the pathname to the program  
→which is used to update the NFS client cache entries.

nfs.cache\_getent\_timeout=  
[NFS] sets the timeout after which an  
→attempt to update a cache entry is deemed to have  
→failed.

nfs.idmap\_cache\_timeout=  
[NFS] set the maximum lifetime for idmapper  
→cache entries.

nfs.enable\_ino64=  
[NFS] enable 64-bit inode numbers.  
If zero, the NFS client will fake up a  
→32-bit inode number for the readdir() and stat()  
→syscalls instead of returning the full 64-bit number.  
The default is to return 64-bit inode  
→numbers.

nfs.max\_session\_cb\_slots=

[NFSv4.1] Sets the maximum number of session slots the client will assign to the callback channel. This determines the maximum number of callbacks the client will process in parallel for a particular server.

```
nfs.max_session_slots=
```

[NFSv4.1] Sets the maximum number of session slots the client will attempt to negotiate with the server. This limits the number of simultaneous RPC requests that the client can send to the NFSv4.1 server. Note that there is little point in setting this value higher than the `max_tcp_slot_table_limit`.

```
nfs.nfs4_disable_idmapping=
```

[NFSv4] When set to the default of '1', ensures that both the RPC level authentication scheme and the NFS level operations agree to use numeric uids/gids if the mount is using the 'sec=sys' security flavour. In effect it is disabling idmapping, which can make migration from legacy NFSv2/v3 systems to NFSv4 easier. Servers that do not support this mode of operation will fall back to using the idmapper. To turn off this behaviour, set the value to '0'.

```
nfs.nfs4_unique_id=
```

[NFS4] Specify an additional fixed unique identification string that NFSv4 clients can insert into their `nfs_client_id4` string. This is typically a UUID that is generated at system install time.

`nfs.send_implementation_id =`  
[NFSv4.1] Send client implementation information in exchange\_id requests. If zero, no implementation information will be sent. The default is to send the implementation information.

`nfs.recover_lost_locks =`  
[NFSv4] Attempt to recover locks that were to a lease timeout on the server. Please note that doing this risks data corruption, since there are no guarantees that the file will remain unchanged after the locks are lost. If you want to enable the kernel legacy behaviour of attempting to recover these locks, then set this parameter to '1'. The default parameter value of '0' causes the kernel not to attempt recovery of lost locks.

`nfs4.layoutstats_timer =`  
[NFSv4.2] Change the rate at which the kernel sends layoutstats to the pNFS metadata server. Setting this to value to 0 causes the kernel to use whatever value is the default set by the layout driver. A non-zero value sets the minimum interval in seconds between layoutstats transmissions.

`nfsd.nfs4_disable_idmapping=`  
[NFSv4] When set to the default of '1', the NFSv4 server will return only numeric uids and gids to clients using auth\_sys, and will accept numeric uids

and gids from such clients. This is intended to ease migration from NFSv2/v3.

`nmi_debug=` [KNL,SH] Specify one or more actions to take when a NMI is triggered.  
Format: [state][,regs][,debounce][,die]

`nmi_watchdog=` [KNL,BUGS=X86] Debugging features for SMP  
Format: [panic,][nopanic,][num]  
Valid num: 0 or 1  
0 - turn hardlockup detector in  
1 - turn hardlockup detector in  
When panic is specified, panic when an NMI timeout occurs (or 'nopanic' to not panic, if CONFIG\_BOOTPARAM\_HARDLOCKUP\_WATCHDOG is set)  
To disable both hard and soft lockup detectors, please see 'nowatchdog'.  
This is useful when you use a panic=... need the box quickly up again.  
These settings can be accessed at runtime via the `nmi_watchdog` and `hardlockup_panic` sysctls.

`netpoll.carrier_timeout=` [NET] Specifies amount of time (in seconds) that netpoll should wait for a carrier. By default netpoll waits 4 seconds.

`no387` [BUGS=X86-32] Tells the kernel to use the 387 maths emulation library even if a 387 maths coprocessor is present.

`no5lvl` [X86-64] Disable 5-level paging mode. Forces kernel to use 4-level paging instead.

`no_console_suspend`

<code>↪suspend and</code>	[HW] Never suspend the console
<code>↪debugging</code>	Disable suspending of consoles during
<code>↪the rest</code>	hibernate operations. Once disabled,
<code>↪while</code>	messages can reach various consoles while
<code>↪This may</code>	of the system is being put to sleep (ie,
<code>↪known</code>	debugging driver suspend/resume hooks).
<code>↪also add</code>	not work reliably with all consoles, but is
<code>↪to control</code>	to work with serial and VGA consoles.
<code>↪suspend) to</code>	To facilitate more flexible debugging, we
	<code>console_suspend</code> , a <code>printk</code> module parameter
	it. Users could use <code>console_suspend</code> (usually
	<code>/sys/module/printk/parameters/console_</code>
	<code>turn on/off it dynamically.</code>
<code>novmcoredd</code>	[KNL, KDUMP]
<code>↪drivers to</code>	Disable device dump. Device dump allows
<code>↪collect driver</code>	append dump data to <code>vmcore</code> so you can
<code>↪the data</code>	specified debug info. Drivers can append
<code>↪in memory,</code>	without any limit and this data is stored
<code>↪ Disabling</code>	so this may cause significant memory stress.
<code>↪driver debug</code>	device dump can help save memory but the
<code>↪parameter</code>	data will be no longer available. This
<code>↪DEVICE_DUMP</code>	is only available when <code>CONFIG_PROC_VMCORE_</code>
	is set.
<code>noaliencache</code>	[MM, NUMA, SLAB] Disables the allocation of
<code>↪alien</code>	caches in the slab allocator. Saves
<code>↪per-node memory,</code>	but will impact performance.
<code>noalign</code>	[KNL, ARM]
<code>noaltinstr</code>	[S390] Disables alternative instructions



	read implies executable mappings
nofpu	[MIPS,SH] Disable hardware FPU at boot time.
nofxsr	[BUGS=X86-32] Disables x86 floating point
→extended	register save and restore. The kernel will
→only save	legacy floating-point registers on task
→switch.	
nohugeiomap	[KNL,x86,PPC] Disable kernel huge I/O
→mappings.	
nosmt	[KNL,S390] Disable symmetric multithreading
→(SMT).	Equivalent to smt=1.
	[KNL,x86] Disable symmetric multithreading
→(SMT).	nosmt=force: Force disable SMT, cannot be
→undone	via the sysfs control file.
nospectre_v1	[X86,PPC] Disable mitigations for Spectre
→Variant 1	(bounds check bypass). With this option
→data leaks are	possible in the system.
nospectre_v2	[X86,PPC_FSL_B00K3E,ARM64] Disable all
→mitigations for	the Spectre variant 2 (indirect branch
→prediction)	vulnerability. System may allow data leaks
→with this	option.
nospec_store_bypass_disable	[HW] Disable all mitigations for the
→Speculative Store Bypass vulnerability	
noxsave	[BUGS=X86] Disables x86 extended register
→state save	and restore using xsave. The kernel will
→fallback to	enabling legacy floating-point and sse
→state.	
noxsaveopt	[X86] Disables xsaveopt used in saving x86
→extended	

<ul style="list-style-type: none"> <li>→to use</li> <li>→parameter,</li> <li>→degraded because</li> <li>→while</li> <li>→systems.</li> </ul>	<p>register states. The kernel will fall back, xsave to save the states. By using this performance of saving the states is xsave doesn't support modified optimization xsaveopt supports it on xsaveopt enabled.</p>
<ul style="list-style-type: none"> <li>noxsaves</li> <li>→saving and</li> <li>→compacted</li> <li>→back to use</li> <li>→states</li> <li>→this</li> <li>→occupy more</li> </ul>	<p>[X86] Disables xsaves and xrstors used in restoring x86 extended register state in form of xsave area. The kernel will fall xsaveopt and xrstor to save and restore the in standard form of xsave area. By using parameter, xsave area per process might memory on xsaves enabled systems.</p>
<ul style="list-style-type: none"> <li>nohlt</li> <li>→sleep(SH) or</li> <li>→and not to</li> <li>→debugger.</li> </ul>	<p>[BUGS=ARM,SH] Tells the kernel that the wfi(ARM) instruction doesn't work correctly use it. This is also useful when using JTAG.</p>
<ul style="list-style-type: none"> <li>no_file_caps</li> <li>→capabilities. The</li> <li>→with privilege</li> </ul>	<p>Tells the kernel not to honor file only way then for a file to be executed is to be setuid root or executed by root.</p>
<ul style="list-style-type: none"> <li>nohalt</li> <li>→power saving</li> <li>→increases</li> <li>→reduces</li> <li>→improve performance</li> <li>→servers or</li> </ul>	<p>[IA-64] Tells the kernel not to use the function PAL_HALT_LIGHT when idle. This power-consumption. On the positive side, it interrupt wake-up latency, which may in certain environments such as networked real-time systems.</p>
<ul style="list-style-type: none"> <li>nohibernate</li> </ul>	<p>[HIBERNATION] Disable hibernation and</p>

↪ resume.

nohz= [KNL] Boottime enable/disable dynamic ticks  
Valid arguments: on, off  
Default: on

nohz\_full= [KNL,BOOT,SMP,ISOL]  
The argument is a cpu list, as described

↪ above.

In kernels built with CONFIG\_NO\_HZ\_FULL=y,

↪ set

the specified list of CPUs whose tick will

↪ be stopped

whenever possible. The boot CPU will be

↪ forced outside

the range to maintain the timekeeping. Any

↪ CPUs

in this list will have their RCU callbacks

↪ offloaded,

just as if they had also been called out in

↪ the

rcu\_nocbs= boot parameter.

noiotrap [SH] Disables trapped I/O port accesses.

noirqdebug [X86-32] Disables the code which attempts  
↪ to detect and disable unhandled interrupt sources.

no\_timer\_check [X86,APIC] Disables the code which tests for  
↪ broken timer IRQ sources.

noisapnp [ISAPNP] Disables ISA PnP code.

noinitrd [RAM] Tells the kernel not to load any  
↪ configured initial RAM disk.

nointremap [X86-64, Intel-IOMMU] Do not enable  
↪ interrupt remapping.  
[Deprecated - use intremap=off]

nointroute [IA-64]

noinvpcid [X86] Disable the INVPCID cpu feature.

nojitter [IA-64] Disables jitter checking for ITC  
↪ timers.

no-kvmclock [X86,KVM] Disable paravirtualized KVM clock

↪driver	
↪no-kvmapf	[X86,KVM] Disable paravirtualized
↪asynchronous page	fault handling.
↪no-vmw-sched-clock	[X86,PV_OPS] Disable paravirtualized VMware
↪scheduler	clock and use the default one.
↪no-steal-acc	[X86,PV_OPS,ARM64] Disable paravirtualized
↪steal time	accounting. steal time is computed, but
↪won't	influence scheduler behaviour
↪nolapic	[X86-32,APIC] Do not enable or use the
↪local APIC.	
↪nolapic_timer	[X86-32,APIC] Do not use the local APIC
↪timer.	
↪noltlbs	[PPC] Do not use large page/tlb entries for
↪kernel	lowmem mapping on PPC40x and PPC8xx
↪nomca	[IA-64] Disable machine check abort handling
↪nomce	[X86-32] Disable Machine Check Exception
↪nomfgpt	[X86-32] Disable Multi-Function General
↪Purpose	Timer usage (for AMD Geode machines).
↪nonmi_ipi	[X86] Disable using NMI IPIs during panic/
↪reboot to	shutdown the other cpus. Instead use the
↪REBOOT_VECTOR	irq.
↪nomodule	Disable module load
↪nopat	[X86] Disable PAT (page attribute table
↪extension of	pagetables) support.
↪nopcid	[X86-64] Disable the PCID cpu feature.
↪norandmaps	Don't use address space randomization.
↪Equivalent to	

	<code>echo 0 &gt; /proc/sys/kernel/randomize_va_space</code>
<code>noreplace-smp</code>	[X86-32,SMP] Don't replace SMP instructions with UP alternatives
<code>nordrand</code>	[X86] Disable kernel use of the RDRAND and RDSEED instructions even if they are
<code>↳supported</code>	by the processor. RDRAND and RDSEED are
<code>↳still</code>	available to user space applications.
<code>noresume</code>	[SWSUSP] Disables resume and restores
<code>↳original swap</code>	space.
<code>no-scroll</code>	[VGA] Disables scrollbar.
<code>↳ib80-piezo Braille</code>	This is required for the Braillex reader made by F.H. Papenmeier (Germany).
<code>nosbagart</code>	[IA-64]
<code>nosep</code>	[BUGS=X86-32] Disables x86 SYSENTER/SYSEXIT
<code>↳support.</code>	
<code>nosmp</code>	[SMP] Tells an SMP kernel to act as a UP
<code>↳kernel,</code>	and disable the IO APIC. legacy for
<code>↳"maxcpus=0".</code>	
<code>nosoftlockup</code>	[KNL] Disable the soft-lockup detector.
<code>nosync</code>	[HW,M68K] Disables sync negotiation for all
<code>↳devices.</code>	
<code>nowatchdog</code>	[KNL] Disable both lockup detectors, i.e. soft-lockup and NMI watchdog (hard-lockup).
<code>nowb</code>	[ARM]
<code>nox2apic</code>	[X86-64,APIC] Do not enable x2APIC mode.
<code>cpu0_hotplug</code>	[X86] Turn on CPU0 hotplug feature when CONFIG_BOOTPARAM_HOTPLUG_CPU0 is off. Some features depend on CPU0. Known
<code>↳dependencies are:</code>	1. Resume from suspend/hibernate depends on
<code>↳CPU0.</code>	Suspend/hibernate will fail if CPU0 is
<code>↳offline and you</code>	

need to online CPU0 before suspend/

↳hibernate.  
↳can't be  
↳CPU0 on some  
↳issues so far  
↳machines.  
↳you can

2. PIC interrupts also depend on CPU0. CPU0 removed if a PIC interrupt is detected. It's said poweroff/reboot may depend on machines although I haven't seen such after CPU0 is offline on a few tested. If the dependencies are under your control, turn on `cpu0_hotplug`.

`nps_mtm_hs_ctr= [KNL,ARC]`  
↳it.

This parameter sets the maximum duration, in cycles, each HW thread of the CTOP can run without interruptions, before HW switches.

The actual maximum duration is 16 times this parameter's value.  
Format: integer between 1 and 255  
Default: 255

`nptcg=`  
↳global TLB  
↳SUMMARY or

[IA-64] Override max number of concurrent purges which is reported from either PAL\_VM\_SAL PAL0.

`nr_cpus=`  
↳SMP kernel  
↳the kernel to  
↳than the  
↳ later in  
↳until it reaches  
↳for per-cpu  
↳physical cpu

[SMP] Maximum number of processors that an could support. `nr_cpus=n : n >= 1` limits support 'n' processors. It could be larger number of already plugged CPU during bootup, runtime you can physically add extra cpu n. So during boot up some boot time memory variables need be pre-allocated for later hot plugging.

`nr_uarts=`  
↳registered.

[SERIAL] maximum number of UARTs to be

`numa_balancing= [KNL,X86]` Enable or disable automatic NUMA

↪balancing. Allowed values are enable and disable

numa\_zonelist\_order= [KNL, BOOT] Select zonelist order for ↪  
↪NUMA. 'node', 'default' can be specified  
This can be set from sysctl after boot.  
See Documentation/admin-guide/sysctl/vm.rst ↪  
↪for details.

ohci1394\_dma=early [HW] enable debugging via the ↪  
↪ohci1394 driver. See Documentation/core-api/  
↪debugging-via-ohci1394.rst for more  
info.

olpc\_ec\_timeout= [OLPC] ms delay when issuing EC commands  
Rather than timing out after 20 ms if an EC  
command is not properly ACKed, override the ↪  
↪length of the timeout. We have interrupts ↪  
↪disabled while waiting for the ACK, so if this is set too ↪  
↪high interrupts *may* be lost!

omap\_mux= [OMAP] Override bootloader pin multiplexing.  
Format: <mux\_mode0.mode\_name=value>...  
For example, to override I2C bus2:  
omap\_mux=i2c2\_scl.i2c2\_scl=0x100,i2c2\_sda.  
↪i2c2\_sda=0x100

oprofile.timer= [HW] Use timer interrupt instead of performance ↪  
↪counters

oprofile.cpu\_type= Force an oprofile cpu type  
This might be useful if you have an older ↪  
↪oprofile userland or if you want common events.  
Format: { arch\_perfmon }  
arch\_perfmon: [X86] Force use of ↪  
↪architectural perfmon on Intel CPUs instead of the  
CPU specific event set.  
timer: [X86] Force use of architectural NMI  
timer mode (see also oprofile.timer  
for generic hr timer mode)

oops=panic Always panic on oopses. Default is to just ↪  
↪kill the

	process, but there is a small probability of deadlocking the machine. This will also cause panics on machine.
↪check exceptions.	
↪reboot.	Useful together with panic=30 to trigger a
page_alloc.shuffle=	[KNL] Boolean flag to control whether the
↪page allocator	should randomize its free lists. The
↪randomization may	be automatically enabled if the kernel
↪detects it is	running on a platform with a direct-mapped
↪memory-side	cache, and this parameter can be used to override/disable that behavior. The state
↪of the flag	can be read from sysfs at: /sys/module/page_alloc/parameters/shuffle.
page_owner=	[KNL] Boot-time page_owner enabling option. Storage of the information about who
↪allocated	each page is disabled in default. With this
↪switch,	we can turn it on. on: enable the feature
page_poison=	[KNL] Boot-time parameter changing the
↪state of	poisoning on the buddy allocator, available
↪with	CONFIG_PAGE_POISONING=y. off: turn off poisoning (default) on: turn on poisoning
panic=	[KNL] Kernel behaviour on panic: delay
↪<timeout>	timeout > 0: seconds before rebooting timeout = 0: wait forever timeout < 0: reboot immediately Format: <timeout>
panic_print=	Bitmask for printing system info when panic
↪happens.	User can chose combination of the following
↪bits:	bit 0: print all tasks info bit 1: print system memory info

bit 2: print timer info  
bit 3: print locks info if CONFIG\_LOCKDEP

→ is on

bit 4: print ftrace buffer  
bit 5: print all printk messages in buffer

panic\_on\_taint= Bitmask for conditionally calling panic()  
→ in add\_taint()

Format: <hex>[,nouser\_taint]  
Hexadecimal bitmask representing the set of

→ TAIN T flags

that will cause the kernel to panic when

→ add\_taint() is

called with any of the flags in this set.  
The optional switch "nouser\_taint" can be

→ utilized to

prevent userspace forced crashes by writing

→ to sysctl

/proc/sys/kernel/tainted any flagset

→ matching with the

bitmask set on panic\_on\_taint.  
See Documentation/admin-guide/

→ tainted-kernels.rst for

extra details on the taint flags that users

→ can pick

to compose the bitmask to assign to panic\_

→ on\_taint.

panic\_on\_warn panic() instead of WARN(). Useful to cause  
→ kdump on a WARN().

crash\_kexec\_post\_notifiers  
Run kdump after running panic-notifiers and

→ dumping

kmsg. This only for the users who doubt

→ kdump always

succeeds in any situation.  
Note that this also increases risks of

→ kdump failure,

because some panic notifiers can make the

→ crashed

kernel more unstable.

parkbd.port= [HW] Parallel port number the keyboard  
→ adapter is

connected to, default is 0.  
Format: <parport#>

parkbd.mode= [HW] Parallel port keyboard adapter mode of  
→ operation,

0 for XT, 1 for AT (default is AT).

Format: <mode>

parport= [HW,PPT] Specify parallel ports. 0 disables.  
 Format: { 0 | auto | 0xBBB[,IRQ[,DMA]] }  
 Use 'auto' to force the driver to use any  
 IRQ/DMA settings detected (the default is to  
 ignore detected IRQ/DMA settings because of  
 possible conflicts). You can specify the

→base address, IRQ, and DMA settings; IRQ and DMA  
 →detected should be numbers, or 'auto' (for using  
 →'nofifo' settings on that particular port), or  
 →detected). (to avoid using a FIFO even if it is  
 →they Parallel ports are assigned in the order  
 are specified on the command line, starting  
 with parport0.

parport\_init\_mode= [HW,PPT]  
 Configure VIA parallel port to operate in  
 a specific mode. This is necessary on

→Pegasos computer where firmware has no options for  
 →setting up parallel port mode and sets it to spp.  
 →chips. Currently this function knows 686a and 8231.

Format: [spp|ps2|epp|ecp|ecpepp]

pause\_on\_oops= Halt all CPUs after the first oops has been  
 →printed for the specified number of seconds. This is  
 →to be used if your oopses keep scrolling off the screen.

pcbit= [HW,ISDN]

pcd. [PARIDE]  
 See header of drivers/block/paride/pcd.c.  
 See also Documentation/admin-guide/blockdev/  
 →paride.rst.

pci=option[,option...] [PCI] various PCI subsystem options.  
 →specific device Some options herein operate on a  
 or a set of devices (<pci\_dev>).

→ These are specified in one of the following

→ formats:

→ `[<domain>:]<bus>:<dev>.<func>[/<dev>`

→ `.<func>]*`

→ `<subdevice>]`

→ PCI

→ may change

→ motherboard

→ caused

→ function

→ the base

→ format

→ multiple

→ kernel `earlydump` dump PCI config space before the

→ don't access `off bios` changes anything [X86] don't probe for the PCI bus [X86-32] force use of PCI BIOS,

→ your machine `nobios` the hardware directly. Use this if

→ only direct hardware access methods are allowed.

→ Use this if you experience crashes upon

→ bootup and you suspect they are caused by the BIOS. [X86] Force use of PCI

→ Configuration Access `conf1` Mechanism 1 (config address in IO



→the boot IRQ equivalent of an IRQ that  
→connects to a chipset where boot IRQs cannot be  
→disabled. The opposite of ioapicreroute.  
biosirq [X86-32] Use PCI BIOS calls to get  
→the interrupt routing table. These calls are  
→known to be buggy on several machines and they hang  
→the machine when used, but on other computers  
→it's the only way to get the interrupt routing  
→table. Try this option if the kernel is unable  
→to allocate IRQs or discover secondary PCI  
→buses on your motherboard.  
rom [X86] Assign address space to  
→expansion ROMs. Use with caution as certain devices  
→share address decoders between ROMs and  
→other resources.  
norom [X86] Do not assign address space to  
→have expansion ROMs that do not already  
nobar [X86] Do not assign address space  
→to the BARs that weren't assigned by the  
→BIOS. [X86] Set a bit mask of IRQs  
irqmask=0xMMMM →allowed to be assigned automatically to PCI  
→devices. You can make the kernel exclude IRQs of  
→your ISA cards this way.  
pirqaddr=0xAAAAA [X86] Specify the physical  
→address of the PIRQ table (normally  
→generated by the BIOS) if it is outside the  
F0000h-100000h range.  
lastbus=N [X86] Scan all buses thru bus #N.  
→Can be

useful if the kernel is unable to  
 find your secondary buses and you want to  
 tell it explicitly which ones they are.  
 assign-busses [X86] Always assign all PCI bus  
 usepirqmask numbers ourselves, overriding  
 whatever the firmware may have done.  
 stored [X86] Honor the possible IRQ mask  
 needed on in the BIOS \$PIR table. This is  
 notably some systems with broken BIOSes,  
 XE3 some HP Pavilion N5400 and Omnibook  
 if ACPI notebooks. This will have no effect  
 noacpi IRQ routing is enabled.  
 routing [X86] Do not use ACPI for IRQ  
 use\_crs or for PCI scanning.  
 information [X86] Use PCI host bridge window  
 later, this from ACPI. On BIOSes from 2008 or  
 to use this, is enabled by default. If you need  
 please report a bug.  
 nocrs [X86] Ignore PCI host bridge  
 windows from ACPI. If you need to use this, please  
 report a bug.  
 routeirq Do IRQ routing for all PCI devices.  
 device(), This is normally done in pci\_enable\_  
 workaround so this option is a temporary  
 it. for broken drivers that don't call  
 skip\_isa\_align [X86] do not align io start addr,  
 so can handle more pci cards  
 noearly [X86] Don't do any early type 1  
 scanning. This might help on some broken  
 boards which machine check when some devices'  
 config space is read. But various workarounds  
 are disabled

and some IOMMU drivers will not work.

`bfsort` Sort PCI devices into breadth-first order. This sorting is done to get a device order compatible with older (<= 2.4) kernels.

`nobfsort` Don't sort PCI devices into breadth-first order.

`pcie_bus_tune_off` Disable PCIe MPS (Max Payload Size) tuning and use the BIOS-configured MPS defaults.

`pcie_bus_safe` Set every device's MPS to the largest value supported by all devices below the root complex.

`pcie_bus_perf` Set device MPS to the largest allowable MPS based on its parent bus. Also set MRRS (Max Read Request Size) to the largest supported value (no larger than the MPS that the device or bus can support) for best performance.

`pcie_bus_peer2peer` Set every device's MPS to 128B, which every device is guaranteed to support. This configuration allows peer-to-peer DMA between any pair of devices, possibly at the cost of reduced performance. This also guarantees that hot-added devices will work.

`cbiosize=nn[KMG]` The fixed amount of bus space which is reserved for the CardBus bridge's IO window. The default value is 256 bytes.

`cbmemsize=nn[KMG]` The fixed amount of bus space which is reserved for the CardBus bridge's memory window. The default value is 64 megabytes.

`resource_alignment=`  
Format:

[<order of align>@]<pci\_dev>[; ...]  
 Specifies alignment and device to

→reassign aligned memory resources. How to specify the device is described

→above. If <order of align> is not

→specified, PAGE\_SIZE is used as alignment. A PCI-PCI bridge can be specified

→if resource windows need to be expanded. To specify the alignment for several instances of a device, the PCI

→vendor, device, subvendor, and subdevice

→may be specified, e.g.,

→12@pci:8086:9c22:103c:198f for 4096-byte alignment.

→ecrc= Enable/disable PCIe ECRC

→(transaction layer end-to-end CRC checking). bios: Use BIOS/firmware settings.

→This is the the default.

→hpbiosize=nn[KMG] The fixed amount of bus off: Turn ECRC off

→space which is reserved for hotplug bridge's IO on: Turn ECRC on.

→window. Default size is 256 bytes.

→hpmmiosize=nn[KMG] The fixed amount of bus

→space which is reserved for hotplug bridge's MMIO

→window. Default size is 2 megabytes.

→hpmmioprefsize=nn[KMG] The fixed amount of bus

→space which is reserved for hotplug bridge's MMIO\_

→PREF window. Default size is 2 megabytes.

→hpmemsize=nn[KMG] The fixed amount of bus

→space which is reserved for hotplug bridge's MMIO

→and MMIO\_PREF window.

→bus numbers hpbusize=nn Default size is 2 megabytes. The minimum amount of additional

reserved for buses below a hotplug.

→bridge.

Default is 1.

→bridge resources realloc= Enable/disable reallocating PCI

→small to if allocations done by BIOS are too

→all child accommodate resources required by

devices.

off: Turn realloc off

on: Turn realloc on

same as realloc=on

noari do not use PCIe ARI.

noats [PCIE, Intel-IOMMU, AMD-IOMMU]

do not use PCIe ATS (and IOMMU)

→device IOTLB).

→Otherwise we pcie\_scan\_all Scan all possible PCIe devices.

→PCIe downstream only look for one device below a

port.

→window to the PCIe big\_root\_window Try to add a big 64bit memory

→hardware root complex on AMD CPUs. Some GFX

→all VRAM. can resize a BAR to allow access to

→(it may Adding the window is slightly risky

→so this conflict with unreported devices),

taints the kernel.

→the format disable\_acs\_redir=<pci\_dev>[; ...]

→semicolons. Specify one or more PCI devices (in

→PCI ACS Each device specified will have the

→which will redirect capabilities forced off

→through allow P2P traffic between devices

bridges without forcing it upstream.

→ Note: this removes isolation between

→devices and may put more devices in an IOMMU

→group. force\_floating [S390] Force usage of floating

↳interrupts.  
                   nomio                  [S390] Do not use MIO instructions.  
                   norid                  [S390] ignore the RID field and  
 ↳force use of  
   one PCI domain per PCI function

                  pcie\_aspm=          [PCIE] Forcibly enable or disable PCIe  
 ↳Active State Power  
                   off                  Management.  
                   force              Disable ASPM.  
                                       Enable ASPM even on devices that claim not  
 ↳to support it.  
                                       WARNING: Forcing ASPM on may cause system  
 ↳lockups.

                  pcie\_ports=          [PCIE] PCIe port services handling:  
                   native              Use native PCIe services (PME, AER, DPC,  
 ↳PCIe hotplug)  
                                       even if the platform doesn't give the OS  
 ↳permission to  
                                       use them. This may cause conflicts if the  
 ↳platform  
                                       also tries to use these services.  
                   dpc-native          Use native PCIe service for DPC  
 ↳only. May  
                                       cause conflicts if firmware uses  
 ↳AER or DPC.  
                   compat              Disable native PCIe services (PME, AER, DPC,  
 ↳ PCIe  
                                       hotplug).

                  pcie\_port\_pm=      [PCIE] PCIe port power management handling:  
                   off                  Disable power management of all PCIe ports  
                   force              Forcibly enable power management of all  
 ↳PCIe ports

                  pcie\_pme=          [PCIE,PM] Native PCIe PME signaling options:  
                   nomsi              Do not use MSI for native PCIe PME  
 ↳signaling (this makes  
                                       all PCIe root ports use INTx for all  
 ↳services).

                  pcmv=              [HW,PCMCIA] BadgePAD 4

                  pd\_ignore\_unused  
                                       [PM]  
 ↳bootloader on,  
                                       Keep all power-domains already enabled by  
 ↳useful  
                                       even if no driver has claimed them. This is  
                                       for debug and development, but should not be

needed on a platform with proper driver.  
→support.

pd. [PARIDE]  
See Documentation/admin-guide/blockdev/  
→paride.rst.

pdchassis= [PARISC,HW] Disable/Enable PDC Chassis.  
→Status codes at boot time.  
Format: { 0 | 1 }  
See arch/parisc/kernel/pdc\_chassis.c

percpu\_alloc= Select which percpu first chunk allocator.  
→to use. Currently supported values are "embed" and  
→"page". Archs may support subset or none of the  
→selections. See comments in mm/percpu.c for details on  
→each allocator. This parameter is primarily for  
→debugging and performance comparison.

pf. [PARIDE]  
See Documentation/admin-guide/blockdev/  
→paride.rst.

pg. [PARIDE]  
See Documentation/admin-guide/blockdev/  
→paride.rst.

pirq= [SMP,APIC] Manual mp-table setup  
See Documentation/x86/i386/IO-APIC.rst.

plip= [PPT,NET] Parallel port network link  
Format: { parport<nr> | timid | 0 }  
See also Documentation/admin-guide/parport.  
→rst.

pmtmr= [X86] Manual setup of pmtmr I/O Port.  
Override pmtimer IOPort with a hex value.  
e.g. pmtmr=0x508

pm\_debug\_messages [SUSPEND,KNL]  
Enable suspend/resume debug messages during  
→boot up.

pnp.debug=1 [PNP]  
Enable PNP debug messages (depends on the

CONFIG\_PNP\_DEBUG\_MESSAGES option). Change  
 ↪ at run-time via /sys/module/pnp/parameters/debug. We  
 ↪ always show current resource usage; turning this on  
 ↪ also shows possible settings and some assignment  
 ↪ information.

```

    pnpacpi=      [ACPI]
                 { off }

    pnpbios=     [ISAPNP]
                 { on | off | curr | res | no-curr | no-res }

    pnp_reserve_irq=
    ↪ autoconfiguration [ISAPNP] Exclude IRQs for the

    pnp_reserve_dma=
    ↪ autoconfiguration [ISAPNP] Exclude DMAs for the

    pnp_reserve_io= [ISAPNP] Exclude I/O ports for the
    ↪ autoconfiguration
    ↪ size). Ranges are in pairs (I/O port base and

    pnp_reserve_mem=
                 [ISAPNP] Exclude memory regions for the
                 autoconfiguration.
                 Ranges are in pairs (memory base and size).

    ports=       [IP_VS_FTP] IPVS ftp helper module
                 Default is 21.
                 Up to 8 (IP_VS_APP_MAX_PORTS) ports
                 may be specified.
                 Format: <port>,<port>....

    powersave=off [PPC] This option disables power saving
    ↪ features.
    ↪ the
    ↪ save
    ↪ reduces
                 [PPC] This option disables cpuidle and sets
                 platform machine description specific power_
                 function to NULL. On Idle the CPU just
                 execution priority.

    ppc_strict_facility_enable
                 [PPC] This option catches any kernel
  
```

↳floating point, Altivec, VSX and SPE outside of regions.  
↳specifically allowed (eg kernel\_enable\_fpu()/kernel\_  
↳disable\_fpu()). There is some performance impact when.  
↳enabling this.

ppc\_tm= [PPC]  
Format: {"off"}  
Disable Hardware Transactional Memory

print-fatal-signals=  
[KNL] debug: print fatal signals

↳handling If enabled, warn about various signal.  
↳signals, related application anomalies: too many.  
↳causing a too many POSIX.1 timers, fatal signals.  
coredump - etc.

↳overflow, If you hit the warning due to signal,  
you might want to try "ulimit -i unlimited".  
default: off.

printk.always\_kmsg\_dump=  
Trigger kmsg\_dump for cases other than.  
↳kernel oops or panics  
Format: <bool> (1/Y/y=enable, 0/N/  
↳n=disable) default: disabled

printk.devkmsg={on,off,ratelimit}  
Control writing to /dev/kmsg.  
on - unlimited logging to /dev/kmsg from.  
↳userspace off - logging to /dev/kmsg disabled  
ratelimit - ratelimit the logging  
Default: ratelimit

printk.time= Show timing data prefixed to each printk.  
↳message line Format: <bool> (1/Y/y=enable, 0/N/  
↳n=disable)

processor.max\_cstate= [HW,ACPI]

Limit processor to maximum C-state  
`max_cstate=9` overrides any DMI blacklist.

`limit.`

`processor.nocst` [HW,ACPI]  
Ignore the `_CST` method to determine  
C-states,  
instead using the legacy FADT method

`profile=` [KNL] Enable kernel profiling via `/proc/`  
`profile`  
Format: [`<profiletype>`,]`<number>`  
Param: `<profiletype>`: "schedule", "sleep",  
or "kvm"  
[defaults to kernel profiling]  
Param: "schedule" - profile schedule points.  
Param: "sleep" - profile D-state sleeping.  
(millisecs).  
Requires `CONFIG_SCHEDSTATS`  
Param: "kvm" - profile VM exits.  
Param: `<number>` - step/bucket size as a  
power of 2 for  
statistical time based profiling.

`prompt_ramdisk=` [RAM] List of RAM disks to prompt for  
floppy disk  
before loading.  
See Documentation/admin-guide/blockdev/  
`ramdisk.rst.`

`prot_virt=` [S390] enable hosting protected virtual  
machines  
isolated from the hypervisor (if hardware  
supports  
that).  
Format: `<bool>`

`psi=` [KNL] Enable or disable pressure stall  
information  
tracking.  
Format: `<bool>`

`psmouse.proto=` [HW,MOUSE] Highest PS2 mouse protocol  
extension to  
probe for; one of (bare|imps|exps|lifebook|any).

`psmouse.rate=` [HW,MOUSE] Set desired mouse report rate,  
in reports  
per second.

`psmouse.resetafter=` [HW,MOUSE]  
Try to reset the device after so many bad

→packets (0 = never).  
psmouse.resolution=  
[HW,MOUSE] Set desired mouse resolution, in  
→dpi.  
psmouse.smartscroll=  
[HW,MOUSE] Controls Logitech smartscroll  
→autorepeat.  
0 = disabled, 1 = enabled (default).  
pstore.backend= Specify the name of the pstore backend to  
→use  
pt. [PARIDE]  
See Documentation/admin-guide/blockdev/  
→paride.rst.  
pti= [X86\_64] Control Page Table Isolation of  
→user and kernel address spaces. Disabling this  
→feature removes hardening, but improves performance  
→of system calls and interrupts.  
on - unconditionally enable  
off - unconditionally disable  
auto - kernel detects whether your CPU  
→model is vulnerable to issues that PTI  
→mitigates  
→pti=auto. Not specifying this option is equivalent to  
nopti [X86\_64]  
Equivalent to pti=off  
pty.legacy\_count=  
[KNL] Number of legacy pty's. Overwrites  
→compiled-in default number.  
quiet [KNL] Disable most log messages  
r128= [HW,DRM]  
raid= [HW,RAID]  
See Documentation/admin-guide/md.rst.  
ramdisk\_size= [RAM] Sizes of RAM disks in kilobytes

See Documentation/admin-guide/blockdev/  
 ↪ ramdisk.rst.

random.trust\_cpu={on,off}  
 [KNL] Enable or disable trusting the use of  
 ↪ the  
 ↪ available) to  
 ↪ controlled  
 CPU's random number generator (if  
 fully seed the kernel's CRNG. Default is  
 by CONFIG\_RANDOM\_TRUST\_CPU.

ras=option[,option,...] [KNL] RAS-specific options

cec\_disable [X86]  
 ↪ Collector,  
 Disable the Correctable Errors  
 see CONFIG\_RAS\_CEC help text.

rcu\_nocbs= [KNL]  
 ↪ above,  
 The argument is a cpu list, as described  
 except that the string "all" can be used to  
 specify every CPU on the system.

↪ set  
 ↪ no-callback CPUs.  
 ↪ will be  
 ↪ that  
 ↪ and  
 ↪ number.  
 ↪ CPUs,  
 ↪ efficiency

In kernels built with CONFIG\_RCU\_NOCB\_CPU=y,  
 the specified list of CPUs to be  
 Invocation of these CPUs' RCU callbacks  
 offloaded to "rcuox/N" kthreads created for  
 purpose, where "x" is "p" for RCU-preempt,  
 "s" for RCU-sched, and "N" is the CPU  
 This reduces OS jitter on the offloaded  
 which can be useful for HPC and real-time  
 workloads. It can also improve energy  
 for asymmetric multiprocessors.

rcu\_nocb\_poll [KNL]  
 Rather than requiring that offloaded CPUs  
 (specified by rcu\_nocbs= above) explicitly  
 awaken the corresponding "rcuoN" kthreads,  
 make these kthreads poll for callbacks.  
 This improves the real-time response for the  
 offloaded CPUs by relieving them of the

→need to wake up the corresponding kthread, but

→degrades energy efficiency by requiring that the

→kthreads periodically wake up to do the polling.

rcutree.blimit= [KNL]  
Set maximum number of finished RCU

→callbacks to process in one batch.

rcutree.dump\_tree= [KNL]  
Dump the structure of the rcu\_node

→combining tree out at early boot. This is used for

→diagnostic purposes, to verify correct tree setup.

rcutree.gp\_cleanup\_delay= [KNL]  
Set the number of jiffies to delay each

→step of RCU grace-period cleanup.

rcutree.gp\_init\_delay= [KNL]  
Set the number of jiffies to delay each

→step of RCU grace-period initialization.

rcutree.gp\_preinit\_delay= [KNL]  
Set the number of jiffies to delay each

→step of RCU grace-period pre-initialization, that

→is, the propagation of recent CPU-hotplug

→changes up the rcu\_node combining tree.

rcutree.use\_softirq= [KNL]  
If set to zero, move all RCU\_SOFTIRQ

→processing to per-CPU rcuc kthreads. Defaults to a

→non-zero value, meaning that RCU\_SOFTIRQ is used by

→default. Specify rcutree.use\_softirq=0 to use rcuc

→kthreads.

rcutree.rcu\_fanout\_exact= [KNL]  
Disable autobalancing of the rcu\_node

→combining

tree. This is used by rcutorture, and might possibly be useful for architectures having

→high cache-to-cache transfer latencies.

rcutree.rcu\_fanout\_leaf= [KNL]  
Change the number of CPUs assigned to each leaf rcu\_node structure. Useful for very large systems, which will choose the value

→64, and for NUMA systems with large

→remote-access latencies, which will choose a value aligned with the appropriate hardware boundaries.

rcutree.jiffies\_till\_first\_fqs= [KNL]  
Set delay from grace-period initialization

→to first attempt to force quiescent states. Units are jiffies, minimum value is zero, and maximum value is HZ.

rcutree.jiffies\_till\_next\_fqs= [KNL]  
Set delay between subsequent attempts to

→force quiescent states. Units are jiffies,

→minimum value is one, and maximum value is HZ.

rcutree.jiffies\_till\_sched\_qs= [KNL]  
Set required age in jiffies for a given grace period before RCU starts soliciting quiescent-state help from rcu\_note\_context\_switch() and cond\_

→resched().

If not specified, the kernel will calculate a value based on the most recent settings of rcutree.jiffies\_till\_first\_fqs and rcutree.jiffies\_till\_next\_fqs. This calculated value may be viewed in rcutree.jiffies\_to\_sched\_qs. Any attempt

→to set rcutree.jiffies\_to\_sched\_qs will be

→cheerfully overwritten.

rcutree.kthread\_prio= [KNL,BOOT]  
Set the SCHED\_FIFO priority of the RCU

→per-CPU kthreads (rcuc/N). This value is also used

→for

rcutree.rcu\_nocb\_gp\_stride= [KNL]  
the priority of the RCU boost threads (rcub/  
and for the RCU grace-period kthreads (rcu\_  
and for the RCU grace-period kthreads (rcu\_  
rcu\_preempt, and rcu\_sched). If RCU\_BOOST is  
set, valid values are 1-99 and the default  
(the least-favored priority). Otherwise,  
RCU\_BOOST is not set, valid values are 0-99,  
the default is zero (non-realtime  
operation).

rcutree.rcu\_nocb\_gp\_stride= [KNL]  
Set the number of NOCB callback kthreads in  
each group, which defaults to the square  
of the number of CPUs. Larger numbers  
the wakeup overhead on the global  
kthread, but increases that same overhead on  
each group's NOCB grace-period kthread.

rcutree.qhimark= [KNL]  
Set threshold of queued RCU callbacks  
beyond which batch limiting is disabled.

rcutree.qlowmark= [KNL]  
Set threshold of queued RCU callbacks below  
which batch limiting is re-enabled.

rcutree.qovld= [KNL]  
Set threshold of queued RCU callbacks  
beyond which RCU's force-quiescent-state scan will  
aggressively enlist help from cond\_resched() and sched\_  
IPIs to help CPUs more quickly reach quiescent  
states.  
Set to less than zero to make this be set  
based on rcutree.qhimark at boot time and to zero  
to disable more aggressive help enlistment.

rcutree.rcu\_idle\_gp\_delay= [KNL]

Set wakeup interval for idle CPUs that have RCU callbacks (RCU\_FAST\_NO\_HZ=y).

rcutree.rcu\_idle\_lazy\_gp\_delay= [KNL]

Set wakeup interval for idle CPUs that have only "lazy" RCU callbacks (RCU\_FAST\_NO\_

→HZ=y).

Lazy RCU callbacks are those which RCU can prove do nothing more than free memory.

rcutree.rcu\_kick\_kthreads= [KNL]

Cause the grace-period kthread to get an

→extra

wake\_up() if it sleeps three times longer

→than

it should at force-quiescent-state time. This wake\_up() will be accompanied by a WARN\_ONCE() splat and an ftrace\_dump().

rcutree.sysrq\_rcu= [KNL]

Commandeer a sysrq key to dump out Tree

→RCU's

rcu\_node tree with an eye towards

→determining

why a new grace period has not yet started.

rcuperf.gp\_async= [KNL]

Measure performance of asynchronous grace-period primitives such as call\_rcu().

rcuperf.gp\_async\_max= [KNL]

Specify the maximum number of outstanding callbacks per writer thread. When a writer thread exceeds this limit, it invokes the corresponding flavor of rcu\_barrier() to

→allow

previously posted callbacks to drain.

rcuperf.gp\_exp= [KNL]

Measure performance of expedited synchronous grace-period primitives.

rcuperf.holdoff= [KNL]

Set test-start holdoff period. The purpose

→of

this parameter is to delay the start of the test until boot completes in order to avoid interference.

rcuperf.kfree\_rcu\_test= [KNL]

Set to measure performance of kfree\_rcu()

→flooding.

rcuperf.kfree\_nthreads= [KNL]

The number of threads running loops of

→kfree\_rcu().

rcuperf.kfree\_alloc\_num= [KNL]

Number of allocations and frees done in an

→iteration.

rcuperf.kfree\_loops= [KNL]

Number of loops doing rcuperf.kfree\_alloc\_

→num number

of allocations and frees.

rcuperf.nreaders= [KNL]

Set number of RCU readers. The value -1

→selects

N, where N is the number of CPUs. A value "n" less than -1 selects N-n+1, where N is

→again

the number of CPUs. For example, -2

→selects N

(the number of CPUs), -3 selects N+1, and

→so on.

A value of "n" less than or equal to -N

→selects

a single reader.

rcuperf.nwriters= [KNL]

Set number of RCU writers. The values

→operate

the same as for rcuperf.nreaders.

N, where N is the number of CPUs

rcuperf.perf\_type= [KNL]

Specify the RCU implementation to test.

rcuperf.shutdown= [KNL]

Shut the system down after performance tests complete. This is useful for hands-off

→automated

testing.

rcuperf.verbose= [KNL]

Enable additional printk() statements.

rcuperf.writer\_holdoff= [KNL]

Write-side holdoff between grace periods, in microseconds. The default of zero says no holdoff.

`rcutorture.fqs_duration= [KNL]`  
Set duration of `force_quiescent_state` bursts in microseconds.

`rcutorture.fqs_holdoff= [KNL]`  
Set holdoff time within `force_quiescent_`  
→ `state` bursts  
in microseconds.

`rcutorture.fqs_stutter= [KNL]`  
Set wait time between `force_quiescent_state_`  
→ `bursts`  
in seconds.

`rcutorture.fwd_progress= [KNL]`  
Enable RCU grace-period forward-progress\_  
→ `testing`  
for the types of RCU supporting this notion.

`rcutorture.fwd_progress_div= [KNL]`  
Specify the fraction of a CPU-stall-warning period to do tight-loop forward-progress\_  
→ `testing.`

`rcutorture.fwd_progress_holdoff= [KNL]`  
Number of seconds to wait between successive forward-progress tests.

`rcutorture.fwd_progress_need_resched= [KNL]`  
Enclose `cond_resched()` calls within checks\_  
→ `for`  
→ `forward-progress`  
`need_resched()` during tight-loop\_  
testing.

`rcutorture.gp_cond= [KNL]`  
Use conditional/asynchronous update-side primitives, if available.

`rcutorture.gp_exp= [KNL]`  
Use expedited update-side primitives, if\_  
→ `available.`

`rcutorture.gp_normal= [KNL]`  
Use normal (non-expedited) asynchronous update-side primitives, if available.

`rcutorture.gp_sync= [KNL]`  
Use normal (non-expedited) synchronous update-side primitives, if available. If\_

→all of `rcutorture.gp_cond=`, `rcutorture.gp_exp=`,  
`rcutorture.gp_normal=`, and `rcutorture.gp_`  
→sync= are zero, `rcutorture` acts as if is  
→interpreted they are all non-zero.

`rcutorture.n_barrier_cbs= [KNL]`  
Set callbacks/threads for `rcu_barrier()`  
→testing.

`rcutorture.nfakewriters= [KNL]`  
Set number of concurrent RCU writers.   
→These just stress RCU, they don't participate in the  
→actual test, hence the "fake".

`rcutorture.nreaders= [KNL]`  
Set number of RCU readers. The value `-1`  
→selects `N-1`, where `N` is the number of CPUs. A value  
→again "n" less than `-1` selects `N-n-2`, where `N` is  
→selects N the number of CPUs. For example, `-2`  
→so on. (the number of CPUs), `-3` selects `N+1`, and

`rcutorture.object_debug= [KNL]`  
Enable debug-object double-call\_rcu()  
→testing.

`rcutorture.onoff_holdoff= [KNL]`  
Set time (s) after boot for CPU-hotplug  
→testing.

`rcutorture.onoff_interval= [KNL]`  
Set time (jiffies) between CPU-hotplug  
→operations, or zero to disable CPU-hotplug testing.

`rcutorture.shuffle_interval= [KNL]`  
Set task-shuffle interval (s). Shuffling  
→tasks allows some CPUs to go into `dyntick-idle`  
→mode during the `rcutorture` test.

`rcutorture.shutdown_secs= [KNL]`

- This Set time (s) after boot system shutdown. `rcutorture.stall_cpu= [KNL]` is useful for hands-off automated testing.
- stall Duration of CPU stall (s) to test RCU CPU warnings, zero to disable.
- result `rcutorture.stall_cpu_block= [KNL]` Sleep while stalling if set. This will in warnings from preemptible RCU in addition to any other stall-related activity.
- stall. `rcutorture.stall_cpu_holdoff= [KNL]` Time to wait (s) after boot before inducing
- `rcutorture.stall_cpu_irqsoff= [KNL]` Disable interrupts while stalling if set.
- cpu `rcutorture.stall_gp_kthread= [KNL]` Duration (s) of forced sleep within RCU grace-period kthread to test RCU CPU stall warnings, zero to disable. If both stall and stall\_gp\_kthread are specified, the kthread is starved first, then the CPU.
- `rcutorture.stat_interval= [KNL]` Time (s) between statistics printk()s.
- specifying `rcutorture.stutter= [KNL]` Time (s) to stutter testing, for example, five seconds causes the test to run for five seconds, wait for five seconds, and so on. This tests RCU's ability to transition abruptly to and from idle.
- 2=yes. `rcutorture.test_boost= [KNL]` Test RCU priority boosting? 0=no, 1=maybe, "Maybe" means test if the RCU implementation under test support RCU priority boosting.
- `rcutorture.test_boost_duration= [KNL]` Duration (s) of each individual boost test.

`rcutorture.test_boost_interval= [KNL]`  
Interval (s) between each boost test.

`rcutorture.test_no_idle_hz= [KNL]`  
Test RCU's dyntick-idle handling. See also `rcutorture.shuffle_interval` parameter.

`rcutorture.torture_type= [KNL]`  
Specify the RCU implementation to test.

`rcutorture.verbose= [KNL]`  
Enable additional `printk()` statements.

`rcupdate.rcu_cpu_stall_ftrace_dump= [KNL]`  
Dump ftrace buffer after reporting RCU CPU stall warning.

`rcupdate.rcu_cpu_stall_suppress= [KNL]`  
Suppress RCU CPU stall warning messages.

`rcupdate.rcu_cpu_stall_suppress_at_boot= [KNL]`  
Suppress RCU CPU stall warning messages and `rcutorture` writer stall warnings that occur during early boot, that is, during the time before the `init` task is spawned.

`rcupdate.rcu_cpu_stall_timeout= [KNL]`  
Set timeout for RCU CPU stall warning messages.

`rcupdate.rcu_expedited= [KNL]`  
Use expedited grace-period primitives, for example, `synchronize_rcu_expedited()` instead of `synchronize_rcu()`. This reduces latency, but can increase CPU utilization, degrade real-time latency, and degrade energy efficiency.

No effect on `CONFIG_TINY_RCU` kernels.

`rcupdate.rcu_normal= [KNL]`  
Use only normal grace-period primitives, for example, `synchronize_rcu()` instead of `synchronize_rcu_expedited()`. This improves real-time latency, CPU utilization, and energy efficiency, but can expose users to increased grace-period latency. This parameter overrides `rcupdate.rcu_expedited`. No effect on

CONFIG\_TINY\_RCU kernels.

rcupdate.rcu\_normal\_after\_boot= [KNL]  
Once boot has completed (that is, after rcu\_end\_inkernel\_boot() has been invoked),  
→use only normal grace-period primitives. No  
→effect on CONFIG\_TINY\_RCU kernels.

rcupdate.rcu\_task\_ipi\_delay= [KNL]  
Set time in jiffies during which RCU tasks  
→will avoid sending IPIs, starting with the  
→beginning of a given grace period. Setting a large number avoids disturbing real-time  
→workloads, but lengthens grace periods.

rcupdate.rcu\_task\_stall\_timeout= [KNL]  
Set timeout in jiffies for RCU task stall  
→warning messages. Disable with a value less than  
→or equal to zero.

rcupdate.rcu\_self\_test= [KNL]  
Run the RCU early boot self tests

rdinit= [KNL]  
Format: <full\_path>  
Run specified binary instead of /init from  
→the ramdisk, used for early userspace startup. See  
→initrd.

rdrand= [X86]  
force - Override the decision by the kernel  
→to hide the advertisement of RDRAND support  
→(this affects certain AMD processors because of  
→buggy BIOS support, specifically around the  
→suspend/resume path).

rdt= [HW,X86,RDT]  
Turn on/off individual RDT features. List  
→is:

```

    cmt, mbmtotal, mbmlocal, l3cat, l3cdp,
↳l2cat, l2cdp,
    mba.
    E.g. to turn on cmt and turn off mba use:
        rdt=cmt,!mba

    reboot=
    [KNL]
    Format (x86 or x86_64):
        [w[arm] | c[old] | h[ard] | s[oft]]
↳| g[pio]] \
        [[,]s[mp]#### \
↳t[riple] | e[fi] | p[ci]] \
        [[,]f[orce]
    Where reboot_mode is one of warm (soft) or
↳cold (hard) or gpio
        (prefix with 'panic_' to
↳set mode for panic
        reboot only),
    reboot_type is one of bios, acpi, kbd,
↳ triple, efi, or pci,
    reboot_force is either force or not
↳specified,
    reboot_cpu is s[mp]#### with ####
↳being the processor
        to be used for booting.

    relax_domain_level=
    [KNL, SMP] Set scheduler's default relax_
↳domain_level.
    See Documentation/admin-guide/cgroup-v1/
↳cpuset.rst.

    reserve=
    [KNL,BUGS] Force kernel to ignore I/O ports
↳or memory
    Format: <base1>,<size1>[,<base2>,<size2>,...
↳]
    Reserve I/O ports or memory so the kernel
↳won't use
    them. If <base> is less than 0x10000, the
↳region
    is assumed to be I/O ports; otherwise it is
↳memory.

    reservetop=
    [X86-32]
    Format: nn[KMG]
    Reserves a hole at the top of the kernel
↳virtual
    address space.

    reservelow=
    [X86]

```

Format: nn[K]  
 Set the amount of memory to reserve for  
 → BIOS at the bottom of the address space.

reset\_devices [KNL] Force drivers to reset the underlying  
 → device during initialization.

resume= [SWSUSP]  
 Specify the partition device for software  
 → suspend

Format:  
 {/dev/<dev> | PARTUUID=<uuid> | <int>:<int>  
 → | <hex>}

resume\_offset= [SWSUSP]  
 Specify the offset from the beginning of  
 → the partition given by "resume=" at which the swap header  
 → is located, in <PAGE\_SIZE> units (needed only for swap  
 → files).  
 See Documentation/power/  
 → swsusp-and-swap-files.rst

resumedelay= [HIBERNATION] Delay (in seconds) to pause  
 → before attempting to read the resume files

resumewait [HIBERNATION] Wait (indefinitely) for  
 → resume device to show up.  
 Useful for devices that are detected  
 → asynchronously (e.g. USB and MMC devices).

hibernate= [HIBERNATION]  
 noresume Don't check if there's a  
 → hibernation image present during boot.  
 nocompress Don't compress/decompress  
 → hibernation images.  
 no Disable hibernation and resume.  
 protect\_image Turn on image protection during  
 → restoration (that will set all pages holding  
 → image data during restoration read-only).

retain\_initrd [RAM] Keep initrd memory after extraction

```
rfkill.default_state=
    0      "airplane mode". All wifi, bluetooth,
↳wimax, gps, fm,
           etc. communication is blocked by default.
    1      Unblocked.

rfkill.master_switch_mode=
    0      The "airplane mode" button does nothing.
    1      The "airplane mode" button toggles between
↳everything
           blocked and the previous configuration.
    2      The "airplane mode" button toggles between
↳everything
           blocked and everything unblocked.

rhash_entries= [KNL,NET]
               Set number of hash buckets for route cache

ring3mwait=disable
↳on supported [KNL] Disable ring 3 MONITOR/MWAIT feature
               CPUs.

ro [KNL] Mount root device read-only on boot

rodata= [KNL]
↳(default). on Mark read-only kernel memory as read-only
↳debugging. off Leave read-only kernel memory writable for
```

```
rockchip.usb_uart
↳designated usb port
↳of the
↳of the usb
↳disabled.
               Enable the uart passthrough on the
               on Rockchip SoCs. When active, the signals
               debug-uart get routed to the D+ and D- pins
               port and the regular usb controller gets
```

```
root= [KNL] Root filesystem
↳C. See name_to_dev_t comment in init/do_mounts.
```

```
rootdelay= [KNL] Delay (in seconds) to pause before
↳attempting to mount the root filesystem
```

```
rootflags= [KNL] Set root filesystem mount option
↳string
```

rootfstype=	[KNL] Set root filesystem type
rootwait	[KNL] Wait (indefinitely) for root device
↳to show up.	
↳asynchronously	Useful for devices that are detected
	(e.g. USB and MMC devices).
rproc_mem=nn[KMG][@address]	[KNL,ARM,CMA] Remoteproc physical memory
↳block.	
↳image,	Memory area to be used by remote processor
	managed by CMA.
rw	[KNL] Mount root device read-write on boot
S	[KNL] Run init in single mode
s390_iommu=	[HW,S390]
strict	Set s390 IOTLB flushing mode
↳will result in	With strict flushing every unmap operation
↳before reuse,	an IOTLB flush. Default is lazy flushing
	which is faster.
sal100ir	[NET]
	See drivers/net/irda/sal100_ir.c.
sbni=	[NET] Granich SBNI12 leased line adapter
sched_debug	[KNL] Enables verbose scheduler debug
↳messages.	
schedstats=	[KNL,X86] Enable or disable scheduled
↳statistics.	
↳feature	Allowed values are enable and disable. This
↳scheduler	incurs a small amount of overhead in the
↳tuning.	but is useful for debugging and performance
sched_thermal_decay_shift=	[KNL, SMP] Set a decay shift for scheduler
↳thermal	
↳follows the	pressure signal. Thermal pressure signal

default decay period of other scheduler pelt signals(usually 32 ms but configurable).  
 ↳Setting  
 ↳the decay  
 ↳the shift  
 ↳32 ms  
 ↳pressure decay pr

`sched_thermal_decay_shift` will left shift  
 period for the thermal pressure signal by  
 value.  
 i.e. with the default pelt decay period of  
`sched_thermal_decay_shift` thermal

1	64 ms
2	128 ms

and so on.  
 Format: integer between 0 and 10  
 Default is 0.

`skew_tick=`  
 ↳cpu to mitigate  
 ↳and/or RCU lock  
 ↳MAXSMP set.  
 ↳CMDLINE="skew\_tick=1"  
 ↳should only be  
 ↳RT) workloads.

[KNL] Offset the periodic timer tick per  
`xtime_lock` contention on larger systems,  
 contention on all systems with CONFIG\_

Format: { "0" | "1" }  
 0 -- disable. (may be 1 via CONFIG\_  
 1 -- enable.  
 Note: increases power consumption, thus  
 enabled if running jitter sensitive (HPC/

`security=`  
 ↳module to  
 ↳the

[SECURITY] Choose a legacy "major" security  
 enable at boot. This has been deprecated by  
 "lsm=" parameter.

`selinux=`  
 ↳time.

[SELINUX] Disable or enable SELinux at boot  
 Format: { "0" | "1" }  
 See security/selinux/Kconfig help text.  
 0 -- disable.  
 1 -- enable.  
 Default value is 1.

`apparmor=`  
 ↳boot time

[APPARMOR] Disable or enable AppArmor at  
 Format: { "0" | "1" }  
 See security/apparmor/Kconfig help text

0 -- disable.  
 1 -- enable.  
 Default value is set via kernel config.

→ option.

serialnumber	[BUGS=X86-32]
shapers=	[NET] Maximal number of shapers.
simeth= simscsi=	[IA-64]
slram=	[HW,MTD]
slab_nomerge	[MM] Disable merging of slabs with similar size.

→ May be necessary if there is some reason to

→ distinguish allocs to different slabs, especially in

→ hardened environments where the risk of heap

→ overflows and layout control by attackers can usually be

→ reduce frustrated by disabling merging. This will

→ single most of the exposure of a heap attack to a

→ on their cache (risks via metadata attacks are mostly

unchanged). Debug options disable merging

own. For more information see Documentation/vm/

→ slub.rst.

slab_max_order=	[MM, SLAB] Determines the maximum allowed order for
-----------------	--

→ slabs. A high setting may cause OOMs due to memory

fragmentation. Defaults to 1 for systems

→ with more than 32MB of RAM, 0 otherwise.

slub_debug[=options[,slabs]]	[MM, SLUB] Enabling slub_debug allows one to determine
------------------------------	---

→ the culprit if slab objects become corrupted.

→ Enabling slub\_debug can create guard zones around

→ objects and

→ tracks the `may poison objects when not in use. Also`  
`last alloc / free. For more information see`  
`Documentation/vm/slub.rst.`

`slub_memcg_sysfs= [MM, SLUB]`  
→ directories for `Determines whether to enable sysfs`  
→ disable. `memory cgroup sub-caches. 1 to enable, 0 to`  
→ MEMCG\_SYSFS\_ON. `The default is determined by CONFIG_SLUB_`  
→ number of debug `Enabling this can lead to a very high`  
`directories and files being created under`  
`/sys/kernel/slub.`

`slub_max_order= [MM, SLUB]`  
→ slabs. `Determines the maximum allowed order for`  
`A high setting may cause OOMs due to memory`  
`fragmentation. For more information see`  
`Documentation/vm/slub.rst.`

`slub_min_objects= [MM, SLUB]`  
→ SLUB will `The minimum number of objects per slab.`  
→ order to `increase the slab order up to slub_max_`  
→ contain `generate a sufficiently large slab able to`  
→ the number `the number of objects indicated. The higher`  
→ tracking slabs `of objects the smaller the overhead of`  
→ acquired. `and the less frequently locks need to be`  
→ slub.rst. `For more information see Documentation/vm/`

`slub_min_order= [MM, SLUB]`  
→ Must be `Determines the minimum page order for slabs.`  
`lower than slub_max_order.`  
→ slub.rst. `For more information see Documentation/vm/`

`slub_nomerge [MM, SLUB]`  
→ for legacy. `Same with slab_nomerge. This is supported`  
`See slab_nomerge for more information.`

```

smart2=                [HW]
                        Format: <io1>[,<io2>[,...,<io8>]]

smc-ircc2.nopnp        [HW] Don't use PNP to discover SMC
→devices
smc-ircc2.ircc_cfg=    [HW] Device configuration I/O port
smc-ircc2.ircc_sir=    [HW] SIR base I/O port
smc-ircc2.ircc_fir=    [HW] FIR base I/O port
smc-ircc2.ircc_irq=    [HW] IRQ line
smc-ircc2.ircc_dma=    [HW] DMA channel
smc-ircc2.ircc_transceiver= [HW] Transceiver type:
→pin select)          0: Toshiba Satellite 1800 (GP data
                        1: Fast pin select (default)
                        2: ATC IRMode

smt                    [KNL,S390] Set the maximum number of
→threads (logical     CPUs) to use per physical CPU on systems
→capable of           symmetric multithreading (SMT). Will be
→capped to the        actual hardware limit.
                        Format: <integer>
                        Default: -1 (no limit)

softlockup_panic=     [KNL] Should the soft-lockup detector
→generate panics.     Format: 0 | 1
                        A value of 1 instructs the soft-lockup
→detector             to panic the machine when a soft-lockup
→occurs. It is        also controlled by the kernel.softlockup_
→panic sysctl         and CONFIG_BOOTPARAM_SOFTLOCKUP_PANIC,
→which is the         respective build-time switch to that
→functionality.

softlockup_all_cpu_backtrace=
→generate             [KNL] Should the soft-lockup detector
                        backtraces on all cpus.
                        Format: 0 | 1

sonypi.*=             [HW] Sony Programmable I/O Control Device
→driver

```

See Documentation/admin-guide/laptops/  
sonypi.rst

spectre\_v2=  
2  
from

on - unconditionally enable, implies spectre\_v2\_user=on  
off - unconditionally disable, implies spectre\_v2\_user=off  
auto - kernel detects whether your CPU  
model is vulnerable

a  
the  
of the  
the

mitigation  
attacks.

kernel and

manually:

branches  
retpoline  
think

retpoline - replace indirect  
retpoline,generic - google's original  
retpoline,amd - AMD-specific minimal

Not specifying this option is equivalent to spectre\_v2=auto.

spectre\_v2\_user=

↳2 [X86] Control mitigation of Spectre variant, (indirect branch speculation) vulnerability, user space tasks

↳between

↳mitigations. Is on - Unconditionally enable, enforced by spectre\_v2=on

↳mitigations. Is off - Unconditionally disable, enforced by spectre\_v2=off

↳enabled, prctl - Indirect branch speculation is, but mitigation can be enabled via, per thread. The mitigation, is inherited on fork.

↳prctl

↳control state

↳STIBP is prctl,ibpb - Like "prctl" above, but only, controlled per thread. IBPB is, always when switching between, space processes.

↳issued

↳different user

↳seccomp seccomp - Same as "prctl" above, but all, threads will enable the, they explicitly opt out.

↳seccomp

↳mitigation unless

↳STIBP is seccomp,ibpb - Like "seccomp" above, but only, controlled per thread. IBPB is, always when switching between, user space processes.

↳issued

↳different

↳depending on auto - Kernel selects the mitigation, the available CPU features and,

↳vulnerability.

Default mitigation:  
If CONFIG\_SECCOMP=y then "seccomp",  
→ otherwise "prctl"

Not specifying this option is equivalent to spectre\_v2\_user=auto.

spec\_store\_bypass\_disable=  
→ Disable mitigation [HW] Control Speculative Store Bypass (SSB)  
(Speculative Store Bypass vulnerability)  
→ against a Certain CPUs are vulnerable to an exploit  
→ optimization known as "Speculative Store Bypass" in which  
→ recent stores to the same memory location may not be  
→ observed by later loads during speculative execution.  
→ The idea is that such stores are unlikely and that  
→ they can be detected prior to instruction retirement  
→ at the end of a particular speculation execution  
→ window.

→ forwarded In vulnerable processors, the speculatively  
→ attack, for store can be used in a cache side channel  
→ attacker does not example to read memory to which the  
→ code). directly have access (e.g. inside sandboxed

→ Speculative Store This parameter controls whether the  
Bypass optimization is used.

On x86 the options are:

→ Speculative Store Bypass	on	- Unconditionally disable
→ Speculative Store Bypass	off	- Unconditionally enable
→ model contains an	auto	- Kernel detects whether the CPU

implementation of Speculative Store Bypass and mitigation. If the CPU is not vulnerable, "off" is selected. If the CPU is vulnerable the default mitigation is architecture and Kconfig dependent. See below.

`prctl` - Control Speculative Store Bypass via `prctl`. Speculative Store Bypass is enabled for a process by default. The state of the control is inherited on fork.

`seccomp` - Same as "prctl" above, but all seccomp threads will disable SSB unless they explicitly opt out.

Default mitigations:  
 X86: If `CONFIG_SECCOMP=y` "seccomp", otherwise "prctl"

On powerpc the options are:

`on,auto` - On Power8 and Power9 insert a barrier on kernel entry and exit. On Power7 perform a software flush on kernel entry and exit.

`off` - No action.

Not specifying this option is equivalent to `spec_store_bypass_disable=auto`.

```

spia_io_base= [HW,MTD]
spia_fio_base=
spia_pedr=
spia_peddr=

split_lock_detect=
[X86] Enable split lock detection

```

When enabled (and if hardware support is present), atomic instructions that access data across cache

→line boundaries will result in an alignment.

→check exception.

off - not enabled

warn - the kernel will emit rate limited warnings about applications triggering the #AC exception. This mode is the default on CPUs that supports split lock detection.

→#AC

→default on CPUs

→detection.

fatal - the kernel will send SIGBUS to applications that trigger the #AC exception.

→applications

If an #AC exception is hit in the kernel or in firmware (i.e. not while executing in user mode) the kernel will oops in either "warn" or "fatal" mode.

→in

→mode)

→"fatal"

mode.

srbd= [X86,INTEL] Control the Special Register Buffer Data (SRBDS) mitigation.

→Sampling

Certain CPUs are vulnerable to an MDS-like exploit which can leak bits from the random number generator.

By default, this issue is mitigated by microcode. However, the microcode fix can cause the RDRAND and RDSEED instructions to become much slower. Among other effects, this will result in reduced throughput from /dev/urandom.

→cause

→urandom.

The microcode mitigation can be disabled with the following option:

off: Disable mitigation and remove performance impact to RDRAND and RDSEED

→RDSEED

srcutree.counter\_wrap\_check [KNL]  
 Specifies how frequently to check for grace-period sequence counter wrap for the srcu\_data structure's ->srcu\_gp\_seq\_needed field.  
 kernel will bits  
 The greater the number of bits set in this parameter, the less frequently counter wrap will be checked for. Note that the bottom two bits are ignored.

srcutree.exp\_holdoff [KNL]  
 Specifies how many nanoseconds must elapse since the end of the last SRCU grace period for a given srcu\_struct until the next normal SRCU grace period will be considered for automatic expediting. Set to zero to disable automatic expediting.

ssbd= [ARM64,HW]  
 Speculative Store Bypass Disable control  
 On CPUs that are vulnerable to the Speculative Store Bypass vulnerability and offer a firmware based mitigation, this parameter indicates how the mitigation should be used:  
 force-on: Unconditionally enable mitigation for both kernel and userspace  
 force-off: Unconditionally disable mitigation for both kernel and userspace  
 kernel: Always enable mitigation in the kernel, and offer a prctl interface to allow userspace to register its interest in being mitigated too.

stack\_guard\_gap= [MM]  
 override the default stack gap protection.  
 The value

is in page units and it defines how many pages prior to (for stacks growing down) resp. after (for stacks growing up) the main stack are reserved for no other mapping. Default value is 256 pages.

`stacktrace` [FTRACE]  
Enabled the stack tracer on boot up.

`stacktrace_filter=[function-list]` [FTRACE] Limit the functions that the stack tracer will trace at boot up. function-list is a comma separated list of functions. This list can be changed at run time by the `stack_trace_filter` file in the `debugfs` tracing directory. Note, this enables stack tracing and the `stacktrace` above is not needed.

`sti=` [PARISC,HW]  
Format: <num>  
Set the STI (builtin display/keyboard on the HP-PARISC machines) console (graphic card) which should be used as the initial boot-console. See also comment in `drivers/video/console/sticore.c`.

`sti_font=` [HW]  
See comment in `drivers/video/console/sticore.c`.

`stifb=` [HW]  
Format: `bpp:<bpp1>[:<bpp2>[:<bpp3>...]]`

`sunrpc.min_resvport=`  
`sunrpc.max_resvport=` [NFS,SUNRPC]  
SunRPC servers often require that client requests originate from a privileged port (i.e. a port in the range `0 < portnr < 1024`). An administrator who wishes to reserve some of these

ports for other uses may adjust the range, that the kernel's sunrpc client considers to be privileged using these two parameters to set the minimum and maximum port values.

```
sunrpc.svc_rpc_per_connection_limit=
[NFS,SUNRPC]
```

Limit the number of requests that the server will process in parallel from a single connection. The default value is 0 (no limit).

```
sunrpc.pool_mode=
[NFS]
```

Control how the NFS server code allocates CPUs to service thread pools. Depending on how many NICs you have and where their interrupts are bound, this option will affect which CPUs will do NFS serving. Note: this parameter cannot be changed while the NFS server is running.

appropriate mode	auto	the server chooses an automatically using heuristics
all CPUs	global	a single global pool contains
(equivalent	percpu	one pool for each CPU
	pernode	one pool for each NUMA node to global on non-NUMA machines)

```
sunrpc.tcp_slot_table_entries=
sunrpc.udp_slot_table_entries=
[NFS,SUNRPC]
```

Sets the upper limit on the number of simultaneous RPC calls that can be sent from the client to a server. Increasing these values may allow you to improve throughput, but will also increase the

amount of memory reserved for use by the `client`.

`suspend.pm_test_delay=`  
[SUSPEND]  
Sets the number of seconds to remain in a `suspend test` mode before resuming the system (see `/sys/power/pm_test`). Only available when `CONFIG_PM_DEBUG` is set. Default value is 5.

`svm=`  
[PPC]  
Format: { on | off | y | n | 1 | 0 }  
This parameter controls use of the Protected Execution Facility on pSeries.

`swappiness=[0|1]`  
[KNL] Enable accounting of swap in memory controller if no parameter or 1 is given or it if 0 is given (See Documentation/admin-guide/cgroup-v1/memory.rst)

`swiotlb=`  
[ARM,IA-64,PPC,MIPS,X86]  
Format: { <int> | force | noforce }  
<int> -- Number of I/O TLB slabs  
force -- force using of bounce buffers even if they wouldn't be automatically used by the kernel  
noforce -- Never use bounce buffers (for debugging)

`switches=`  
[HW,M68k]

`sysctl.*=`  
[KNL]  
Set a sysctl parameter, right before loading the init process, as if the value was written to the respective `/proc/sys/...` file. Both `'.'` and `'/'` are recognized as separators. Unrecognized parameters and invalid values are reported in the kernel log. Sysctls registered later by a loaded module cannot be set this way.

Example: `sysctl.vm.swappiness=40`

`sysfs.deprecated=0|1` [KNL]  
 Enable/disable old style sysfs layout for  
 →old udev  
 →enabled  
 →this option  
 →compiled)  
 →V2 set in  
 on older distributions. When this option is  
 very new udev will not work anymore. When  
 is disabled (or `CONFIG_SYSFS_DEPRECATED` not  
 in older udev will not work anymore.  
 Default depends on `CONFIG_SYSFS_DEPRECATED_`  
 the kernel configuration.

`sysrq_always_enabled`  
 [KNL]  
 Ignore sysrq setting - this boot parameter  
 →will  
 →sysrq.  
 neutralize any effect of `/proc/sys/kernel/`  
 Useful for debugging.

`tcpmhash_entries=` [KNL,NET]  
 Set the number of `tcp_metrics_hash` slots.  
 Default value is 8192 or 16384 depending on  
 →total  
 →metrics  
 →ip-sysctl.rst  
 →details.  
 ram pages. This is used to specify the TCP  
 cache size. See `Documentation/networking/`  
 "tcp\_no\_metrics\_save" section for more

`tdfx=` [HW,DRM]  
  
`test_suspend=` [SUSPEND][,N]  
 Specify "mem" (for Suspend-to-RAM) or  
 →"standby" (for  
 →type freeze)  
 →startup with  
 →of times.  
 standby suspend) or "freeze" (for suspend  
 as the system sleep state during system  
 the optional capability to repeat N number  
 The system is woken from this state using a  
 wakeup-capable RTC alarm.

`thash_entries=` [KNL,NET]  
 Set number of hash buckets for TCP  
 →connection

`thermal.act=` [HW,ACPI]  
-1: disable all active trip points in all thermal zones  
<degrees C>: override all lowest active trip points

`thermal.crt=` [HW,ACPI]  
-1: disable all critical trip points in all thermal zones  
<degrees C>: override all critical trip points

`thermal.nocrt=` [HW,ACPI]  
Set to disable actions on ACPI thermal zone critical and hot trip points.

`thermal.off=` [HW,ACPI]  
1: disable ACPI thermal control

`thermal.psv=` [HW,ACPI]  
-1: disable all passive trip points  
<degrees C>: override all passive trip points to this value

`thermal.tzp=` [HW,ACPI]  
Specify global default ACPI thermal zone polling rate  
<deci-seconds>: poll all this frequency  
0: no polling (default)

`threadirqs` [KNL]  
Force threading of all interrupt handlers except those marked explicitly `IRQF_NO_THREAD`.

`topology=` [S390]  
Format: {off | on}  
Specify if the kernel should make use of topology information if the hardware supports this. The scheduler will make use of this information and e.g. base its process migration decisions on it.  
Default is on.

`topology_updates=` [KNL, PPC, NUMA]  
Format: {off}

Specify if the kernel should ignore (off) topology updates sent by the hypervisor to `lpar.boot.ignore_topology_updates`.

`lpar.boot.ignore_topology_updates` [KVM]

Prevent the CPU-hotplug component of `lpar.boot` until after `init` has spawned.

`lpar.boot.ignore_topology_updates` [HW,PS2]

`lpar.boot.ignore_topology_updates` [HW,TPM]

Format: integer pcr id

Specify that at suspend time, the tpm driver should extend the specified pcr with zeros, as a workaround for some chips which fail to flush the last written pcr on `TPM_SaveState`. This will guarantee that all the other pcrs are saved.

`lpar.boot.trace_buf_size`=nn[KMG]

[FTRACE] will set tracing buffer size on `lpar.boot` each cpu.

`lpar.boot.trace_event`=[event-list]

[FTRACE] Set and start specified trace `lpar.boot` events in order to facilitate early boot debugging. The `lpar.boot` event-list is a comma separated list of trace events to `lpar.boot` enable. See also `Documentation/trace/events.rst`

`lpar.boot.trace_options`=[option-list]

[FTRACE] Enable or disable tracer options `lpar.boot` at boot. The option-list is a comma delimited list `lpar.boot` of options that can be enabled or disabled just as if `lpar.boot` you were to echo the option name into `/sys/kernel/debug/tracing/trace_options`

For example, to enable `stacktrace` option `lpar.boot` (to dump the stack trace of each event), add to the `lpar.boot` command line:

```
lpar.boot trace_options=stacktrace
```

→ "trace options" See also Documentation/trace/ftrace.rst, section.

→ as the `tp_printk[FTRACE]` Have the tracepoints sent to `printk` as well, tracing ring buffer. This is useful for where the system hangs or reboots and does not give the option for reading the tracing buffer or performing a `ftrace_dump_on_oops`.

→ `printk,` To turn off having tracepoints sent to `printk`,  
`echo 0 > /proc/sys/kernel/tracepoint_printk`  
Note, echoing 1 into this file without the `tracepoint_printk` kernel cmdline option has no effect.

→ activating high **\*\* CAUTION \*\***  
Having tracepoints sent to `printk()` and frequency tracepoints such as `irq` or `sched`, can cause the system to live lock.

→ "tracing\_on" `traceoff_on_warning` [FTRACE] enable this option to disable tracing when a warning is hit. This turns off "tracing\_on". Tracing can be enabled again by echoing '1' into the file located in `/sys/kernel/debug/tracing/`

→ trace before This option is useful, as it disables the the WARNING dump is called, which prevents the trace to be filled with content caused by the warning output.

→ the `sysctl` This option can also be set at run time via the option: `kernel/traceoff_on_warning`

transparent\_hugepage=  
 [KNL]  
 Format: [always|advise|never]  
 Can be used to control the default behavior of the system with respect to transparent hugepages. See Documentation/admin-guide/mm/transhuge. for more details.

→rst

tsc=  
 →TSC. Disable clocksource stability checks for reliable, this disables clocksource verification at runtime, as well as the stability checks done at bootup. Used to enable high-resolution timer mode on older hardware, and in virtualized environment. →accounting. [x86] noirqtime: Do not use TSC to do irq accounting. Used to run time disable IRQ\_TIME\_ACCOUNTING on any platforms where RDTSC is slow and this can add overhead. →accounting [x86] unstable: mark the TSC clocksource as unstable, this marks the TSC unconditionally unstable at bootup and avoids any further wobbles once the TSC watchdog notices. →watchdog. Used in situations with strict latency requirements (where interruptions from clocksource watchdog are not acceptable).

tsc\_early\_khz=  
 →the given [X86] Skip early TSC calibration and use value instead. Useful when the early TSC frequency discovery procedure is not reliable, such as on overclocked systems with CPUID.16h support and partial CPUID.15h support.

Format: <unsigned int>

tsx= [X86] Control Transactional Synchronization Extensions (TSX) feature in Intel processors that support TSX control.

This parameter controls the TSX feature.

The options are:

- on - Enable TSX on the system.

Although there are mitigations for all known security vulnerabilities, TSX has been known to be an accelerator for several previous speculation-related CVEs, and so there may be unknown security risks associated with leaving it enabled.

- off - Disable TSX on the system. (Note option takes effect only on newer CPUs which are not vulnerable to MDS, i.e., have MSR\_IA32\_ARCH\_CAPABILITIES.MDS\_NO=1 and which get the new IA32\_TSX\_CTRL MSR through a microcode update. This new MSR allows for the reliable deactivation of the TSX functionality.)

- auto - Disable TSX if X86\_BUG\_TAA is present, otherwise enable TSX on the system.

Not specifying this option is equivalent to tsx=off.

See Documentation/admin-guide/hw-vuln/tsx\_async\_abort.rst for more details.

tsx\_async\_abort= [X86,INTEL] Control mitigation for the Async Abort (TAA) vulnerability.

Similar to Micro-architectural Data Sampling (MDS), certain CPUs that support Transactional Synchronization Extensions (TSX) are vulnerable to an exploit against CPU internal buffers which can forward information to a disclosure gadget under certain conditions.

In vulnerable processors, the speculatively forwarded data can be used in a cache side channel attack, to access data to which the attacker does not have direct access.

This parameter controls the TAA mitigation. The options are:

- `full` - Enable TAA mitigation on vulnerable CPUs if TSX is enabled.
- `full,nosmt` - Enable TAA mitigation and disable SMT on vulnerable CPUs. If TSX is not disabled because CPU is not vulnerable to cross-thread TAA attacks.
- `off` - Unconditionally disable TAA mitigation.

On MDS-affected machines, `tsx_async_abort=off` can be prevented by an active MDS mitigation as both vulnerabilities are mitigated with the same mechanism so in order to disable this mitigation, you need to specify `tsx_async_abort=off` too.

On CPUs which are MDS affected

and deploy MDS mitigation, TAA mitigation.  
is not required and doesn't provide any additional mitigation.  
For details see:  
Documentation/admin-guide/hw-vuln/tsx\_async\_

abort.rst

turbografx.map[2|3]= [HW,JOY]  
TurboGraFX parallel port interface  
Format:  
<port#>,<js1>,<js2>,<js3>,<js4>,<js5>,<js6>,  
<js7>  
See also Documentation/input/devices/  
joystick-parport.rst

udbg-immortal [PPC] When debugging early kernel crashes,  
that happen after console\_init() and before a  
proper console driver takes over, this boot  
options might help "seeing" what's going on.

uhash\_entries= [KNL,NET]  
Set number of hash buckets for UDP/UDP-Lite  
connections

uhci-hcd.ignore\_oc=  
[USB] Ignore overcurrent events (default N).  
Some badly-designed motherboards generate  
lots of bogus events, for ports that aren't wired to  
anything. Set this parameter to avoid log  
spamming.  
Note that genuine overcurrent events won't  
be reported either.

unknown\_nmi\_panic  
[X86] Cause panic on unknown NMI.

usbcore.authorized\_default=  
[USB] Default USB device authorization:  
(default -1 = authorized except for  
wireless USB,  
0 = not authorized, 1 = authorized, 2 =  
authorized if device connected to internal port)

`usbcore.autosuspend=`  
[USB] The autosuspend time delay (in seconds) used for newly-detected USB devices (default 2). This is the time required before an idle device will be autosuspended. Devices for which the delay is set to a negative value won't be autosuspended at all.

`usbcore.usbfs_snoop=`  
[USB] Set to log all usbfs traffic (default 0 = off).

`usbcore.usbfs_snoop_max=`  
[USB] Maximum number of bytes to snoop in each URB (default = 65536).

`usbcore.blinkenlights=`  
[USB] Set to cycle leds on hubs (default 0 = off).

`usbcore.old_scheme_first=`  
[USB] Start with the old device initialization scheme (default 0 = off).

`usbcore.usbfs_memory_mb=`  
[USB] Memory limit (in MB) for buffers allocated by usbfs (default = 16, 0 = max = 2047).

`usbcore.use_both_schemes=`  
[USB] Try the other device initialization scheme if the first one fails (default 1 = enabled).

`usbcore.initial_descriptor_timeout=`  
[USB] Specifies timeout for the initial 64-byte USB\_REQ\_GET\_DESCRIPTOR request in milliseconds (default 5000 = 5.0 seconds).

`usbcore.nousb` [USB] Disable the USB subsystem

`usbcore.quirks=`

→the built-in [USB] A list of quirk entries to augment  
→separated by usb core quirk list. List entries are  
→4-digit hex commas. Each entry has the form  
→letter VendorID:ProductID:Flags. The IDs are  
→if it is numbers and Flags is a set of letters. Each  
→letters have will change the built-in quirk; setting it  
→(string clear and clearing it if it is set. The  
→fetched using the following meanings:  
→can't resume a = USB\_QUIRK\_STRING\_FETCH\_255  
→instead); descriptors must not be  
→can't handle b = USB\_QUIRK\_RESET\_RESUME (device  
→(device can't correctly so reset it  
→Interface c = USB\_QUIRK\_NO\_SET\_INTF (device  
→be reset Set-Interface requests);  
→use reset); d = USB\_QUIRK\_CONFIG\_INTF\_STRINGS  
→(device has handle its Configuration or  
→than the strings);  
→can't handle e = USB\_QUIRK\_RESET (device can't  
→interfaces); (e.g morph devices), don't  
→needs a pause f = USB\_QUIRK\_HONOR\_BNUMINTERFACES  
→after we read more interface descriptions,  
→BINTERVAL (For bNumInterfaces count, and  
talking to these  
g = USB\_QUIRK\_DELAY\_INIT (device  
during initialization,  
the device descriptor);  
h = USB\_QUIRK\_LINEAR\_UFRAME\_INTR  
high speed and super speed

```

↳interrupt
↳USB 3.0 spec
↳microframes (1
↳microseconds) to be
↳report their
↳this
↳exponent
↳calculation);
↳(device can't
↳descriptor
↳(device
↳ignore
↳handle Link
↳BINTERVAL
↳bInterval as linear
↳0
↳(Device needs
↳suspend to
↳(Device needs a
↳message);
↳needs extra
↳port);
endpoints, the USB 2.0 and
require the interval in
microframe = 125
calculated as interval = 2 ^
(bInterval-1).
Devices with this quirk
bInterval as the result of
calculation instead of the
variable used in the
i = USB_QUIRK_DEVICE_QUALIFIER
handle device_qualifier
requests);
j = USB_QUIRK_IGNORE_REMOTE_WAKEUP
generates spurious wakeup,
remote wakeup capability);
k = USB_QUIRK_NO_LPM (device can't
Power Management);
l = USB_QUIRK_LINEAR_FRAME_INTR_
(Device reports its
frames instead of the USB 2.
calculation);
m = USB_QUIRK_DISCONNECT_SUSPEND
to be disconnected before
prevent spurious wakeup);
n = USB_QUIRK_DELAY_CTRL_MSG
pause after every control
o = USB_QUIRK_HUB_SLOW_RESET (Hub
delay after resetting its

```

Example: quirks=0781:5580:bk,0a5c:5834:gij

`usbhid.mousepoll=`  
[USBHID] The interval which mice are to be polled at.

`usbhid.jspoll=`  
[USBHID] The interval which joysticks are to be polled at.

`usbhid.kbpoll=`  
[USBHID] The interval which keyboards are to be polled at.

`usb-storage.delay_use=`  
[UMS] The delay in seconds before a new device is scanned for Logical Units (default 1).

`usb-storage.quirks=`  
[UMS] A list of quirks entries to supplement or override the built-in `unusual_devs` list. List entries are separated by commas. Each entry has the form `VID:PID:Flags` where VID and PID are Vendor and Product ID values (4-digit hex numbers) and Flags is a set of characters, each corresponding to a common `usb-storage` quirk flag as follows:

- a = `SANE_SENSE` (collect more than 18 bytes of sense data, not on uas);
- b = `BAD_SENSE` (don't collect more than 18 bytes of sense data, not on uas);
- c = `FIX_CAPACITY` (decrease the reported device capacity by one sector);
- d = `NO_READ_DISC_INFO` (don't use `READ_DISC_INFO` command, not on uas);
- e = `NO_READ_CAPACITY_16` (don't use `READ_CAPACITY_16` command);
- f = `NO_REPORT_OPCODES` (don't use report opcodes

```

command, uas only);
g = MAX_SECTORS_240 (don't transfer
more than
only);
the
one
odd);
this
report luns
and
on uas);
more
a time,
of the
not on uas);
on uas);
is ON
reports
on uas);
one
and ATA(16)
uas driver);
whether the
command, uas only);
h = CAPACITY_HEURISTICS (decrease
reported device capacity by
sector if the number is
i = IGNORE_DEVICE (don't bind to
device);
j = NO_REPORT_LUNS (don't use
command, uas only);
l = NOT_LOCKABLE (don't try to lock
unlock ejectable media, not
m = MAX_SECTORS_64 (don't transfer
than 64 sectors = 32 KB at
not on uas);
n = INITIAL_READ10 (force a retry
initial READ(10) command,
o = CAPACITY_OK (accept the capacity
reported by the device, not
p = WRITE_CACHE (the device cache
by default, not on uas);
r = IGNORE_RESIDUE (the device
bogus residue values, not
s = SINGLE_LUN (the device has only
Logical Unit);
t = NO_ATA_1X (don't allow ATA(12)
commands, uas only);
u = IGNORE_UAS (don't bind to the
w = NO_WP_DETECT (don't test
medium is write-protected).

```

`→SYNCHRONIZE_CACHE` `y = ALWAYS_SYNC` (issue a `□`  
even if the device claims `□`  
`→no cache,`  
not on uas)  
Example: `quirks=0419:aaf5:rl,0421:0433:rc`

`user_debug=` `[KNL,ARM]`  
Format: `<int>`  
See `arch/arm/Kconfig.debug` help text.  
1 - undefined instruction events  
2 - system calls  
4 - invalid data aborts  
8 - SIGSEGV faults  
16 - SIGBUS faults  
Example: `user_debug=31`

`userpte=` `[X86]` Flags controlling user PTE `□`  
`→allocations.`  
`→in` `nohigh = do not allocate PTE pages` `□`  
`→setting` `HIGHMEM` regardless of `□`  
of `CONFIG_HIGHPTTE`.

`vdso=` `[X86,SH]`  
On `X86_32`, this is an alias for `vdso32=`. `□`  
`→Otherwise:`  
`vdso=1: enable VDSO (the default)`  
`vdso=0: disable VDSO mapping`

`vdso32=` `[X86]` Control the 32-bit vDSO  
`vdso32=1: enable 32-bit VDSO`  
`vdso32=0 or vdso32=2: disable 32-bit VDSO`  
`→for more` See the help text for `CONFIG_COMPAT_VDSO` `□`  
`→default is` details. If `CONFIG_COMPAT_VDSO` is set, the `□`  
`→vdso32=1.` `vdso32=0`; otherwise, the default is `□`

`→vdso32=2 is an` For compatibility with older kernels, `□`  
alias for `vdso32=0`.

`→says:` Try `vdso32=0` if you encounter an error that `□`

dl\_main: Assertion `(void \*) ph->p\_vaddr ==  
 ↪\_rtld\_local.\_dl\_sysinfo\_dso' failed!

vector= [IA-64,SMP]  
 vector=percpu: enable percpu vector domain

video= [FB] Frame buffer configuration  
 See Documentation/fb/modedb.rst.

video.brightness\_switch\_enabled= [0,1]  
 If set to 1, on receiving an ACPI notify

↪event generated by hotkey, video driver will  
 ↪adjust brightness level and then send out the event to user  
 ↪space through the allocated input device; If set to 0,  
 ↪video driver will only send out the event without  
 ↪touching backlight brightness level.  
 default: 1

virtio\_mmio.device=  
 [VMMIO] Memory mapped virtio (platform)

↪device. <size>@<baseaddr>:<irq>[:<id>]  
 where:

↪standard suffixes <size> := size (can use  
 like K, M and G)  
 ↪passed to <baseaddr> := physical base address  
 <irq> := interrupt number (as  
 ↪device id request\_irq())  
 <id> := (optional) platform

example:

virtio\_mmio.device=1K@0x100b0000:48:7

↪devices. Can be used multiple times for multiple

vga= [BOOT,X86-32] Select a particular video mode  
 See Documentation/x86/boot.rst and  
 Documentation/admin-guide/svgas.rst.  
 Use vga=ask for menu.

↪the value is This is actually a boot loader parameter;  
 passed to the kernel using a special

→protocol.

vm\_debug[=options] [KNL] Available with CONFIG\_DEBUG\_VM=y.  
→VM=y.  
→when  
→memory.  
→memory  
May slow down system boot speed, especially enabled on systems with a large amount of memory. All options are enabled by default, and this interface is meant to allow for selectively enabling or disabling specific virtual debugging features.

→poisoning  
Available options are:  
P Enable page structure init time  
- Disable all of the above options

vmalloc=nn[KMG] [KNL,B00T] Forces the vmalloc area to have an exact size of <nn>. This can be used to increase the minimum size (128MB on x86). It can also be used to decrease the size and leave more room for directly mapped kernel RAM.

vmcp\_cma=nn[MG] [KNL,S390]  
→contiguous memory  
Sets the memory size reserved for allocations for the vmcp device driver.

vmhalt=  
→system halt. [KNL,S390] Perform z/VM CP command after  
Format: <command>

vmpanic=  
→kernel panic. [KNL,S390] Perform z/VM CP command after  
Format: <command>

vmloff=  
→power off. [KNL,S390] Perform z/VM CP command after  
Format: <command>

vsyscall=  
→calls to [X86-64]  
→legacy Controls the behavior of vsyscalls (i.e. fixed addresses of 0xfffffffff600x00 from

code). Most statically-linked binaries and versions of glibc use these calls. Because functions are at fixed addresses, they make targets for exploits that can control RIP.

older these nice emulate [default] Vsyscalls turn into emulated reasonably safely. The vsyscall page is readable.

traps and are xonly Vsyscalls turn into traps and emulated reasonably safely. The vsyscall page is not readable.

The vsyscall none Vsyscalls don't work at all. This makes them quite hard to use for exploits but might break your system.

vt.color= [VT] Default text color.  
Format: 0xYX, X = foreground, Y = background.  
Default: 0x07 = light gray on black.

vt.cur\_default= [VT] Default cursor shape.  
Format: 0xCCBBAA, where AA, BB, and CC are the same as the parameters of the <Esc>[?A;B;Cc escape sequence;  
see VGA-softcursor.txt. Default: 2 = underline.

vt.default\_blu= [VT]  
Format: <blue0>,<blue1>,<blue2>,...,<blue15>  
Change the default blue palette of the console.  
This is a 16-member array composed of values ranging from 0-255.

vt.default\_grn= [VT]  
Format: <green0>,<green1>,<green2>,...,<green15>  
Change the default green palette of the console.

This is a 16-member array composed of values ranging from 0-255.

`vt.default_red=` [VT]  
Format: `<red0>,<red1>,<red2>,...,<red15>`  
Change the default red palette of the `console`.

This is a 16-member array composed of values ranging from 0-255.

`vt.default_utf8=`  
[VT]  
Format=`<0|1>`  
Set system-wide default UTF-8 mode for all `tty`'s.  
Default is 1, i.e. UTF-8 mode is enabled for all newly opened terminals.

`vt.global_cursor_default=`  
[VT]  
Format=`<-1|0|1>`  
Set system-wide default for whether a cursor is shown on new VTs. Default is -1, i.e. cursors will be created by default unless overridden by individual drivers. 0 will hide cursors, 1 will display them.

`vt.italic=` [VT] Default color for italic text; 0-15.  
Default: 2 = green.

`vt.underline=` [VT] Default color for underlined text; 0-15.  
Default: 3 = cyan.

`watchdog timers` [HW,WDT] For information on watchdog timers, see `Documentation/watchdog/watchdog-parameters.rst` or other driver-specific files in the `Documentation/watchdog/` directory.

`watchdog_thresh=`  
[KNL]  
Set the hard lockup detector stall duration threshold in seconds. The soft lockup detector threshold is set to twice the value. A value of 0 disables both lockup detectors. Default is

↪10 seconds.

workqueue.watchdog\_thresh=  
 If CONFIG\_WQ\_WATCHDOG is configured, ↪  
 ↪workqueue can warn stall conditions and dump internal ↪  
 ↪state to help debugging. 0 disables workqueue stall ↪  
 ↪threshold detection; otherwise, it's the stall ↪  
 ↪30 and duration in seconds. The default value is ↪  
 ↪the it can be updated at runtime by writing to ↪  
 corresponding sysfs file.

workqueue.disable\_numa  
 By default, all work items queued to unbound ↪  
 ↪they're workqueues are affine to the NUMA nodes ↪  
 ↪in issued on, which results in better behavior ↪  
 ↪disabled for general. If NUMA affinity needs to be ↪  
 ↪Note whatever reason, this option can be used. ↪  
 ↪per-workqueue for that this also can be controlled ↪  
 ↪. workqueues visible under /sys/bus/workqueue/

workqueue.power\_efficient  
 Per-cpu workqueues are generally preferred ↪  
 ↪because they show better performance thanks to cache ↪  
 ↪tend to locality; unfortunately, per-cpu workqueues ↪  
 ↪workqueues. be more power hungry than unbound ↪

↪which Enabling this makes the per-cpu workqueues ↪  
 ↪to power were observed to contribute significantly ↪  
 ↪lower consumption unbound, leading to measurably ↪  
 power usage at the cost of small performance ↪  
 overhead.

The default value of this parameter is ↪

↳determined by the config option CONFIG\_WQ\_POWER\_EFFICIENT\_

↳DEFAULT.

workqueue.debug\_force\_rr\_cpu Workqueue used to implicitly guarantee that

↳work items queued without explicit CPU specified

↳are put on the local CPU. This guarantee is no

↳longer true and while local CPU is still preferred work

↳items may be put on foreign CPUs. This debug

↳option forces round-robin CPU selection to flush

↳out usages which depend on the now broken

↳guarantee. When enabled, memory and cache locality

↳will be impacted.

x2apic\_phys [X86-64,APIC] Use x2apic physical mode

↳instead of default x2apic cluster mode on platforms supporting x2apic.

x86\_intel\_mid\_timer= [X86-32,APBT] Choose timer option for x86 Intel MID

↳platform. Two valid options are apbt timer only and

↳lapic timer plus one apbt timer for broadcast timer.

↳apbt x86\_intel\_mid\_timer=apbt\_only | lapic\_and\_

xen\_512gb\_limit [KNL,X86-64,XEN] Restricts the kernel running

↳paravirtualized under Xen to use only up to 512 GB of RAM. The reason

↳to do so is crash analysis tools and Xen tools for

↳doing domain save/restore/migration must be enabled to

↳handle larger domains.

xen\_emul\_unplug= [HW,X86,XEN] Unplug Xen emulated devices

Format: [unplug0,][unplug1]

→ devices                    ide-disks -- unplug primary master IDE

→ IDE devices                aux-ide-disks -- unplug non-primary-master

→ and IDE disks)            nics -- unplug network devices  
                               all -- unplug all emulated devices (NICs

→ is                            unnecessary -- unplugging emulated devices

→ not respond to             unnecessary even if the host did

→ check succeeds             the unplug protocol  
                               never -- do not unplug even if version

                  xen\_legacy\_crash            [X86,XEN]  
 → executing late              Crash from Xen panic notifier, without

                                  panic() code such as dumping handler.

                  xen\_nopvspin            [X86,XEN]  
 → PV                            Disables the ticketlock slowpath using Xen

                                  optimizations.

                  xen\_nopv                [X86]  
 → HVM guest to                Disables the PV optimizations forcing the

→ option, which                run as generic HVM guest with no PV drivers.  
                                   This option is obsoleted by the "nopv"

                                  has equivalent effect for XEN platform.

                  xen\_scrub\_pages=            [XEN]  
 → before giving them back    Boolean option to control scrubbing pages

→ also changed at runtime    to Xen, for use by other domains. Can be

→ memory0/scrub\_pages.        with /sys/devices/system/xen\_memory/xen\_

→ SCRUB\_PAGES\_DEFAULT.        Default value controlled with CONFIG\_XEN\_

                  xen\_timer\_slop= [X86-64,XEN]  
 → virtual Xen                 Set the timer slop (in nanoseconds) for the

→ the minimum                 timers (default is 100000). This adjusts

→ lower values                 delta of virtualized Xen timers, where

                                  improve timer resolution at the expense of

↳processing more timer interrupts.

    nopv= [X86,XEN,KVM,HYPER\_V,VMWARE]  
Disables the PV optimizations forcing the

↳guest to run as generic guest with no PV drivers.

↳Currently support XEN HVM, KVM, HYPER\_V and VMWARE guest.

    xirc2ps\_cs= [NET,PCMCIA]  
Format:  
    <irq>,<irq\_mask>,<io>,<full\_duplex>,<do\_

↳sound>,<lockup\_hack>[,<irq2>[,<irq3>[,<irq4>]]]

    xive= [PPC]  
By default on POWER9 and above, the kernel

↳will natively use the XIVE interrupt controller.

↳This option allows the fallback firmware mode to be

↳used: off Fallback to firmware control of

↳XIVE interrupt controller on both pseries and

↳powernv platforms. Only useful on POWER9

↳and above.

    xhci-hcd.quirks [USB,KNL]  
A hex value specifying bitmask with

↳supplemental xhci host controller quirks. Meaning of each bit

↳can be consulted in header drivers/usb/host/xhci.h.

    xmon [PPC]  
Format: { early | on | rw | ro | off }  
Controls if xmon debugger is enabled.

↳Default is off. Passing only "xmon" is equivalent to

↳"xmon=early". early Call xmon as early as possible on

↳boot; xmon debugger is called from setup\_

↳arch(). on xmon debugger hooks will be

↳installed so xmon is only called on a kernel crash.

↳Default mode,

↪controlled		i.e. either "ro" or "rw" mode, is
↪installed so xmon	rw	with CONFIG_XMON_DEFAULT_RO_MODE. xmon debugger hooks will be
↪mode is write,		is called only on a kernel crash,
↪other data		meaning SPR registers, memory and,
↪registers,	ro	can be written using xmon commands. same as "rw" option above but SPR
↪written using		memory, and other data can't be
	off	xmon commands. xmon is disabled.

## 2.2 Todo

Add more DRM drivers.



## LINUX ALLOCATED DEVICES (4.X+ VERSION)

This list is the Linux Device List, the official registry of allocated device numbers and /dev directory nodes for the Linux operating system.

The LaTeX version of this document is no longer maintained, nor is the document that used to reside at lanana.org. This version in the mainline Linux kernel is the master document. Updates shall be sent as patches to the kernel maintainers (see the Documentation/process/submitting-patches.rst document). Specifically explore the sections titled “CHAR and MISC DRIVERS”, and “BLOCK LAYER” in the MAINTAINERS file to find the right maintainers to involve for character and block devices.

This document is included by reference into the Filesystem Hierarchy Standard (FHS). The FHS is available from <https://www.pathname.com/fhs/>.

Allocations marked (68k/Amiga) apply to Linux/68k on the Amiga platform only. Allocations marked (68k/Atari) apply to Linux/68k on the Atari platform only.

This document is in the public domain. The authors requests, however, that semantically altered versions are not distributed without permission of the authors, assuming the authors can be contacted without an unreasonable effort.

**Attention:** DEVICE DRIVERS AUTHORS PLEASE READ THIS

Linux now has extensive support for dynamic allocation of device numbering and can use `sysfs` and `udev` (`systemd`) to handle the naming needs. There are still some exceptions in the serial and boot device area. Before asking for a device number make sure you actually need one.

To have a major number allocated, or a minor number in situations where that applies (e.g. `busmice`), please submit a patch and send to the authors as indicated above.

Keep the description of the device in the same format as this list. The reason for this is that it is the only way we have found to ensure we have all the requisite information to publish your device and avoid conflicts.

Finally, sometimes we have to play “namespace police.” Please don’t be offended. We often get submissions for /dev names that would be bound to cause conflicts down the road. We are trying to avoid getting in a situation where we would have to suffer an incompatible forward change. Therefore, please consult with us **before** you make your device names and numbers in any way public, at least to the point where it would be at all difficult to get them changed.

Your cooperation is appreciated.

0 Unnamed devices (e.g. non-device mounts)  
0 = reserved as null device number  
See block major 144, 145, 146 for expansion areas.

1 char Memory devices

1 = /dev/mem	Physical memory access
2 = /dev/kmem	Kernel virtual memory access
3 = /dev/null	Null device
4 = /dev/port	I/O port access
5 = /dev/zero	Null byte source
6 = /dev/core	OBSOLETE - replaced by /

→proc/kcore

7 = /dev/full	Returns ENOSPC on write
8 = /dev/random	Nondeterministic random

→number gen.

9 = /dev/urandom	Faster, less secure random
------------------	----------------------------

→number gen.

10 = /dev/aio	Asynchronous I/O
---------------	------------------

→notification interface

11 = /dev/kmsg	Writes to this come out as
----------------	----------------------------

→printk's, reads

export the buffered printk

→records.

12 = /dev/oldmem	OBSOLETE - replaced by /
------------------	--------------------------

→proc/vmcore

1 block RAM disk

0 = /dev/ram0	First RAM disk
1 = /dev/ram1	Second RAM disk
...	
250 = /dev/initrd	Initial RAM disk

Older kernels had /dev/ramdisk (1, 1) here.  
/dev/initrd refers to a RAM disk which was preloaded by the boot loader; newer kernels use /dev/ram0 for the initrd.

2 char Pseudo-TTY masters

0 = /dev/ptyp0	First PTY master
1 = /dev/ptyp1	Second PTY master
...	
255 = /dev/ptyef	256th PTY master

Pseudo-tty's are named as follows:  
\* Masters are "pty", slaves are "tty";  
\* the fourth letter is one of pqrstuvwxyzabcde

→indicating

the 1st through 16th series of 16 pseudo-ttys

↪each, and  
 ↪indicating

\* the fifth letter is one of 0123456789abcdef,

the position within the series.

These are the old-style (BSD) PTY devices; Unix98 devices are on major 128 and above and use the PTY master multiplex (/dev/ptmx) to acquire a PTY on demand.

2 block	Floppy disks	
↪autodetect	0 = /dev/fd0	Controller 0, drive 0, <a href="#">↵</a>
↪autodetect	1 = /dev/fd1	Controller 0, drive 1, <a href="#">↵</a>
↪autodetect	2 = /dev/fd2	Controller 0, drive 2, <a href="#">↵</a>
↪autodetect	3 = /dev/fd3	Controller 0, drive 3, <a href="#">↵</a>
↪autodetect	128 = /dev/fd4	Controller 1, drive 0, <a href="#">↵</a>
↪autodetect	129 = /dev/fd5	Controller 1, drive 1, <a href="#">↵</a>
↪autodetect	130 = /dev/fd6	Controller 1, drive 2, <a href="#">↵</a>
↪autodetect	131 = /dev/fd7	Controller 1, drive 3, <a href="#">↵</a>

To specify format, add to the autodetect device,

↪number:	0 = /dev/fd?	Autodetect format
↪drive(1)	4 = /dev/fd?d360	5.25" 360K in a 360K <a href="#">↵</a>
↪drive(1)	20 = /dev/fd?h360	5.25" 360K in a 1200K <a href="#">↵</a>
↪drive(1)	48 = /dev/fd?h410	5.25" 410K in a 1200K drive
↪drive(1)	64 = /dev/fd?h420	5.25" 420K in a 1200K drive
↪drive(1)	24 = /dev/fd?h720	5.25" 720K in a 1200K drive
↪drive(1)	80 = /dev/fd?h880	5.25" 880K in a 1200K <a href="#">↵</a>
↪drive(1)	8 = /dev/fd?h1200	5.25" 1200K in a 1200K <a href="#">↵</a>
↪drive(1)	40 = /dev/fd?h1440	5.25" 1440K in a 1200K <a href="#">↵</a>
↪drive(1)	56 = /dev/fd?h1476	5.25" 1476K in a 1200K drive
↪drive(1)	72 = /dev/fd?h1494	5.25" 1494K in a 1200K drive
↪drive(1)	92 = /dev/fd?h1600	5.25" 1600K in a 1200K <a href="#">↵</a>
↪Density(2)	12 = /dev/fd?u360	3.5" 360K Double <a href="#">↵</a>

```

    16 = /dev/fd?u720      3.5"   720K Double
↪Density(1)
    120 = /dev/fd?u800    3.5"   800K Double
↪Density(2)
    52 = /dev/fd?u820    3.5"   820K Double Density
    68 = /dev/fd?u830    3.5"   830K Double Density
    84 = /dev/fd?u1040   3.5"  1040K Double
↪Density(1)
    88 = /dev/fd?u1120   3.5"  1120K Double
↪Density(1)
    28 = /dev/fd?u1440   3.5"  1440K High Density(1)
    124 = /dev/fd?u1600  3.5"  1600K High Density(1)
    44 = /dev/fd?u1680   3.5"  1680K High Density(3)
    60 = /dev/fd?u1722   3.5"  1722K High Density
    76 = /dev/fd?u1743   3.5"  1743K High Density
    96 = /dev/fd?u1760   3.5"  1760K High Density
    116 = /dev/fd?u1840  3.5"  1840K High Density(3)
    100 = /dev/fd?u1920  3.5"  1920K High Density(1)
    32 = /dev/fd?u2880   3.5"  2880K Extra Density(1)
    104 = /dev/fd?u3200  3.5"  3200K Extra Density
    108 = /dev/fd?u3520  3.5"  3520K Extra Density
    112 = /dev/fd?u3840  3.5"  3840K Extra Density(1)

```

```

    36 = /dev/fd?CompaQ   Compaq 2880K drive;
↪obsolete?
    (1) Autodetectable format
    (2) Autodetectable format in a Double Density
↪(720K) drive only
    (3) Autodetectable format in a High Density (1440K)
↪drive only

```

NOTE: The letter in the device name (d, q, h or u) signifies the type of drive: 5.25" Double Density

```

↪(d),
↪5"
    (any model, u). The use of the capital letters D, H and E for the 3.5" models have been deprecated,
↪since
    the drive type is insignificant for these devices.

```

```

3 char   Pseudo-TTY slaves
         0 = /dev/ttyp0      First PTY slave
         1 = /dev/ttyp1      Second PTY slave
         ...
         255 = /dev/ttyef    256th PTY slave

```

These are the old-style (BSD) PTY devices; Unix98 devices are on major 136 and above.

3 block First MFM, RLL and IDE hard disk/CD-ROM interface  
 0 = /dev/hda Master: whole disk (or ↵  
 ↵CD-ROM)  
 64 = /dev/hdb Slave: whole disk (or ↵  
 ↵CD-ROM)

For partitions, add to the whole disk device number:  
 0 = /dev/hd? Whole disk  
 1 = /dev/hd?1 First partition  
 2 = /dev/hd?2 Second partition  
 ...  
 63 = /dev/hd?63 63rd partition

For Linux/i386, partitions 1-4 are the primary partitions, and 5 and above are logical partitions. Other versions of Linux use partitioning schemes appropriate to their respective architectures.

4 char TTY devices  
 0 = /dev/tty0 Current virtual console  
 1 = /dev/tty1 First virtual console  
 ...  
 63 = /dev/tty63 63rd virtual console  
 64 = /dev/ttyS0 First UART serial port  
 ...  
 255 = /dev/ttyS191 192nd UART serial port

UART serial ports refer to 8250/16450/16550 series ↵  
 ↵devices.

Older versions of the Linux kernel used this major number for BSD PTY devices. As of Linux 2.1.115, ↵  
 ↵this  
 is no longer supported. Use major numbers 2 and 3.

4 block Aliases for dynamically allocated major devices to ↵  
 ↵be used  
 ↵nodes  
 because the root filesystem is mounted read-only.

0 = /dev/root

5 char Alternate TTY devices  
 0 = /dev/tty Current TTY device  
 1 = /dev/console System console  
 2 = /dev/ptmx PTY master multiplex  
 3 = /dev/ttyprintk User messages via printk ↵  
 ↵TTY device  
 64 = /dev/cua0 Callout device for ttyS0

...

255 = /dev/cua191            Callout device for ttyS191

(5,1) is /dev/console starting with Linux 2.1.71. [↵](#)

↪ See

the section on terminal devices for more information on /dev/console.

6 char            Parallel printer devices

    0 = /dev/lp0            Parallel printer on parport0

    1 = /dev/lp1            Parallel printer on parport1

    ...

Current Linux kernels no longer have a fixed mapping between parallel ports and I/O addresses. Instead, they are redirected through the parport multiplex, [↵](#)

↪ layer.

7 char            Virtual console capture devices

    0 = /dev/vcs            Current vc text (glyph) [↵](#)

↪ contents

    1 = /dev/vcs1           tty1 text (glyph) contents

    ...

    63 = /dev/vcs63        tty63 text (glyph) contents

    64 = /dev/vcsu         Current vc text (unicode) [↵](#)

↪ contents

    65 = /dev/vcsu1        tty1 text (unicode) contents

    ...

    127 = /dev/vcsu63      tty63 text (unicode) [↵](#)

↪ contents

    128 = /dev/vcsa        Current vc text/attribute [↵](#)

↪ (glyph) contents

    129 = /dev/vcsa1       tty1 text/attribute (glyph) [↵](#)

↪ contents

    ...

    191 = /dev/vcsa63      tty63 text/attribute [↵](#)

↪ (glyph) contents

NOTE: These devices permit both read and write [↵](#)

↪ access.

7 block           Loopback devices

    0 = /dev/loop0         First loop device

    1 = /dev/loop1         Second loop device

    ...

The loop devices are used to mount filesystems not associated with block devices. The binding to the loop devices is handled by mount(8) or losetup(8).

8 block           SCSI disk devices (0-15)

	0 = /dev/sda	First SCSI disk whole disk
	16 = /dev/sdb	Second SCSI disk whole disk
	32 = /dev/sdc	Third SCSI disk whole disk
	...	
	240 = /dev/sdp	Sixteenth SCSI disk whole disk
→disk		
	Partitions are handled in the same way as for IDE disks (see major number 3) except that the limit on partitions is 15.	
9 char	SCSI tape devices	
	0 = /dev/st0	First SCSI tape, mode 0
	1 = /dev/st1	Second SCSI tape, mode 0
	...	
	32 = /dev/st0l	First SCSI tape, mode 1
	33 = /dev/st1l	Second SCSI tape, mode 1
	...	
	64 = /dev/st0m	First SCSI tape, mode 2
	65 = /dev/st1m	Second SCSI tape, mode 2
	...	
	96 = /dev/st0a	First SCSI tape, mode 3
	97 = /dev/st1a	Second SCSI tape, mode 3
	...	
→rewind	128 = /dev/nst0	First SCSI tape, mode 0, no
→no rewind	129 = /dev/nst1	Second SCSI tape, mode 0,
	...	
→rewind	160 = /dev/nst0l	First SCSI tape, mode 1, no
→no rewind	161 = /dev/nst1l	Second SCSI tape, mode 1,
	...	
→rewind	192 = /dev/nst0m	First SCSI tape, mode 2, no
→no rewind	193 = /dev/nst1m	Second SCSI tape, mode 2,
	...	
→rewind	224 = /dev/nst0a	First SCSI tape, mode 3, no
→no rewind	225 = /dev/nst1a	Second SCSI tape, mode 3,
	...	
→MTOFFL	"No rewind" refers to the omission of the default automatic rewind on device close. The MTREW or	
→of	ioctl()'s can be used to rewind the tape regardless	
	the device used to access it.	

9 block	Metadisk (RAID) devices	
	0 = /dev/md0	First metadisk group
	1 = /dev/md1	Second metadisk group
	...	
	The metadisk driver is used to span a filesystem across multiple physical disks.	
10 char	Non-serial mice, misc features	
	0 = /dev/logibm	Logitech bus mouse
	1 = /dev/psaux	PS/2-style mouse port
	2 = /dev/inportbm	Microsoft Inport bus mouse
	3 = /dev/atibm	ATI XL bus mouse
	4 = /dev/jbm	J-mouse
	4 = /dev/amigamouse	Amiga mouse (68k/Amiga)
	5 = /dev/atarimouse	Atari mouse
	6 = /dev/sunmouse	Sun mouse
	7 = /dev/amigamouse1	Second Amiga mouse
	8 = /dev/smouse	Simple serial mouse driver
	9 = /dev/pc110pad	IBM PC-110 digitizer pad
	10 = /dev/adbmouse	Apple Desktop Bus mouse
	11 = /dev/vrtpanel	Vr41xx embedded touch panel
	13 = /dev/vpcmouse	Connectix Virtual PC Mouse
	14 = /dev/touchscreen/ucb1x00	UCB 1x00 touchscreen
	15 = /dev/touchscreen/mk712	MK712 touchscreen
	128 = /dev/beep	Fancy beep device
	129 =	
	130 = /dev/watchdog	Watchdog timer port
	131 = /dev/temperature	Machine internal temperature
	132 = /dev/hwtrap	Hardware fault trap
	133 = /dev/exttrp	External device trap
	134 = /dev/apm_bios	Advanced Power Management <a href="#">␣</a>
→BIOS	135 = /dev/rtc	Real Time Clock
	137 = /dev/vhci	Bluetooth virtual HCI driver
	139 = /dev/openprom	SPARC OpenBoot PROM
	140 = /dev/relay8	Berkshire Products Octal <a href="#">␣</a>
→relay card	141 = /dev/relay16	Berkshire Products ISO-16 <a href="#">␣</a>
→relay card	142 =	
	143 = /dev/pciconf	PCI configuration space
	144 = /dev/nvram	Non-volatile configuration <a href="#">␣</a>
→RAM	145 = /dev/hfmodem	Soundcard shortwave modem <a href="#">␣</a>
→control	146 = /dev/graphics	Linux/SGI graphics device
	147 = /dev/opengl	Linux/SGI OpenGL pipe
	148 = /dev/gfx	Linux/SGI graphics effects <a href="#">␣</a>
→device		

	149 = /dev/input/mouse	Linux/SGI Irix emulation
→mouse		
	150 = /dev/input/keyboard	Linux/SGI Irix emulation
→keyboard		
	151 = /dev/led	Front panel LEDs
	152 = /dev/kpoll	Kernel Poll Driver
	153 = /dev/mergemem	Memory merge device
	154 = /dev/pmu	Macintosh PowerBook power
→manager		
	155 = /dev/isictl	MultiTech ISICom serial
→control		
	156 = /dev/lcd	Front panel LCD display
	157 = /dev/ac	Applicom Intl Profibus card
	158 = /dev/nwbutton	Netwinder external button
	159 = /dev/nwdebug	Netwinder debug interface
	160 = /dev/nwflash	Netwinder flash memory
	161 = /dev/userdma	User-space DMA access
	162 = /dev/smbus	System Management Bus
	163 = /dev/lik	Logitech Internet Keyboard
	164 = /dev/ipmo	Intel Intelligent Platform
→Management		
	165 = /dev/vmmon	VMware virtual machine
→monitor		
	166 = /dev/i2o/ctl	I2O configuration manager
	167 = /dev/specialix_sxctl	Specialix serial control
	168 = /dev/tcldrv	Technology Concepts serial
→control		
	169 = /dev/specialix_rioctl	Specialix RIO serial
→control		
	170 = /dev/thinkpad/thinkpad	IBM Thinkpad devices
	171 = /dev/srripc	QNX4 API IPC manager
	172 = /dev/usemaclone	Semaphore clone device
	173 = /dev/ipmikcs	Intelligent Platform
→Management		
	174 = /dev/uctrl	SPARCbook 3 microcontroller
	175 = /dev/agpgart	AGP Graphics Address
→Remapping Table		
	176 = /dev/gtrsc	Gorgy Timing radio clock
	177 = /dev/cbm	Serial CBM bus
	178 = /dev/jsflash	JavaStation OS flash SIMM
	179 = /dev/xsvc	High-speed shared-mem/
→semaphore service		
	180 = /dev/vrbuttons	Vr41xx button input device
	181 = /dev/toshiba	Toshiba laptop SMM support
	182 = /dev/perfctr	Performance-monitoring
→counters		
	183 = /dev/hwrng	Generic random number
→generator		
	184 = /dev/cpu/microcode	CPU microcode update
→interface		
	186 = /dev/atomicps	Atomic snapshot of process

↪state data	187 = /dev/irnet	IrNET device
	188 = /dev/smbusbios	SMBus BIOS
	189 = /dev/ussp_ctl	User space serial port_
↪control	190 = /dev/crash	Mission Critical Linux_
↪crash dump facility	191 = /dev/pcl181	<information missing>
	192 = /dev/nas_xbus	NAS xbus LCD/buttons access
	193 = /dev/d7s	SPARC 7-segment display
	194 = /dev/zkshim	Zero-Knowledge network shim_
↪control	195 = /dev/elographics/e2201	Elographics_
↪touchscreen E271-2201	196 = /dev/vfio/vfio	VFIO userspace driver_
↪interface	197 = /dev/pxa3xx-gcu	PXA3xx graphics controller_
↪unit driver	198 = /dev/sexec	Signed executable interface
	199 = /dev/scanners/cuecat	:CueCat barcode scanner
	200 = /dev/net/tun	TAP/TUN network device
	201 = /dev/button/gulpb	Transmeta GULP-B buttons
	202 = /dev/emd/ctl	Enhanced Metadisk RAID_
↪(EMD) control	203 = /dev/cuse	Cuse (character device in_
↪user-space)	204 = /dev/video/em8300	EM8300 DVD decoder_
↪control	205 = /dev/video/em8300_mv	EM8300 DVD decoder_
↪video	206 = /dev/video/em8300_ma	EM8300 DVD decoder_
↪audio	207 = /dev/video/em8300_sp	EM8300 DVD decoder_
↪subpicture	208 = /dev/compaq/cpqhpc	Compaq PCI Hot Plug_
↪Controller	209 = /dev/compaq/cpqrid	Compaq Remote_
↪Insight Driver	210 = /dev/impi/bt	IMPI coprocessor block_
↪transfer	211 = /dev/impi/smic	IMPI coprocessor stream_
↪interface	212 = /dev/watchdogs/0	First watchdog device
	213 = /dev/watchdogs/1	Second watchdog device
	214 = /dev/watchdogs/2	Third watchdog device
	215 = /dev/watchdogs/3	Fourth watchdog device
	216 = /dev/fujitsu/apanel	Fujitsu/Siemens_
↪application panel	217 = /dev/ni/natmotn	National_
↪Instruments Motion	218 = /dev/kchuid	Inter-process chuid control

	219 = /dev/modems/mwave	MWave modem firmware upload
	220 = /dev/mptctl	Message passing technology
↳(MPT) control	221 = /dev/mvista/hssdsi	Montavista PICMG
↳hot swap system driver	222 = /dev/mvista/hasi	Montavista PICMG
↳high availability	223 = /dev/input/uinput	User level driver
↳support for input	224 = /dev/tpm	TCPA TPM driver
	225 = /dev/pps	Pulse Per Second driver
	226 = /dev/systrace	Systrace device
	227 = /dev/mcelog	X86_64 Machine Check
↳Exception driver	228 = /dev/hpet	HPET driver
	229 = /dev/fuse	Fuse (virtual filesystem in
↳user-space)	230 = /dev/midishare	MidiShare driver
	231 = /dev/snapshot	System memory snapshot
↳device	232 = /dev/kvm	Kernel-based virtual
↳machine (hardware virtualization extensions)	233 = /dev/kmview	View-OS A process with a
↳view	234 = /dev/btrfs-control	Btrfs control device
	235 = /dev/autofs	Autofs control device
	236 = /dev/mapper/control	Device-Mapper
↳control device	237 = /dev/loop-control	Loopback control device
	238 = /dev/vhost-net	Host kernel accelerator for
↳virtio net	239 = /dev/uhid	User-space I/O driver
↳support for HID subsystem	240 = /dev/userio	Serio driver testing device
	241 = /dev/vhost-vsock	Host kernel driver for
↳virtio vsock	242-254	Reserved for local use
	255	Reserved for MISC_DYNAMIC_
↳MINOR		
11 char	Raw keyboard device	(Linux/SPARC only)
	0 = /dev/kbd	Raw keyboard device
11 char	Serial Mux device	(Linux/PA-RISC only)
	0 = /dev/ttyB0	First mux port
	1 = /dev/ttyB1	Second mux port
	...	
11 block	SCSI CD-ROM devices	
	0 = /dev/scd0	First SCSI CD-ROM

1 = /dev/scd1            Second SCSI CD-ROM  
...

The prefix /dev/sr (instead of /dev/scd) has been  
→ deprecated.

12 char            QIC-02 tape  
2 = /dev/ntpqic11        QIC-11, no rewind-on-close  
3 = /dev/tpqic11        QIC-11, rewind-on-close  
4 = /dev/ntpqic24        QIC-24, no rewind-on-close  
5 = /dev/tpqic24        QIC-24, rewind-on-close  
6 = /dev/ntpqic120       QIC-120, no rewind-on-close  
7 = /dev/tpqic120       QIC-120, rewind-on-close  
8 = /dev/ntpqic150       QIC-150, no rewind-on-close  
9 = /dev/tpqic150       QIC-150, rewind-on-close

The device names specified are proposed -- if there  
are "standard" names for these devices, please let  
→ me know.

12 block

13 char            Input core  
0 = /dev/input/js0       First joystick  
1 = /dev/input/js1       Second joystick  
...  
32 = /dev/input/mouse0   First mouse  
33 = /dev/input/mouse1   Second mouse  
...  
63 = /dev/input/mice     Unified mouse  
64 = /dev/input/event0   First event queue  
65 = /dev/input/event1   Second event queue  
...

Each device type has 5 bits (32 minors).

13 block            Previously used for the XT disk (/dev/xdN)  
Deleted in kernel v3.9.

14 char            Open Sound System (OSS)  
0 = /dev/mixer           Mixer control  
1 = /dev/sequencer       Audio sequencer  
2 = /dev/midi00           First MIDI port  
3 = /dev/dsp             Digital audio  
4 = /dev/audio           Sun-compatible digital audio  
6 =  
7 = /dev/audioctl        SPARC audio control device  
8 = /dev/sequencer2      Sequencer -- alternate  
→ device  
16 = /dev/mixer1         Second soundcard mixer  
→ control

	17 = /dev/patmgr0	Sequencer patch manager
	18 = /dev/midi01	Second MIDI port
	19 = /dev/dspl	Second soundcard digital
↪ audio		
	20 = /dev/audio1	Second soundcard Sun
↪ digital audio		
	33 = /dev/patmgr1	Sequencer patch manager
	34 = /dev/midi02	Third MIDI port
	50 = /dev/midi03	Fourth MIDI port
14 block		
15 char	Joystick	
	0 = /dev/js0	First analog joystick
	1 = /dev/js1	Second analog joystick
	...	
	128 = /dev/djs0	First digital joystick
	129 = /dev/djs1	Second digital joystick
	...	
15 block	Sony CDU-31A/CDU-33A CD-ROM	
	0 = /dev/sonycd	Sony CDU-31a CD-ROM
16 char	Non-SCSI scanners	
	0 = /dev/g4500	Genius 4500 handheld scanner
16 block	GoldStar CD-ROM	
	0 = /dev/gscd	GoldStar CD-ROM
17 char	OBSOLETE (was Chase serial card)	
	0 = /dev/ttyH0	First Chase port
	1 = /dev/ttyH1	Second Chase port
	...	
17 block	Optics Storage CD-ROM	
	0 = /dev/optcd	Optics Storage CD-ROM
18 char	OBSOLETE (was Chase serial card - alternate devices)	
	0 = /dev/cuh0	Callout device for ttyH0
	1 = /dev/cuh1	Callout device for ttyH1
	...	
18 block	Sanyo CD-ROM	
	0 = /dev/sjcd	Sanyo CD-ROM
19 char	Cyclades serial card	
	0 = /dev/ttyC0	First Cyclades port
	...	
	31 = /dev/ttyC31	32nd Cyclades port
19 block	"Double" compressed disk	
	0 = /dev/double0	First compressed disk
	...	
	7 = /dev/double7	Eighth compressed disk

<code>↪disk</code>	128 = /dev/cdouble0	Mirror of first compressed
	...	
<code>↪disk</code>	135 = /dev/cdouble7	Mirror of eighth compressed
	See the Double documentation for the meaning of the mirror devices.	
20 char	Cyclades serial card - alternate devices	
	0 = /dev/cub0	Callout device for ttyC0
	...	
	31 = /dev/cub31	Callout device for ttyC31
20 block	Hitachi CD-ROM (under development)	
	0 = /dev/hitcd	Hitachi CD-ROM
21 char	Generic SCSI access	
	0 = /dev/sg0	First generic SCSI device
	1 = /dev/sg1	Second generic SCSI device
	...	
	Most distributions name these /dev/sga, /dev/sgb...; this sets an unnecessary limit of 26 SCSI devices in the system and is counter to standard Linux device-naming practice.	
21 block	Acorn MFM hard drive interface	
	0 = /dev/mfma	First MFM drive whole disk
	64 = /dev/mfmb	Second MFM drive whole disk
	This device is used on the ARM-based Acorn RiscPC. Partitions are handled the same way as for IDE disks (see major number 3).	
22 char	Digiboard serial card	
	0 = /dev/ttyD0	First Digiboard port
	1 = /dev/ttyD1	Second Digiboard port
	...	
22 block	Second IDE hard disk/CD-ROM interface	
<code>↪CD-ROM)</code>	0 = /dev/hdc	Master: whole disk (or
<code>↪CD-ROM)</code>	64 = /dev/hdd	Slave: whole disk (or
	Partitions are handled the same way as for the first interface (see major number 3).	
23 char	Digiboard serial card - alternate devices	
	0 = /dev/cud0	Callout device for ttyD0
	1 = /dev/cud1	Callout device for ttyD1

23 block	<pre> ... Mitsumi proprietary CD-ROM 0 = /dev/mcd           Mitsumi CD-ROM </pre>
24 char	<pre> Stallion serial card 0 = /dev/ttyE0         Stallion port 0 card 0 1 = /dev/ttyE1         Stallion port 1 card 0 ... 64 = /dev/ttyE64       Stallion port 0 card 1 65 = /dev/ttyE65       Stallion port 1 card 1 ... 128 = /dev/ttyE128     Stallion port 0 card 2 129 = /dev/ttyE129     Stallion port 1 card 2 ... 192 = /dev/ttyE192     Stallion port 0 card 3 193 = /dev/ttyE193     Stallion port 1 card 3 </pre>
24 block	<pre> ... Sony CDU-535 CD-ROM 0 = /dev/cdu535        Sony CDU-535 CD-ROM </pre>
25 char	<pre> Stallion serial card - alternate devices 0 = /dev/cue0          Callout device for ttyE0 1 = /dev/cue1          Callout device for ttyE1 ... 64 = /dev/cue64        Callout device for ttyE64 65 = /dev/cue65        Callout device for ttyE65 ... 128 = /dev/cue128      Callout device for ttyE128 129 = /dev/cue129      Callout device for ttyE129 ... 192 = /dev/cue192      Callout device for ttyE192 193 = /dev/cue193      Callout device for ttyE193 </pre>
25 block	<pre> ... First Matsushita (Panasonic/SoundBlaster) CD-ROM 0 = /dev/sbpcd0        Panasonic CD-ROM controller </pre>
→0 unit 0	
→0 unit 1	1 = /dev/sbpcd1        Panasonic CD-ROM controller
→0 unit 2	2 = /dev/sbpcd2        Panasonic CD-ROM controller
→0 unit 3	3 = /dev/sbpcd3        Panasonic CD-ROM controller
26 char	
26 block	<pre> ... Second Matsushita (Panasonic/SoundBlaster) CD-ROM 0 = /dev/sbpcd4        Panasonic CD-ROM controller </pre>
→1 unit 0	1 = /dev/sbpcd5        Panasonic CD-ROM controller
→1 unit 1	2 = /dev/sbpcd6        Panasonic CD-ROM controller

```
→1 unit 2          3 = /dev/sbpcd7      Panasonic CD-ROM controller,
→1 unit 3
27 char           QIC-117 tape
                  0 = /dev/qft0      Unit 0, rewind-on-close
                  1 = /dev/qft1      Unit 1, rewind-on-close
                  2 = /dev/qft2      Unit 2, rewind-on-close
                  3 = /dev/qft3      Unit 3, rewind-on-close
                  4 = /dev/nqft0     Unit 0, no rewind-on-close
                  5 = /dev/nqft1     Unit 1, no rewind-on-close
                  6 = /dev/nqft2     Unit 2, no rewind-on-close
                  7 = /dev/nqft3     Unit 3, no rewind-on-close
                  16 = /dev/zqft0    Unit 0, rewind-on-close,
→compression      17 = /dev/zqft1      Unit 1, rewind-on-close,
→compression      18 = /dev/zqft2      Unit 2, rewind-on-close,
→compression      19 = /dev/zqft3      Unit 3, rewind-on-close,
→compression      20 = /dev/nzqft0    Unit 0, no rewind-on-close,
→compression      21 = /dev/nzqft1    Unit 1, no rewind-on-close,
→compression      22 = /dev/nzqft2    Unit 2, no rewind-on-close,
→compression      23 = /dev/nzqft3    Unit 3, no rewind-on-close,
→compression      32 = /dev/rawqft0   Unit 0, rewind-on-close, no
→file marks       33 = /dev/rawqft1   Unit 1, rewind-on-close, no
→file marks       34 = /dev/rawqft2   Unit 2, rewind-on-close, no
→file marks       35 = /dev/rawqft3   Unit 3, rewind-on-close, no
→file marks       36 = /dev/nrawqft0   Unit 0, no rewind-on-close,
→no file marks    37 = /dev/nrawqft1   Unit 1, no rewind-on-close,
→no file marks    38 = /dev/nrawqft2   Unit 2, no rewind-on-close,
→no file marks    39 = /dev/nrawqft3   Unit 3, no rewind-on-close,
→no file marks
27 block          Third Matsushita (Panasonic/SoundBlaster) CD-ROM
                  0 = /dev/sbpcd8     Panasonic CD-ROM controller,
→2 unit 0         1 = /dev/sbpcd9     Panasonic CD-ROM controller,
→2 unit 1
```

	2 = /dev/sbpcd10	Panasonic CD-ROM controller
↪2 unit 2		
	3 = /dev/sbpcd11	Panasonic CD-ROM controller
↪2 unit 3		
28 char	Stallion serial card - card programming	
	0 = /dev/staliomem0	First Stallion card I/O
↪memory		
	1 = /dev/staliomem1	Second Stallion card I/O
↪memory		
	2 = /dev/staliomem2	Third Stallion card I/O
↪memory		
	3 = /dev/staliomem3	Fourth Stallion card I/O
↪memory		
28 char	Atari SLM ACSI laser printer (68k/Atari)	
	0 = /dev/slm0	First SLM laser printer
	1 = /dev/slm1	Second SLM laser printer
	...	
28 block	Fourth Matsushita (Panasonic/SoundBlaster) CD-ROM	
	0 = /dev/sbpcd12	Panasonic CD-ROM controller
↪3 unit 0		
	1 = /dev/sbpcd13	Panasonic CD-ROM controller
↪3 unit 1		
	2 = /dev/sbpcd14	Panasonic CD-ROM controller
↪3 unit 2		
	3 = /dev/sbpcd15	Panasonic CD-ROM controller
↪3 unit 3		
28 block	ACSI disk (68k/Atari)	
	0 = /dev/ada	First ACSI disk whole disk
	16 = /dev/adb	Second ACSI disk whole disk
	32 = /dev/adc	Third ACSI disk whole disk
	...	
	240 = /dev/adp	16th ACSI disk whole disk
	Partitions are handled in the same way as for IDE disks (see major number 3) except that the limit on partitions is 15, like SCSI.	
29 char	Universal frame buffer	
	0 = /dev/fb0	First frame buffer
	1 = /dev/fb1	Second frame buffer
	...	
	31 = /dev/fb31	32nd frame buffer
29 block	Aztech/Orchid/Okano/Wearnes CD-ROM	
	0 = /dev/aztcd	Aztech CD-ROM
30 char	iBCS-2 compatibility devices	
	0 = /dev/socksys	Socket access

```
1 = /dev/spx           SVR3 local X interface
32 = /dev/inet/ip     Network access
33 = /dev/inet/icmp
34 = /dev/inet/ggp
35 = /dev/inet/ipip
36 = /dev/inet/tcp
37 = /dev/inet/egp
38 = /dev/inet/pup
39 = /dev/inet/udp
40 = /dev/inet/idp
41 = /dev/inet/rawip
```

Additionally, iBCS-2 requires the following links:

```
/dev/ip -> /dev/inet/ip
/dev/icmp -> /dev/inet/icmp
/dev/ggp -> /dev/inet/ggp
/dev/ipip -> /dev/inet/ipip
/dev/tcp -> /dev/inet/tcp
/dev/egp -> /dev/inet/egp
/dev/pup -> /dev/inet/pup
/dev/udp -> /dev/inet/udp
/dev/idp -> /dev/inet/idp
/dev/rawip -> /dev/inet/rawip
/dev/inet/arp -> /dev/inet/udp
/dev/inet/rip -> /dev/inet/udp
/dev/nfsd -> /dev/socksys
/dev/X0R -> /dev/null (? apparently not required ?)
```

```
30 block      Philips LMS CM-205 CD-ROM
               0 = /dev/cm205cd      Philips LMS CM-205 CD-ROM

/dev/lmscd is an older name for this device. This
driver does not work with the CM-205MS CD-ROM.

31 char       MPU-401 MIDI
               0 = /dev/mpu401data   MPU-401 data port
               1 = /dev/mpu401stat   MPU-401 status port

31 block      ROM/flash memory card
               0 = /dev/rom0         First ROM card (rw)
               ...
               7 = /dev/rom7         Eighth ROM card (rw)
               8 = /dev/rrom0       First ROM card (ro)
               ...
               15 = /dev/rrom7      Eighth ROM card (ro)
               16 = /dev/flash0     First flash memory card (rw)
               ...
               23 = /dev/flash7     Eighth flash memory card (rw)
→ (rw)
               24 = /dev/rflash0    First flash memory card (ro)
```

...  
 31 = /dev/rflash7 Eighth flash memory card<sub>␣</sub>  
 →(ro)

The read-write (rw) devices support back-caching written data in RAM, as well as writing to flash RAM devices. The read-only devices (ro) support reading only.

32 char Specialix serial card  
 0 = /dev/ttyX0 First Specialix port  
 1 = /dev/ttyX1 Second Specialix port

...  
 32 block Philips LMS CM-206 CD-ROM  
 0 = /dev/cm206cd Philips LMS CM-206 CD-ROM

33 char Specialix serial card - alternate devices  
 0 = /dev/cux0 Callout device for ttyX0  
 1 = /dev/cux1 Callout device for ttyX1

...  
 33 block Third IDE hard disk/CD-ROM interface  
 0 = /dev/hde Master: whole disk (or<sub>␣</sub>  
 →CD-ROM)  
 64 = /dev/hdf Slave: whole disk (or<sub>␣</sub>  
 →CD-ROM)

Partitions are handled the same way as for the first interface (see major number 3).

34 char Z8530 HDLC driver  
 0 = /dev/scc0 First Z8530, first port  
 1 = /dev/scc1 First Z8530, second port  
 2 = /dev/scc2 Second Z8530, first port  
 3 = /dev/scc3 Second Z8530, second port  
 ...

In a previous version these devices were named /dev/sc1 for /dev/scc0, /dev/sc2 for /dev/scc1, and<sub>␣</sub>  
 →SO on.

34 block Fourth IDE hard disk/CD-ROM interface  
 0 = /dev/hdg Master: whole disk (or<sub>␣</sub>  
 →CD-ROM)  
 64 = /dev/hdh Slave: whole disk (or<sub>␣</sub>  
 →CD-ROM)

Partitions are handled the same way as for the first interface (see major number 3).

35 char tclmidi MIDI driver

	0 = /dev/midi0	First MIDI port, kernel
→timed		
	1 = /dev/midi1	Second MIDI port, kernel
→timed		
	2 = /dev/midi2	Third MIDI port, kernel
→timed		
	3 = /dev/midi3	Fourth MIDI port, kernel
→timed		
	64 = /dev/rmidi0	First MIDI port, untimed
	65 = /dev/rmidi1	Second MIDI port, untimed
	66 = /dev/rmidi2	Third MIDI port, untimed
	67 = /dev/rmidi3	Fourth MIDI port, untimed
	128 = /dev/smpte0	First MIDI port, SMPTE timed
	129 = /dev/smpte1	Second MIDI port, SMPTE
→timed		
	130 = /dev/smpte2	Third MIDI port, SMPTE timed
	131 = /dev/smpte3	Fourth MIDI port, SMPTE
→timed		
35 block	Slow memory ramdisk	
	0 = /dev/slram	Slow memory ramdisk
36 char	Netlink support	
	0 = /dev/route	Routing, device updates,
→kernel to user		
	1 = /dev/skip	enSKIP security cache
→control		
	3 = /dev/fwmonitor	Firewall packet copies
	16 = /dev/tap0	First Ethertap device
	...	
	31 = /dev/tap15	16th Ethertap device
36 block	OBSOLETE (was MCA ESDI hard disk)	
37 char	IDE tape	
	0 = /dev/ht0	First IDE tape
	1 = /dev/ht1	Second IDE tape
	...	
	128 = /dev/nht0	First IDE tape, no
→rewind-on-close		
	129 = /dev/nht1	Second IDE tape, no
→rewind-on-close		
	...	
	Currently, only one IDE tape drive is supported.	
37 block	Zorro II ramdisk	
	0 = /dev/z2ram	Zorro II ramdisk
38 char	Myricom PCI Myrinet board	
	0 = /dev/mlanai0	First Myrinet board

```

        1 = /dev/mlanail      Second Myrinet board
        ...

This device is used for status query, board control
and "user level packet I/O." This board is also
accessible as a standard networking "eth" device.

38 block      OBSOLETE (was Linux/AP+)

39 char      ML-16P experimental I/O board
    0 = /dev/ml16pa-a0      First card, first analog
↪channel

    1 = /dev/ml16pa-a1      First card, second analog
↪channel

        ...
    15 = /dev/ml16pa-a15    First card, 16th analog
↪channel

    16 = /dev/ml16pa-d      First card, digital lines
    17 = /dev/ml16pa-c0     First card, first counter/
↪timer

    18 = /dev/ml16pa-c1     First card, second counter/
↪timer

    19 = /dev/ml16pa-c2     First card, third counter/
↪timer

    32 = /dev/ml16pb-a0     Second card, first analog
↪channel

    33 = /dev/ml16pb-a1     Second card, second analog
↪channel

        ...
    47 = /dev/ml16pb-a15    Second card, 16th analog
↪channel

    48 = /dev/ml16pb-d      Second card, digital lines
    49 = /dev/ml16pb-c0     Second card, first counter/
↪timer

    50 = /dev/ml16pb-c1     Second card, second counter/
↪timer

    51 = /dev/ml16pb-c2     Second card, third counter/
↪timer

        ...

39 block

40 char

40 block

41 char      Yet Another Micro Monitor
    0 = /dev/yamm           Yet Another Micro Monitor

41 block

42 char      Demo/sample use

```

42 block      Demo/sample use

This number is intended for use in sample code, as well as a general "example" device number. It should never be used for a device driver that is

→being distributed; either obtain an official number or use the local/experimental range. The sudden addition

→or removal of a driver with this number should not

→cause ill effects to the system (bugs excepted.)

IN PARTICULAR, ANY DISTRIBUTION WHICH CONTAINS A DEVICE DRIVER USING MAJOR NUMBER 42 IS NONCOMPLIANT.

43 char      isdn4linux virtual modem

0 = /dev/ttyI0	First virtual modem
...	
63 = /dev/ttyI63	64th virtual modem

43 block      Network block devices

0 = /dev/nb0	First network block device
1 = /dev/nb1	Second network block device
...	

Network Block Device is somehow similar to loopback devices: If you read from it, it sends packet across network asking server for data. If you write to it,

→it sends packet telling server to write. It could be

→used to mounting filesystems over the net, swapping over the net, implementing block device in userland etc.

44 char      isdn4linux virtual modem - alternate devices

0 = /dev/cui0	Callout device for ttyI0
...	
63 = /dev/cui63	Callout device for ttyI63

44 block      Flash Translation Layer (FTL) filesystems

0 = /dev/ftla	FTL on first Memory
→Technology Device	
16 = /dev/ftlb	FTL on second Memory
→Technology Device	
32 = /dev/ftlc	FTL on third Memory
→Technology Device	
...	
240 = /dev/ftlp	FTL on 16th Memory
→Technology Device	

Partitions are handled in the same way as for IDE disks (see major number 3) except that the partition limit is 15 rather than 63 per disk (same as SCSI.)

```

45 char      isdn4linux ISDN BRI driver
              0 = /dev/isdn0          First virtual B channel raw
→data
              ...
              63 = /dev/isdn63        64th virtual B channel raw
→data
              64 = /dev/isdnctrl0     First channel control/debug
              ...
              127 = /dev/isdnctrl63   64th channel control/debug
              128 = /dev/ipp0         First SyncPPP device
              ...
              191 = /dev/ipp63        64th SyncPPP device
              255 = /dev/isdninfo     ISDN monitor interface

```

```

45 block     Parallel port IDE disk devices
              0 = /dev/pda            First parallel port IDE disk
→disk        16 = /dev/pdb            Second parallel port IDE
              32 = /dev/pdc            Third parallel port IDE disk
→disk        48 = /dev/pdd            Fourth parallel port IDE

```

Partitions are handled in the same way as for IDE disks (see major number 3) except that the partition limit is 15 rather than 63 per disk.

```

46 char      Control Rocketport serial card
              0 = /dev/ttyR0          First Rocketport port
              1 = /dev/ttyR1          Second Rocketport port
              ...
46 block     Parallel port ATAPI CD-ROM devices
→CD-ROM      0 = /dev/pcd0            First parallel port ATAPI
→CD-ROM      1 = /dev/pcd1            Second parallel port ATAPI
→CD-ROM      2 = /dev/pcd2            Third parallel port ATAPI
→CD-ROM      3 = /dev/pcd3            Fourth parallel port ATAPI
47 char      Control Rocketport serial card - alternate devices
              0 = /dev/cur0           Callout device for ttyR0
              1 = /dev/cur1           Callout device for ttyR1
              ...

```

```
47 block      Parallel port ATAPI disk devices
              0 = /dev/pf0          First parallel port ATAPI
↪disk
              1 = /dev/pf1          Second parallel port ATAPI
↪disk
              2 = /dev/pf2          Third parallel port ATAPI
↪disk
              3 = /dev/pf3          Fourth parallel port ATAPI
↪disk
```

This driver is intended for floppy disks and similar devices and hence does not support partitioning.

```
48 char      SDL RISCom serial card
              0 = /dev/ttyL0        First RISCom port
              1 = /dev/ttyL1        Second RISCom port
              ...
48 block     Mylex DAC960 PCI RAID controller; first controller
              0 = /dev/rd/c0d0      First disk, whole disk
              8 = /dev/rd/c0d1      Second disk, whole disk
              ...
              248 = /dev/rd/c0d31   32nd disk, whole disk

For partitions add:
              0 = /dev/rd/c?d?      Whole disk
              1 = /dev/rd/c?d?p1    First partition
              ...
              7 = /dev/rd/c?d?p7    Seventh partition
```

```
49 char      SDL RISCom serial card - alternate devices
              0 = /dev/cul0         Callout device for ttyL0
              1 = /dev/cul1         Callout device for ttyL1
              ...
```

```
49 block     Mylex DAC960 PCI RAID controller; second controller
              0 = /dev/rd/c1d0      First disk, whole disk
              8 = /dev/rd/c1d1      Second disk, whole disk
              ...
              248 = /dev/rd/c1d31   32nd disk, whole disk
```

Partitions are handled as for major 48.

```
50 char      Reserved for GLINT
```

```
50 block     Mylex DAC960 PCI RAID controller; third controller
              0 = /dev/rd/c2d0      First disk, whole disk
              8 = /dev/rd/c2d1      Second disk, whole disk
              ...
              248 = /dev/rd/c2d31   32nd disk, whole disk
```

```
51 char      Baycom radio modem OR Radio Tech BIM-XXX-RS232
↪radio modem
```

```

    0 = /dev/bc0           First Baycom radio modem
    1 = /dev/bc1           Second Baycom radio modem
    ...
51 block  Mylex DAC960 PCI RAID controller; fourth controller
    0 = /dev/rd/c3d0       First disk, whole disk
    8 = /dev/rd/c3d1       Second disk, whole disk
    ...
    248 = /dev/rd/c3d31    32nd disk, whole disk

Partitions are handled as for major 48.

52 char   Spellcaster DataComm/BRI ISDN card
    0 = /dev/dcbri0       First DataComm card
    1 = /dev/dcbri1       Second DataComm card
    2 = /dev/dcbri2       Third DataComm card
    3 = /dev/dcbri3       Fourth DataComm card

52 block  Mylex DAC960 PCI RAID controller; fifth controller
    0 = /dev/rd/c4d0       First disk, whole disk
    8 = /dev/rd/c4d1       Second disk, whole disk
    ...
    248 = /dev/rd/c4d31    32nd disk, whole disk

Partitions are handled as for major 48.

53 char   BDM interface for remote debugging MC683xx,
↳microcontrollers
    0 = /dev/pd_bdm0       PD BDM interface on lp0
    1 = /dev/pd_bdm1       PD BDM interface on lp1
    2 = /dev/pd_bdm2       PD BDM interface on lp2
    4 = /dev/icd_bdm0      ICD BDM interface on lp0
    5 = /dev/icd_bdm1      ICD BDM interface on lp1
    6 = /dev/icd_bdm2      ICD BDM interface on lp2

↳MC683xx
↳of a
This device is used for the interfacing to the
microcontrollers via Background Debug Mode by use
Parallel Port interface. PD is the Motorola Public
Domain Interface and ICD is the commercial interface
by P&E.

53 block  Mylex DAC960 PCI RAID controller; sixth controller
    0 = /dev/rd/c5d0       First disk, whole disk
    8 = /dev/rd/c5d1       Second disk, whole disk
    ...
    248 = /dev/rd/c5d31    32nd disk, whole disk

Partitions are handled as for major 48.

54 char   Electrocardiognosis Holter serial card

```

```
0 = /dev/holter0    First Holter port
1 = /dev/holter1    Second Holter port
2 = /dev/holter2    Third Holter port
```

A custom serial card used by Electrocardiognosis SRL  
<mseritan@ottonel.pub.ro> to transfer data from

→ Holter

24-hour heart monitoring equipment.

```
54 block    Mylex DAC960 PCI RAID controller; seventh controller
            0 = /dev/rd/c6d0    First disk, whole disk
            8 = /dev/rd/c6d1    Second disk, whole disk
            ...
            248 = /dev/rd/c6d31  32nd disk, whole disk
```

Partitions are handled as for major 48.

```
55 char     DSP56001 digital signal processor
            0 = /dev/dsp56k    First DSP56001
```

```
55 block    Mylex DAC960 PCI RAID controller; eighth controller
            0 = /dev/rd/c7d0    First disk, whole disk
            8 = /dev/rd/c7d1    Second disk, whole disk
            ...
            248 = /dev/rd/c7d31  32nd disk, whole disk
```

Partitions are handled as for major 48.

```
56 char     Apple Desktop Bus
            0 = /dev/adb        ADB bus control
```

Additional devices will be added to this number, all  
starting with /dev/adb.

```
56 block    Fifth IDE hard disk/CD-ROM interface
            0 = /dev/hdi        Master: whole disk (or
→ CD-ROM)
            64 = /dev/hdj       Slave: whole disk (or
→ CD-ROM)
```

Partitions are handled the same way as for the first  
interface (see major number 3).

```
57 char     Hayes ESP serial card
            0 = /dev/ttyP0      First ESP port
            1 = /dev/ttyP1      Second ESP port
            ...
```

```
57 block    Sixth IDE hard disk/CD-ROM interface
            0 = /dev/hdk        Master: whole disk (or
→ CD-ROM)
```

64 = /dev/hdl                    Slave: whole disk (or ↵  
↵CD-ROM)

Partitions are handled the same way as for the first interface (see major number 3).

58 char            Hayes ESP serial card - alternate devices  
                  0 = /dev/cup0            Callout device for ttyP0  
                  1 = /dev/cup1            Callout device for ttyP1  
                  ...

58 block            Reserved for logical volume manager

59 char            sf firewall package  
                  0 = /dev/firewall        Communication with sf ↵  
↵kernel module

59 block            Generic PDA filesystem device  
                  0 = /dev/pda0            First PDA device  
                  1 = /dev/pda1            Second PDA device  
                  ...

The pda devices are used to mount filesystems on remote pda's (basically slow handheld machines with proprietary OS's and limited memory and storage running small fs translation drivers) through ↵  
↵serial /  
                  IRDA / parallel links.

NAMING CONFLICT -- PROPOSED REVISED NAME /dev/rpda0 ↵  
↵etc

60-63 char        LOCAL/EXPERIMENTAL USE

60-63 block        LOCAL/EXPERIMENTAL USE  
                  Allocated for local/experimental use. For devices ↵  
↵not  
                  assigned official numbers, these ranges should be  
                  used in order to avoid conflicting with future ↵  
↵assignments.

64 char            ENskip kernel encryption package  
                  0 = /dev/enskip            Communication with ENskip ↵  
↵kernel module

64 block            Scramdisk/DriveCrypt encrypted devices  
                  0 = /dev/scramdisk/master    Master node for ↵  
↵ioctls  
                  1 = /dev/scramdisk/1        First encrypted ↵  
↵device  
                  2 = /dev/scramdisk/2        Second encrypted ↵

↪device

```
...  
255 = /dev/scramdisk/255      255th encrypted
```

↪device

The filename of the encrypted container and the

↪passwords

are sent via ioctls (using the sdmount tool) to the

↪master

node which then activates them via one of the  
/dev/scramdisk/x nodes for loop mounting (all

↪handled

through the sdmount tool).

Requested by: andy@scramdisklinux.org

65 char  
↪unused)

Sundance "plink" Transputer boards (obsolete,

```
0 = /dev/plink0      First plink device  
1 = /dev/plink1      Second plink device  
2 = /dev/plink2      Third plink device  
3 = /dev/plink3      Fourth plink device  
64 = /dev/rplink0     First plink device, raw  
65 = /dev/rplink1     Second plink device, raw  
66 = /dev/rplink2     Third plink device, raw  
67 = /dev/rplink3     Fourth plink device, raw  
128 = /dev/plink0d    First plink device, debug  
129 = /dev/plink1d    Second plink device, debug  
130 = /dev/plink2d    Third plink device, debug  
131 = /dev/plink3d    Fourth plink device, debug  
192 = /dev/rplink0d   First plink device, raw,
```

↪debug

```
193 = /dev/rplink1d   Second plink device, raw,
```

↪debug

```
194 = /dev/rplink2d   Third plink device, raw,
```

↪debug

```
195 = /dev/rplink3d   Fourth plink device, raw,
```

↪debug

This is a commercial driver; contact James Howes  
<jth@prosig.demon.co.uk> for information.

65 block

SCSI disk devices (16-31)

```
0 = /dev/sdq          17th SCSI disk whole disk  
16 = /dev/sdr         18th SCSI disk whole disk  
32 = /dev/sds         19th SCSI disk whole disk  
...  
240 = /dev/sdaf       32nd SCSI disk whole disk
```

Partitions are handled in the same way as for IDE  
disks (see major number 3) except that the limit on

partitions is 15.

66 char YARC PowerPC PCI coprocessor card  
 0 = /dev/yppcpci0 First YARC card  
 1 = /dev/yppcpci1 Second YARC card  
 ...

66 block SCSI disk devices (32-47)  
 0 = /dev/sdag 33th SCSI disk whole disk  
 16 = /dev/sdah 34th SCSI disk whole disk  
 32 = /dev/sdai 35th SCSI disk whole disk  
 ...  
 240 = /dev/sdav 48nd SCSI disk whole disk

Partitions are handled in the same way as for IDE disks (see major number 3) except that the limit on partitions is 15.

67 char Coda network file system  
 0 = /dev/cfs0 Coda cache manager

See <http://www.coda.cs.cmu.edu> for information.

→about Coda.

67 block SCSI disk devices (48-63)  
 0 = /dev/sdaw 49th SCSI disk whole disk  
 16 = /dev/sdax 50th SCSI disk whole disk  
 32 = /dev/sday 51st SCSI disk whole disk  
 ...  
 240 = /dev/sdbl 64th SCSI disk whole disk

Partitions are handled in the same way as for IDE disks (see major number 3) except that the limit on partitions is 15.

68 char CAPI 2.0 interface  
 0 = /dev/capi20 Control device  
 1 = /dev/capi20.00 First CAPI 2.0 application  
 2 = /dev/capi20.01 Second CAPI 2.0 application  
 ...  
 20 = /dev/capi20.19 19th CAPI 2.0 application

ISDN CAPI 2.0 driver for use with CAPI 2.0 applications; currently supports the AVM B1 card.

68 block SCSI disk devices (64-79)  
 0 = /dev/sdbm 65th SCSI disk whole disk  
 16 = /dev/sdbn 66th SCSI disk whole disk  
 32 = /dev/sdbo 67th SCSI disk whole disk  
 ...  
 240 = /dev/sdcb 80th SCSI disk whole disk

Partitions are handled in the same way as for IDE disks (see major number 3) except that the limit on partitions is 15.

69 char	MA16 numeric accelerator card
	0 = /dev/ma16            Board memory access
69 block	SCSI disk devices (80-95)
	0 = /dev/sdcc            81st SCSI disk whole disk
	16 = /dev/sdcd          82nd SCSI disk whole disk
	32 = /dev/sdce          83th SCSI disk whole disk
	...
	240 = /dev/sdcr          96th SCSI disk whole disk

Partitions are handled in the same way as for IDE disks (see major number 3) except that the limit on partitions is 15.

70 char	SpellCaster Protocol Services Interface
	0 = /dev/apscfg          Configuration interface
	1 = /dev/apsauth        Authentication interface
	2 = /dev/apslog        Logging interface
	3 = /dev/apsdbg        Debugging interface
	64 = /dev/apsisdn       ISDN command interface
	65 = /dev/apsasync     Async command interface
	128 = /dev/apsmon       Monitor interface

70 block	SCSI disk devices (96-111)
	0 = /dev/sdcs            97th SCSI disk whole disk
	16 = /dev/sdct          98th SCSI disk whole disk
	32 = /dev/sdcu          99th SCSI disk whole disk
	...
	240 = /dev/sddh          112nd SCSI disk whole disk

Partitions are handled in the same way as for IDE disks (see major number 3) except that the limit on partitions is 15.

71 char	Computone IntelliPort II serial card
→port 0	0 = /dev/ttyF0          IntelliPort II board 0, <small>□</small>
→port 1	1 = /dev/ttyF1          IntelliPort II board 0, <small>□</small>
	...
→port 63	63 = /dev/ttyF63        IntelliPort II board 0, <small>□</small>
→port 0	64 = /dev/ttyF64        IntelliPort II board 1, <small>□</small>
→port 1	65 = /dev/ttyF65        IntelliPort II board 1, <small>□</small>

```

    ...
    127 = /dev/ttyF127      IntelliPort II board 1, u
↪port 63
    128 = /dev/ttyF128      IntelliPort II board 2, u
↪port 0
    129 = /dev/ttyF129      IntelliPort II board 2, u
↪port 1
    ...
    191 = /dev/ttyF191      IntelliPort II board 2, u
↪port 63
    192 = /dev/ttyF192      IntelliPort II board 3, u
↪port 0
    193 = /dev/ttyF193      IntelliPort II board 3, u
↪port 1
    ...
    255 = /dev/ttyF255      IntelliPort II board 3, u
↪port 63

71 block      SCSI disk devices (112-127)
    0 = /dev/sddi          113th SCSI disk whole disk
    16 = /dev/sddj         114th SCSI disk whole disk
    32 = /dev/sddk         115th SCSI disk whole disk
    ...
    240 = /dev/sddx        128th SCSI disk whole disk

Partitions are handled in the same way as for IDE
disks (see major number 3) except that the limit on
partitions is 15.

72 char      Computone IntelliPort II serial card - alternate u
↪devices
    0 = /dev/cuf0          Callout device for ttyF0
    1 = /dev/cuf1          Callout device for ttyF1
    ...
    63 = /dev/cuf63        Callout device for ttyF63
    64 = /dev/cuf64        Callout device for ttyF64
    65 = /dev/cuf65        Callout device for ttyF65
    ...
    127 = /dev/cuf127      Callout device for ttyF127
    128 = /dev/cuf128      Callout device for ttyF128
    129 = /dev/cuf129      Callout device for ttyF129
    ...
    191 = /dev/cuf191      Callout device for ttyF191
    192 = /dev/cuf192      Callout device for ttyF192
    193 = /dev/cuf193      Callout device for ttyF193
    ...
    255 = /dev/cuf255      Callout device for ttyF255

72 block      Compaq Intelligent Drive Array, first controller
↪disk        0 = /dev/ida/c0d0      First logical drive whole u

```

↪disk                    16 = /dev/ida/c0d1            Second logical drive whole\_

                          ...  
↪disk                    240 = /dev/ida/c0d15        16th logical drive whole\_

↪on                      Partitions are handled the same way as for Mylex  
DAC960 (see major number 48) except that the limit\_  
partitions is 15.

↪devices                73 char                    Computone IntelliPort II serial card - control\_

                          0 = /dev/ip2ipl0            Loadware device for board 0  
                          1 = /dev/ip2stat0         Status device for board 0  
                          4 = /dev/ip2ipl1            Loadware device for board 1  
                          5 = /dev/ip2stat1         Status device for board 1  
                          8 = /dev/ip2ipl2            Loadware device for board 2  
                          9 = /dev/ip2stat2         Status device for board 2  
                         12 = /dev/ip2ipl3            Loadware device for board 3  
                         13 = /dev/ip2stat3         Status device for board 3

↪disk                    73 block                    Compaq Intelligent Drive Array, second controller  
                          0 = /dev/ida/c1d0            First logical drive whole\_

↪disk                    16 = /dev/ida/c1d1            Second logical drive whole\_

                          ...  
↪disk                    240 = /dev/ida/c1d15        16th logical drive whole\_

↪on                      Partitions are handled the same way as for Mylex  
DAC960 (see major number 48) except that the limit\_  
partitions is 15.

↪PCI-SCI                74 char                    SCI bridge  
                          0 = /dev/SCI/0             SCI device 0  
                          1 = /dev/SCI/1             SCI device 1  
                          ...

↪PCI-SCI                Currently for Dolphin Interconnect Solutions' \_  
bridge.

↪disk                    74 block                    Compaq Intelligent Drive Array, third controller  
                          0 = /dev/ida/c2d0            First logical drive whole\_

↪disk                    16 = /dev/ida/c2d1            Second logical drive whole\_

                          ...

240 = /dev/ida/c2d15      16th logical drive whole<sub>u</sub>  
 ↪disk

Partitions are handled the same way as for Mylex  
 DAC960 (see major number 48) except that the limit<sub>u</sub>

↪on  
 partitions is 15.

75 char      Specialix I08+ serial card  
           0 = /dev/ttyW0      First I08+ port, first card  
           1 = /dev/ttyW1      Second I08+ port, first card  
           ...  
           8 = /dev/ttyW8      First I08+ port, second card  
           ...

75 block      Compaq Intelligent Drive Array, fourth controller  
 ↪disk           0 = /dev/ida/c3d0      First logical drive whole<sub>u</sub>

↪disk           16 = /dev/ida/c3d1      Second logical drive whole<sub>u</sub>

          ...  
 ↪disk           240 = /dev/ida/c3d15      16th logical drive whole<sub>u</sub>

Partitions are handled the same way as for Mylex  
 DAC960 (see major number 48) except that the limit<sub>u</sub>

↪on  
 partitions is 15.

76 char      Specialix I08+ serial card - alternate devices  
           0 = /dev/cuw0      Callout device for ttyW0  
           1 = /dev/cuw1      Callout device for ttyW1  
           ...  
           8 = /dev/cuw8      Callout device for ttyW8  
           ...

76 block      Compaq Intelligent Drive Array, fifth controller  
 ↪disk           0 = /dev/ida/c4d0      First logical drive whole<sub>u</sub>

↪disk           16 = /dev/ida/c4d1      Second logical drive whole<sub>u</sub>

          ...  
 ↪disk           240 = /dev/ida/c4d15      16th logical drive whole<sub>u</sub>

Partitions are handled the same way as for Mylex  
 DAC960 (see major number 48) except that the limit<sub>u</sub>

↪on  
 partitions is 15.

```
77 char      ComScire Quantum Noise Generator
              0 = /dev/qng          ComScire Quantum Noise
↳Generator

77 block     Compaq Intelligent Drive Array, sixth controller
              0 = /dev/ida/c5d0     First logical drive whole
↳disk
              16 = /dev/ida/c5d1    Second logical drive whole
↳disk
              ...
              240 = /dev/ida/c5d15  16th logical drive whole
↳disk

Partitions are handled the same way as for Mylex
DAC960 (see major number 48) except that the limit
↳on
partitions is 15.

78 char      PAM Software's multimodem boards
              0 = /dev/ttyM0        First PAM modem
              1 = /dev/ttyM1        Second PAM modem
              ...

78 block     Compaq Intelligent Drive Array, seventh controller
              0 = /dev/ida/c6d0     First logical drive whole
↳disk
              16 = /dev/ida/c6d1    Second logical drive whole
↳disk
              ...
              240 = /dev/ida/c6d15  16th logical drive whole
↳disk

Partitions are handled the same way as for Mylex
DAC960 (see major number 48) except that the limit
↳on
partitions is 15.

79 char      PAM Software's multimodem boards - alternate devices
              0 = /dev/cum0         Callout device for ttyM0
              1 = /dev/cum1         Callout device for ttyM1
              ...

79 block     Compaq Intelligent Drive Array, eighth controller
              0 = /dev/ida/c7d0     First logical drive whole
↳disk
              16 = /dev/ida/c7d1    Second logical drive whole
↳disk
              ...
              240 = /dev/ida/c715   16th logical drive whole
↳disk
```

Partitions are handled the same way as for Mylex DAC960 (see major number 48) except that the limit on partitions is 15.

```

80 char      Photometrics AT200 CCD camera
             0 = /dev/at200          Photometrics AT200 CCD
→camera

80 block     I20 hard disk
             0 = /dev/i2o/hda       First I20 hard disk, whole
→disk

             16 = /dev/i2o/hdb     Second I20 hard disk, whole
→disk

             ...
             240 = /dev/i2o/hdp    16th I20 hard disk, whole
→disk

```

Partitions are handled in the same way as for IDE disks (see major number 3) except that the limit on partitions is 15.

```

81 char      video4linux
             0 = /dev/video0       Video capture/overlay device
             ...
             63 = /dev/video63    Video capture/overlay device
             64 = /dev/radio0     Radio device
             ...
             127 = /dev/radio63   Radio device
             128 = /dev/swradio0  Software Defined Radio
→device

             ...
             191 = /dev/swradio63 Software Defined Radio
→device

             224 = /dev/vbi0      Vertical blank interrupt
             ...
             255 = /dev/vbi31    Vertical blank interrupt

```

Minor numbers are allocated dynamically unless CONFIG\_VIDEO\_FIXED\_MINOR\_RANGES (default n) configuration option is set.

```

81 block     I20 hard disk
             0 = /dev/i2o/hdq      17th I20 hard disk, whole
→disk

             16 = /dev/i2o/hdr    18th I20 hard disk, whole
→disk

             ...
             240 = /dev/i2o/hdaf  32nd I20 hard disk, whole
→disk

```

Partitions are handled in the same way as for IDE disks (see major number 3) except that the limit on partitions is 15.

82 char      WinRADI0 communications receiver card  
              0 = /dev/winradio0      First WinRADI0 card  
              1 = /dev/winradio1      Second WinRADI0 card  
              ...

The driver and documentation may be obtained from <http://www.winradio.com/>

82 block      I20 hard disk  
              0 = /dev/i2o/hdag      33rd I20 hard disk, whole\_␣  
↪disk  
              16 = /dev/i2o/hdah      34th I20 hard disk, whole\_␣  
↪disk  
              ...  
              240 = /dev/i2o/hdav      48th I20 hard disk, whole\_␣  
↪disk

Partitions are handled in the same way as for IDE disks (see major number 3) except that the limit on partitions is 15.

83 char      Matrox mga\_vid video driver  
              0 = /dev/mga\_vid0      1st video card  
              1 = /dev/mga\_vid1      2nd video card  
              2 = /dev/mga\_vid2      3rd video card  
              ...  
              15 = /dev/mga\_vid15      16th video card

83 block      I20 hard disk  
              0 = /dev/i2o/hdaw      49th I20 hard disk, whole\_␣  
↪disk  
              16 = /dev/i2o/hdax      50th I20 hard disk, whole\_␣  
↪disk  
              ...  
              240 = /dev/i2o/hdbl      64th I20 hard disk, whole\_␣  
↪disk

Partitions are handled in the same way as for IDE disks (see major number 3) except that the limit on partitions is 15.

84 char      Ikon 1011[57] Versatec Greensheet Interface  
              0 = /dev/ihcp0      First Greensheet port  
              1 = /dev/ihcp1      Second Greensheet port

84 block      I20 hard disk  
              0 = /dev/i2o/hdbm      65th I20 hard disk, whole\_␣

→disk                    16 = /dev/i2o/hdbn            66th I20 hard disk, whole\_

→disk

                          ...  
240 = /dev/i2o/hdcb            80th I20 hard disk, whole\_

→disk

Partitions are handled in the same way as for IDE disks (see major number 3) except that the limit on partitions is 15.

85 char                  Linux/SGI shared memory input queue  
                          0 = /dev/shmiq                Master shared input queue  
                          1 = /dev/qcntl0                First device pushed  
                          2 = /dev/qcntl1                Second device pushed  
                          ...

85 block                I20 hard disk  
                          0 = /dev/i2o/hdcc                81st I20 hard disk, whole\_

→disk

                          16 = /dev/i2o/hdcd                82nd I20 hard disk, whole\_

→disk

                          ...  
240 = /dev/i2o/hdcr                96th I20 hard disk, whole\_

→disk

Partitions are handled in the same way as for IDE disks (see major number 3) except that the limit on partitions is 15.

86 char                  SCSI media changer  
                          0 = /dev/sch0                First SCSI media changer  
                          1 = /dev/sch1                Second SCSI media changer  
                          ...

86 block                I20 hard disk  
                          0 = /dev/i2o/hdcs                97th I20 hard disk, whole\_

→disk

                          16 = /dev/i2o/hdct                98th I20 hard disk, whole\_

→disk

                          ...  
240 = /dev/i2o/hddh                112th I20 hard disk, whole\_

→disk

Partitions are handled in the same way as for IDE disks (see major number 3) except that the limit on partitions is 15.

87 char                  Sony Control-A1 stereo control bus  
                          0 = /dev/controla0            First device on chain  
                          1 = /dev/controla1            Second device on chain

```

    ...
87 block      I20 hard disk
    0 = /dev/i2o/hddi      113rd I20 hard disk, whole
↪disk
    16 = /dev/i2o/hddj     114th I20 hard disk, whole
↪disk
    ...
    240 = /dev/i2o/hddx    128th I20 hard disk, whole
↪disk
```

Partitions are handled in the same way as for IDE disks (see major number 3) except that the limit on partitions is 15.

```

88 char      COMX synchronous serial card
    0 = /dev/comx0        COMX channel 0
    1 = /dev/comx1        COMX channel 1
    ...
```

```

88 block      Seventh IDE hard disk/CD-ROM interface
    0 = /dev/hdm          Master: whole disk (or
↪CD-ROM)
    64 = /dev/hdn         Slave: whole disk (or
↪CD-ROM)
```

Partitions are handled the same way as for the first interface (see major number 3).

```

89 char      I2C bus interface
    0 = /dev/i2c-0        First I2C adapter
    1 = /dev/i2c-1        Second I2C adapter
    ...
```

```

89 block      Eighth IDE hard disk/CD-ROM interface
    0 = /dev/hdo          Master: whole disk (or
↪CD-ROM)
    64 = /dev/hdp         Slave: whole disk (or
↪CD-ROM)
```

Partitions are handled the same way as for the first interface (see major number 3).

```

90 char      Memory Technology Device (RAM, ROM, Flash)
    0 = /dev/mtd0         First MTD (rw)
    1 = /dev/mtdr0        First MTD (ro)
    ...
    30 = /dev/mtd15       16th MTD (rw)
    31 = /dev/mtdr15      16th MTD (ro)
```

```

90 block      Ninth IDE hard disk/CD-ROM interface
```

```

    0 = /dev/hdq          Master: whole disk (or ↵
↵CD-ROM)
    64 = /dev/hdr        Slave: whole disk (or ↵
↵CD-ROM)

Partitions are handled the same way as for the first
interface (see major number 3).

91 char    CAN-Bus devices
           0 = /dev/can0      First CAN-Bus controller
           1 = /dev/can1      Second CAN-Bus controller
           ...

91 block   Tenth IDE hard disk/CD-ROM interface
↵CD-ROM)  0 = /dev/hds          Master: whole disk (or ↵
↵CD-ROM)  64 = /dev/hdt        Slave: whole disk (or ↵

Partitions are handled the same way as for the first
interface (see major number 3).

92 char    Reserved for ith Kommunikationstechnik MIC ISDN card

92 block   PPDD encrypted disk driver
           0 = /dev/ppdd0      First encrypted disk
           1 = /dev/ppdd1      Second encrypted disk
           ...

Partitions are handled in the same way as for IDE
disks (see major number 3) except that the limit on
partitions is 15.

93 char

93 block   NAND Flash Translation Layer filesystem
           0 = /dev/nftla      First NFTL layer
           16 = /dev/nftlb     Second NFTL layer
           ...
           240 = /dev/nftlp    16th NTFL layer

94 char

94 block   IBM S/390 DASD block storage
           0 = /dev/dasda First DASD device, major
           1 = /dev/dasda1 First DASD device, block 1
           2 = /dev/dasda2 First DASD device, block 2
           3 = /dev/dasda3 First DASD device, block 3
           4 = /dev/dasdb Second DASD device, major
           5 = /dev/dasdb1 Second DASD device, block 1
           6 = /dev/dasdb2 Second DASD device, block 2

```

```

    7 = /dev/dasdb3 Second DASD device, block 3
    ...

95 char      IP filter
    0 = /dev/ipl          Filter control device/log_
↪file
    1 = /dev/ipnat       NAT control device/log file
    2 = /dev/ipstate    State information log file
    3 = /dev/ipauth     Authentication control_
↪device/log file
    ...

96 char      Parallel port ATAPI tape devices
    0 = /dev/pt0        First parallel port ATAPI_
↪tape
    1 = /dev/pt1        Second parallel port ATAPI_
↪tape
    ...
    128 = /dev/npt0     First p.p. ATAPI tape, no_
↪rewind
    129 = /dev/npt1    Second p.p. ATAPI tape, no_
↪rewind
    ...

96 block     Inverse NAND Flash Translation Layer
    0 = /dev/inftla     First INFTL layer
    16 = /dev/inftlb    Second INFTL layer
    ...
    240 = /dev/inftlp   16th INTFL layer

97 char      Parallel port generic ATAPI interface
    0 = /dev/pg0        First parallel port ATAPI_
↪device
    1 = /dev/pg1        Second parallel port ATAPI_
↪device
    2 = /dev/pg2        Third parallel port ATAPI_
↪device
    3 = /dev/pg3        Fourth parallel port ATAPI_
↪device

↪SCSI        These devices support the same API as the generic_
             devices.

98 char      Control and Measurement Device (comedi)
    0 = /dev/comedi0    First comedi device
    1 = /dev/comedi1    Second comedi device
    ...
    47 = /dev/comedi47  48th comedi device

             Minors 48 to 255 are reserved for comedi subdevices_
```

↳with pathnames of the form `"/dev/comediX_subdY"`, where

↳"X" is the minor number of the associated comedi device and

↳"Y" is the subdevice number. These subdevice minors are

↳assigned dynamically, so there is no fixed mapping from

↳subdevice pathnames to minor numbers.

↳the Comedi See <http://www.comedi.org/> for information about project.

98 block User-mode virtual block device

0 = /dev/ubda	First user-mode block device
16 = /dev/udbb	Second user-mode block
...	

↳device

Partitions are handled in the same way as for IDE disks (see major number 3) except that the limit on partitions is 15.

↳port. This device is used by the user-mode virtual kernel

99 char Raw parallel ports

0 = /dev/parport0	First parallel port
1 = /dev/parport1	Second parallel port
...	

99 block JavaStation flash disk

0 = /dev/jsfd	JavaStation flash disk
---------------	------------------------

100 char Telephony for Linux

0 = /dev/phone0	First telephony device
1 = /dev/phone1	Second telephony device
...	

101 char Motorola DSP 56xxx board

0 = /dev/mdspstat	Status information
1 = /dev/mdsp1	First DSP board I/O controls
...	
16 = /dev/mdsp16	16th DSP board I/O controls

101 block AMI HyperDisk RAID controller

0 = /dev/amiraid/ar0	First array whole disk
16 = /dev/amiraid/ar1	Second array whole disk
...	

240 = /dev/amiraid/ar15 16th array whole disk

For each device, partitions are added as:

0 = /dev/amiraid/ar? Whole disk  
1 = /dev/amiraid/ar?p1 First partition  
2 = /dev/amiraid/ar?p2 Second partition  
...  
15 = /dev/amiraid/ar?p15 15th partition

102 char

102 block Compressed block device

0 = /dev/cbd/a First compressed block  
↪device, whole device  
16 = /dev/cbd/b Second compressed block  
↪device, whole device  
...  
240 = /dev/cbd/p 16th compressed block  
↪device, whole device

Partitions are handled in the same way as for IDE disks (see major number 3) except that the limit on partitions is 15.

103 char Arla network file system

0 = /dev/nnpfs0 First NNPFS device  
1 = /dev/nnpfs1 Second NNPFS device

Arla is a free clone of the Andrew File System, AFS. The NNPFS device gives user mode filesystem implementations a kernel presence for caching and

↪easy

mounting. For more information about the project, write to <arla-drinkers@stacken.kth.se> or see <http://www.stacken.kth.se/project/arla/>

103 block Audit device

0 = /dev/audit Audit device

104 char Flash BIOS support

104 block Compaq Next Generation Drive Array, first controller

0 = /dev/cciss/c0d0 First logical drive, whole  
↪disk  
16 = /dev/cciss/c0d1 Second logical drive, whole  
↪disk  
...  
240 = /dev/cciss/c0d15 16th logical drive, whole  
↪disk

Partitions are handled the same way as for Mylex

↪on	DAC960 (see major number 48) except that the limit partitions is 15.
105 char	Control VS-1000 serial controller
	0 = /dev/ttyV0            First VS-1000 port
	1 = /dev/ttyV1            Second VS-1000 port
	...
105 block	Compaq Next Generation Drive Array, second
↪controller	
	0 = /dev/cciss/cld0    First logical drive, whole
↪disk	
	16 = /dev/cciss/cld1    Second logical drive, whole
↪disk	
	...
	240 = /dev/cciss/cld15 16th logical drive, whole
↪disk	
	Partitions are handled the same way as for Mylex DAC960 (see major number 48) except that the limit partitions is 15.
↪on	
106 char	Control VS-1000 serial controller - alternate
↪devices	
	0 = /dev/cuv0            First VS-1000 port
	1 = /dev/cuv1            Second VS-1000 port
	...
106 block	Compaq Next Generation Drive Array, third controller
↪disk	
	0 = /dev/cciss/c2d0    First logical drive, whole
↪disk	
	16 = /dev/cciss/c2d1    Second logical drive, whole
↪disk	
	...
	240 = /dev/cciss/c2d15 16th logical drive, whole
↪disk	
	Partitions are handled the same way as for Mylex DAC960 (see major number 48) except that the limit partitions is 15.
↪on	
107 char	3Dfx Voodoo Graphics device
	0 = /dev/3dfx            Primary 3Dfx graphics device
107 block	Compaq Next Generation Drive Array, fourth
↪controller	
	0 = /dev/cciss/c3d0    First logical drive, whole
↪disk	

16 = /dev/cciss/c3d1 Second logical drive, whole  
→disk

...  
240 = /dev/cciss/c3d15 16th logical drive, whole  
→disk

Partitions are handled the same way as for Mylex  
DAC960 (see major number 48) except that the limit  
→on partitions is 15.

108 char Device independent PPP interface  
0 = /dev/ppp Device independent PPP  
→interface

108 block Compaq Next Generation Drive Array, fifth controller  
0 = /dev/cciss/c4d0 First logical drive, whole  
→disk

16 = /dev/cciss/c4d1 Second logical drive, whole  
→disk

...  
240 = /dev/cciss/c4d15 16th logical drive, whole  
→disk

Partitions are handled the same way as for Mylex  
DAC960 (see major number 48) except that the limit  
→on partitions is 15.

109 char Reserved for logical volume manager

109 block Compaq Next Generation Drive Array, sixth controller  
0 = /dev/cciss/c5d0 First logical drive, whole  
→disk

16 = /dev/cciss/c5d1 Second logical drive, whole  
→disk

...  
240 = /dev/cciss/c5d15 16th logical drive, whole  
→disk

Partitions are handled the same way as for Mylex  
DAC960 (see major number 48) except that the limit  
→on partitions is 15.

110 char miroMEDIA Surround board  
0 = /dev/srnd0 First miroMEDIA Surround  
→board

1 = /dev/srnd1 Second miroMEDIA Surround  
→board

...

```

110 block      Compaq Next Generation Drive Array, seventh
↪controller
           0 = /dev/cciss/c6d0   First logical drive, whole
↪disk
           16 = /dev/cciss/c6d1  Second logical drive, whole
↪disk
           ...
           240 = /dev/cciss/c6d15 16th logical drive, whole
↪disk

Partitions are handled the same way as for Mylex
DAC960 (see major number 48) except that the limit
↪on
partitions is 15.

111 char

111 block      Compaq Next Generation Drive Array, eighth
↪controller
           0 = /dev/cciss/c7d0   First logical drive, whole
↪disk
           16 = /dev/cciss/c7d1  Second logical drive, whole
↪disk
           ...
           240 = /dev/cciss/c7d15 16th logical drive, whole
↪disk

Partitions are handled the same way as for Mylex
DAC960 (see major number 48) except that the limit
↪on
partitions is 15.

112 char      ISI serial card
           0 = /dev/ttyM0       First ISI port
           1 = /dev/ttyM1       Second ISI port
           ...

There is currently a device-naming conflict between
these and PAM multimodems (major 78).

112 block      IBM iSeries virtual disk
↪disk
           0 = /dev/iseriess/vda First virtual disk, whole
↪disk
           8 = /dev/iseriess/vdb Second virtual disk, whole
↪disk
           ...
           200 = /dev/iseriess/vdz 26th virtual disk, whole
↪disk
           208 = /dev/iseriess/vdaa 27th virtual disk, whole
↪disk

```

```

    ...
    248 = /dev/iseries/vdaf 32nd virtual disk, whole_
↪disk

Partitions are handled in the same way as for IDE
disks (see major number 3) except that the limit on
partitions is 7.

113 char    ISI serial card - alternate devices
            0 = /dev/cum0          Callout device for ttyM0
            1 = /dev/cum1          Callout device for ttyM1
            ...

113 block   IBM iSeries virtual CD-ROM
            0 = /dev/iseries/vcda First virtual CD-ROM
            1 = /dev/iseries/vcdb Second virtual CD-ROM
            ...

114 char    Picture Elements ISE board
            0 = /dev/ise0          First ISE board
            1 = /dev/ise1          Second ISE board
            ...
↪board     128 = /dev/isex0          Control node for first ISE_
↪board     129 = /dev/isex1          Control node for second ISE_
            ...

↪general   The ISE board is an embedded computer, optimized for
↪command   image processing. The /dev/iseN nodes are the
            I/O access to the board, the /dev/isex0 nodes_
            nodes used to control the board.

114 block   IDE BIOS powered software RAID interfaces such as_
↪the       Promise Fastrak

            0 = /dev/ataraid/d0
            1 = /dev/ataraid/d0p1
            2 = /dev/ataraid/d0p2
            ...
            16 = /dev/ataraid/d1
            17 = /dev/ataraid/d1p1
            18 = /dev/ataraid/d1p2
            ...
            255 = /dev/ataraid/d15p15

Partitions are handled in the same way as for IDE
disks (see major number 3) except that the limit on
```

partitions is 15.

115 char TI link cable devices (115 was formerly the console,  
↳driver speaker)

0 = /dev/tipar0 Parallel cable on first,  
↳parallel port

...  
7 = /dev/tipar7 Parallel cable on seventh,  
↳parallel port

8 = /dev/tiser0 Serial cable on first serial,  
↳port

...  
15 = /dev/tiser7 Serial cable on seventh serial,  
↳port

16 = /dev/tiusb0 First USB cable

...  
47 = /dev/tiusb31 32nd USB cable

115 block NetWare (NWFS) Devices (0-255)

The NWFS (NetWare) devices are used to present a collection of NetWare Mirror Groups or NetWare Partitions as a logical storage segment for use in mounting NetWare volumes. A maximum of 256 NetWare volumes can be supported in a single machine.

↳people/jmerkey/nwfs/  
<http://cgfa.telepac.pt/ftp2/kernel.org/linux/kernel/>

0 = /dev/nwfs/v0 First NetWare (NWFS) Logical,  
↳Volume

1 = /dev/nwfs/v1 Second NetWare (NWFS) Logical,  
↳Volume

2 = /dev/nwfs/v2 Third NetWare (NWFS) Logical,  
↳Volume

...  
255 = /dev/nwfs/v255 Last NetWare (NWFS),  
↳Logical Volume

116 char Advanced Linux Sound Driver (ALSA)

116 block MicroMemory battery backed RAM adapter (NVRAM)  
Supports 16 boards, 15 partitions each.  
Requested by neilb at cse.unsw.edu.au.

0 = /dev/umem/d0 Whole of first board  
1 = /dev/umem/d0p1 First partition of first,  
↳board

↪board            2 = /dev/umem/d0p2        Second partition of first\_

                  15 = /dev/umem/d0p15     15th partition of first board

↪board            16 = /dev/umem/d1         Whole of second board

                  17 = /dev/umem/d1p1       First partition of second\_

                  ...

                  255= /dev/umem/d15p15    15th partition of 16th board.

117 char           COSA/SRP synchronous serial card

                  0 = /dev/cosa0c0          1st board, 1st channel

                  1 = /dev/cosa0c1          1st board, 2nd channel

                  ...

                  16 = /dev/cosalc0         2nd board, 1st channel

                  17 = /dev/cosalc1         2nd board, 2nd channel

                  ...

117 block           Enterprise Volume Management System (EVMS)

↪provide           The EVMS driver uses a layered, plug-in model to\_

↪managing          unparalleled flexibility and extensibility in\_

↪customization     storage. This allows for easy expansion or\_

↪by                  of various levels of volume management. Requested\_

                  Mark Peloquin (peloquin at us.ibm.com).

                  Note: EVMS populates and manages all the devnodes in /dev/evms.

<http://sf.net/projects/evms>

↪device            0 = /dev/evms/block\_device    EVMS block device

                  1 = /dev/evms/legacyname1     First EVMS legacy\_

↪device            2 = /dev/evms/legacyname2     Second EVMS legacy\_

                  ...

↪meet.             Both ranges can grow (down or up) until they\_

                  ...

↪device            254 = /dev/evms/EVMSname2     Second EVMS native\_

↪device            255 = /dev/evms/EVMSname1     First EVMS native\_

↪legacy            Note: legacyname(s) are derived from the normal\_

- device names. For example, /dev/hda5 would become /dev/evms/hda5.
- 118 char IBM Cryptographic Accelerator  
 ↪ Accelerators 0 = /dev/ica Virtual interface to all IBM Crypto  
 1 = /dev/ica0 IBMCA Device 0  
 2 = /dev/ica1 IBMCA Device 1  
 ...
- 119 char VMware virtual network control  
 0 = /dev/vnet0 1st virtual network  
 1 = /dev/vnet1 2nd virtual network  
 ...
- 120-127 char LOCAL/EXPERIMENTAL USE
- 120-127 block LOCAL/EXPERIMENTAL USE  
 ↪ not Allocated for local/experimental use. For devices  
 ↪ assignments. assigned official numbers, these ranges should be used in order to avoid conflicting with future
- 128-135 char Unix98 PTY masters
- These devices should not have corresponding device nodes; instead they should be accessed through the /dev/ptmx cloning interface.
- 128 block SCSI disk devices (128-143)  
 0 = /dev/sddy 129th SCSI disk whole disk  
 16 = /dev/sddz 130th SCSI disk whole disk  
 32 = /dev/sdea 131th SCSI disk whole disk  
 ...  
 240 = /dev/sden 144th SCSI disk whole disk
- Partitions are handled in the same way as for IDE disks (see major number 3) except that the limit on partitions is 15.
- 129 block SCSI disk devices (144-159)  
 0 = /dev/sdeo 145th SCSI disk whole disk  
 16 = /dev/sdep 146th SCSI disk whole disk  
 32 = /dev/sdeq 147th SCSI disk whole disk  
 ...  
 240 = /dev/sdfd 160th SCSI disk whole disk
- Partitions are handled in the same way as for IDE disks (see major number 3) except that the limit on partitions is 15.

- 130 char (Misc devices)
- 130 block SCSI disk devices (160-175)
- |                 |                            |
|-----------------|----------------------------|
| 0 = /dev/sdfe   | 161st SCSI disk whole disk |
| 16 = /dev/sdff  | 162nd SCSI disk whole disk |
| 32 = /dev/sdfg  | 163rd SCSI disk whole disk |
| ...             |                            |
| 240 = /dev/sdft | 176th SCSI disk whole disk |
- Partitions are handled in the same way as for IDE disks (see major number 3) except that the limit on partitions is 15.
- 131 block SCSI disk devices (176-191)
- |                 |                            |
|-----------------|----------------------------|
| 0 = /dev/sdfu   | 177th SCSI disk whole disk |
| 16 = /dev/sdfv  | 178th SCSI disk whole disk |
| 32 = /dev/sdfw  | 179th SCSI disk whole disk |
| ...             |                            |
| 240 = /dev/sdgj | 192nd SCSI disk whole disk |
- Partitions are handled in the same way as for IDE disks (see major number 3) except that the limit on partitions is 15.
- 132 block SCSI disk devices (192-207)
- |                 |                            |
|-----------------|----------------------------|
| 0 = /dev/sdgm   | 193rd SCSI disk whole disk |
| 16 = /dev/sdgn  | 194th SCSI disk whole disk |
| 32 = /dev/sdgo  | 195th SCSI disk whole disk |
| ...             |                            |
| 240 = /dev/sdgz | 208th SCSI disk whole disk |
- Partitions are handled in the same way as for IDE disks (see major number 3) except that the limit on partitions is 15.
- 133 block SCSI disk devices (208-223)
- |                 |                            |
|-----------------|----------------------------|
| 0 = /dev/sdha   | 209th SCSI disk whole disk |
| 16 = /dev/sdhb  | 210th SCSI disk whole disk |
| 32 = /dev/sdhc  | 211th SCSI disk whole disk |
| ...             |                            |
| 240 = /dev/sdhp | 224th SCSI disk whole disk |
- Partitions are handled in the same way as for IDE disks (see major number 3) except that the limit on partitions is 15.
- 134 block SCSI disk devices (224-239)
- |                |                            |
|----------------|----------------------------|
| 0 = /dev/sdhq  | 225th SCSI disk whole disk |
| 16 = /dev/sdhr | 226th SCSI disk whole disk |
| 32 = /dev/sdhs | 227th SCSI disk whole disk |

```

...
240 = /dev/sdif          240th SCSI disk whole disk

```

Partitions are handled in the same way as for IDE disks (see major number 3) except that the limit on partitions is 15.

```

135 block      SCSI disk devices (240-255)
                0 = /dev/sdig          241st SCSI disk whole disk
                16 = /dev/sdih         242nd SCSI disk whole disk
                32 = /dev/sdih         243rd SCSI disk whole disk
                ...
                240 = /dev/sdiv        256th SCSI disk whole disk

```

Partitions are handled in the same way as for IDE disks (see major number 3) except that the limit on partitions is 15.

```

136-143 char   Unix98 PTY slaves
                0 = /dev/pts/0         First Unix98 pseudo-TTY
                1 = /dev/pts/1         Second Unix98 pseudo-TTY
                ...

```

These device nodes are automatically generated with the proper permissions and modes by mounting the devpts filesystem onto /dev/pts with the appropriate mount options (distribution dependent, however, on \*most\* distributions the appropriate options are "mode=0620,gid=<gid of the "tty" group>".)

```

136 block      Mylex DAC960 PCI RAID controller; ninth controller
                0 = /dev/rd/c8d0        First disk, whole disk
                8 = /dev/rd/c8d1        Second disk, whole disk
                ...
                248 = /dev/rd/c8d31     32nd disk, whole disk

```

Partitions are handled as for major 48.

```

137 block      Mylex DAC960 PCI RAID controller; tenth controller
                0 = /dev/rd/c9d0        First disk, whole disk
                8 = /dev/rd/c9d1        Second disk, whole disk
                ...
                248 = /dev/rd/c9d31     32nd disk, whole disk

```

Partitions are handled as for major 48.

```

138 block      Mylex DAC960 PCI RAID controller; eleventh
↳controller
                0 = /dev/rd/c10d0       First disk, whole disk
                8 = /dev/rd/c10d1       Second disk, whole disk
                ...

```

```
248 = /dev/rd/c10d31    32nd disk, whole disk

Partitions are handled as for major 48.

139 block      Mylex DAC960 PCI RAID controller; twelfth controller
    0 = /dev/rd/c11d0    First disk, whole disk
    8 = /dev/rd/c11d1    Second disk, whole disk
    ...
248 = /dev/rd/c11d31    32nd disk, whole disk

Partitions are handled as for major 48.

140 block      Mylex DAC960 PCI RAID controller; thirteenth_
→controller
    0 = /dev/rd/c12d0    First disk, whole disk
    8 = /dev/rd/c12d1    Second disk, whole disk
    ...
248 = /dev/rd/c12d31    32nd disk, whole disk

Partitions are handled as for major 48.

141 block      Mylex DAC960 PCI RAID controller; fourteenth_
→controller
    0 = /dev/rd/c13d0    First disk, whole disk
    8 = /dev/rd/c13d1    Second disk, whole disk
    ...
248 = /dev/rd/c13d31    32nd disk, whole disk

Partitions are handled as for major 48.

142 block      Mylex DAC960 PCI RAID controller; fifteenth_
→controller
    0 = /dev/rd/c14d0    First disk, whole disk
    8 = /dev/rd/c14d1    Second disk, whole disk
    ...
248 = /dev/rd/c14d31    32nd disk, whole disk

Partitions are handled as for major 48.

143 block      Mylex DAC960 PCI RAID controller; sixteenth_
→controller
    0 = /dev/rd/c15d0    First disk, whole disk
    8 = /dev/rd/c15d1    Second disk, whole disk
    ...
248 = /dev/rd/c15d31    32nd disk, whole disk

Partitions are handled as for major 48.

144 char      Encapsulated PPP
    0 = /dev/pppox0      First PPP over Ethernet
    ...
```

	63 = /dev/pppox63	64th PPP over Ethernet
	This is primarily used for ADSL.	
	The SST 5136-DN DeviceNet interface driver has been relocated to major 183 due to an unfortunate	
→conflict.		
144 block →mounts	Expansion Area #1 for more non-device (e.g. NFS)	
	0 = mounted device	256
	255 = mounted device	511
145 char	SAM9407-based soundcard	
	0 = /dev/sam0_mixer	
	1 = /dev/sam0_sequencer	
	2 = /dev/sam0_midi00	
	3 = /dev/sam0_dsp	
	4 = /dev/sam0_audio	
	6 = /dev/sam0_sndstat	
	18 = /dev/sam0_midi01	
	34 = /dev/sam0_midi02	
	50 = /dev/sam0_midi03	
	64 = /dev/sam1_mixer	
	...	
	128 = /dev/sam2_mixer	
	...	
	192 = /dev/sam3_mixer	
	...	
	Device functions match OSS, but offer a number of addons, which are sam9407 specific. OSS can be operated simultaneously, taking care of the codec.	
145 block →mounts	Expansion Area #2 for more non-device (e.g. NFS)	
	0 = mounted device	512
	255 = mounted device	767
146 char	SYSTRAM SCRAMNet mirrored-memory network	
	0 = /dev/scramnet0	First SCRAMNet device
	1 = /dev/scramnet1	Second SCRAMNet device
	...	
146 block →mounts	Expansion Area #3 for more non-device (e.g. NFS)	
	0 = mounted device	768
	255 = mounted device	1023
147 char	Aureal Semiconductor Vortex Audio device	
	0 = /dev/aureal0	First Aureal Vortex

	1 = /dev/aureal1	Second Aureal Vortex
	...	
147 block	Distributed Replicated Block Device (DRBD)	
	0 = /dev/drbd0	First DRBD device
	1 = /dev/drbd1	Second DRBD device
	...	
148 char	Technology Concepts serial card	
	0 = /dev/ttyT0	First TCL port
	1 = /dev/ttyT1	Second TCL port
	...	
149 char	Technology Concepts serial card - alternate devices	
	0 = /dev/cut0	Callout device for ttyT0
	1 = /dev/cut1	Callout device for ttyT1
	...	
150 char	Real-Time Linux FIFOs	
	0 = /dev/rtf0	First RTLinux FIFO
	1 = /dev/rtf1	Second RTLinux FIFO
	...	
151 char	DPT I20 SmartRaid V controller	
	0 = /dev/dpti0	First DPT I20 adapter
	1 = /dev/dpti1	Second DPT I20 adapter
	...	
152 char	EtherDrive Control Device	
↳EtherDrive	0 = /dev/etherd/ctl	Connect/Disconnect an
	1 = /dev/etherd/err	Monitor errors
	2 = /dev/etherd/raw	Raw AoE packet monitor
152 block	EtherDrive Block Devices	
	0 = /dev/etherd/0	EtherDrive 0
	...	
	255 = /dev/etherd/255	EtherDrive 255
153 char	SPI Bus Interface (sometimes referred to as	
↳MicroWire)		
	0 = /dev/spi0	First SPI device on the bus
	1 = /dev/spi1	Second SPI device on the bus
	...	
↳bus	15 = /dev/spi15	Sixteenth SPI device on the
153 block	Enhanced Metadisk RAID (EMD) storage units	
	0 = /dev/emd/0	First unit
	1 = /dev/emd/0p1	Partition 1 on First unit
	2 = /dev/emd/0p2	Partition 2 on First unit

```

    ...
    15 = /dev/emd/0p15      Partition 15 on First unit
    16 = /dev/emd/1       Second unit
    32 = /dev/emd/2       Third unit
    ...
    240 = /dev/emd/15     Sixteenth unit

```

Partitions are handled in the same way as for IDE disks (see major number 3) except that the limit on partitions is 15.

```

154 char      Specialix RIO serial card
               0 = /dev/ttySR0      First RIO port
               ...
               255 = /dev/ttySR255  256th RIO port

155 char      Specialix RIO serial card - alternate devices
               0 = /dev/cusr0      Callout device for ttySR0
               ...
               255 = /dev/cusr255  Callout device for ttySR255

156 char      Specialix RIO serial card
               0 = /dev/ttySR256   257th RIO port
               ...
               255 = /dev/ttySR511 512th RIO port

157 char      Specialix RIO serial card - alternate devices
               0 = /dev/cusr256   Callout device for ttySR256
               ...
               255 = /dev/cusr511  Callout device for ttySR511

158 char      Dialogic GammaLink fax driver
               0 = /dev/gfax0     GammaLink channel 0
               1 = /dev/gfax1     GammaLink channel 1
               ...

159 char      RESERVED

159 block     RESERVED

160 char      General Purpose Instrument Bus (GPIB)
               0 = /dev/gpib0     First GPIB bus
               1 = /dev/gpib1     Second GPIB bus
               ...

160 block     Carmel 8-port SATA Disks on First Controller
               0 = /dev/carmel/0   SATA disk 0 whole disk
               1 = /dev/carmel/0p1 SATA disk 0 partition 1
               ...
               31 = /dev/carmel/0p31 SATA disk 0 partition 31

```

```
32 = /dev/carmel/1      SATA disk 1 whole disk
64 = /dev/carmel/2      SATA disk 2 whole disk
...
224 = /dev/carmel/7     SATA disk 7 whole disk
```

Partitions are handled in the same way as for IDE disks (see major number 3) except that the limit on partitions is 31.

```
161 char  IrCOMM devices (IrDA serial/parallel emulation)
          0 = /dev/ircomm0      First IrCOMM device
          1 = /dev/ircomm1      Second IrCOMM device
          ...
          16 = /dev/irlpt0      First IrLPT device
          17 = /dev/irlpt1      Second IrLPT device
          ...
```

```
161 block Carmel 8-port SATA Disks on Second Controller
          0 = /dev/carmel/8     SATA disk 8 whole disk
          1 = /dev/carmel/8p1   SATA disk 8 partition 1
          ...
          31 = /dev/carmel/8p31 SATA disk 8 partition 31
          ...
          32 = /dev/carmel/9     SATA disk 9 whole disk
          64 = /dev/carmel/10    SATA disk 10 whole disk
          ...
          224 = /dev/carmel/15   SATA disk 15 whole disk
```

Partitions are handled in the same way as for IDE disks (see major number 3) except that the limit on partitions is 31.

```
162 char  Raw block device interface
          0 = /dev/rawctl       Raw I/O control device
          1 = /dev/raw/raw1     First raw I/O device
          2 = /dev/raw/raw2     Second raw I/O device
          ...
```

max minor number of raw device is set by kernel\_

→config

MAX\_RAW\_DEVS or raw module parameter 'max\_raw\_devs'

```
163 char
```

```
164 char
```

```
Chase Research AT/PCI-Fast serial card
          0 = /dev/ttyCH0       AT/PCI-Fast board 0, port 0
          ...
          15 = /dev/ttyCH15     AT/PCI-Fast board 0, port 15
          16 = /dev/ttyCH16     AT/PCI-Fast board 1, port 0
          ...
          31 = /dev/ttyCH31     AT/PCI-Fast board 1, port 15
```

```

    32 = /dev/ttyCH32      AT/PCI-Fast board 2, port 0
    ...
    47 = /dev/ttyCH47      AT/PCI-Fast board 2, port 15
    48 = /dev/ttyCH48      AT/PCI-Fast board 3, port 0
    ...
    63 = /dev/ttyCH63      AT/PCI-Fast board 3, port 15

165 char devices      Chase Research AT/PCI-Fast serial card - alternate
    0 = /dev/cuch0        Callout device for ttyCH0
    ...
    63 = /dev/cuch63      Callout device for ttyCH63

166 char              ACM USB modems
    0 = /dev/ttyACM0      First ACM modem
    1 = /dev/ttyACM1      Second ACM modem
    ...

167 char              ACM USB modems - alternate devices
    0 = /dev/cuacm0      Callout device for ttyACM0
    1 = /dev/cuacm1      Callout device for ttyACM1
    ...

168 char              Eracom CSA7000 PCI encryption adaptor
    0 = /dev/ecsa0        First CSA7000
    1 = /dev/ecsa1        Second CSA7000
    ...

169 char              Eracom CSA8000 PCI encryption adaptor
    0 = /dev/ecsa8-0      First CSA8000
    1 = /dev/ecsa8-1      Second CSA8000
    ...

170 char              AMI MegaRAC remote access controller
    0 = /dev/megarac0     First MegaRAC card
    1 = /dev/megarac1     Second MegaRAC card
    ...

171 char              Reserved for IEEE 1394 (Firewire)

172 char              Moxa Intellio serial card
    0 = /dev/ttyMX0       First Moxa port
    1 = /dev/ttyMX1       Second Moxa port
    ...
    127 = /dev/ttyMX127   128th Moxa port
    128 = /dev/moxactl    Moxa control port

173 char              Moxa Intellio serial card - alternate devices
    0 = /dev/cumx0        Callout device for ttyMX0
    1 = /dev/cumx1        Callout device for ttyMX1
    ...

```

```

127 = /dev/cumx127          Callout device for ttyMX127

174 char                  SmartIO serial card
    0 = /dev/ttySI0        First SmartIO port
    1 = /dev/ttySI1        Second SmartIO port
    ...

175 char                  SmartIO serial card - alternate devices
    0 = /dev/cusi0         Callout device for ttySI0
    1 = /dev/cusi1         Callout device for ttySI1
    ...

176 char                  nCipher nFast PCI crypto accelerator
    0 = /dev/nfastpci0     First nFast PCI device
    1 = /dev/nfastpci1     First nFast PCI device
    ...

177 char                  TI PCILynx memory spaces
→card                    0 = /dev/pcilynx/aux0  AUX space of first PCILynx
    ...
→card                    15 = /dev/pcilynx/aux15  AUX space of 16th PCILynx
→card                    16 = /dev/pcilynx/rom0  ROM space of first PCILynx
    ...
→card                    31 = /dev/pcilynx/rom15  ROM space of 16th PCILynx
→card                    32 = /dev/pcilynx/ram0  RAM space of first PCILynx
    ...
→card                    47 = /dev/pcilynx/ram15  RAM space of 16th PCILynx

178 char                  Giganet cLAN1xxx virtual interface adapter
    0 = /dev/clanvi0        First cLAN adapter
    1 = /dev/clanvi1        Second cLAN adapter
    ...

179 block                 MMC block devices
→MMC card                0 = /dev/mmcblk0        First SD/MMC card
    1 = /dev/mmcblk0p1      First partition on first
    8 = /dev/mmcblk1        Second SD/MMC card
    ...

→modprobe                The start of next SD/MMC card can be configured with
→would                    CONFIG_MMC_BLOCK_MINORS, or overridden at boot/
                           time using the mmcblk.perdev_minors option. That

```

bump the offset between each card to be the  
 ↪configured value instead of the default 8.

179 char CCube DVXChip-based PCI products  
         0 = /dev/dvxirq0       First DVX device  
         1 = /dev/dvxirq1       Second DVX device  
         ...

180 char USB devices  
         0 = /dev/usb/lp0       First USB printer  
         ...  
         15 = /dev/usb/lp15     16th USB printer  
         48 = /dev/usb/scanner0 First USB scanner  
         ...  
         63 = /dev/usb/scanner15 16th USB scanner  
         64 = /dev/usb/rio500   Diamond Rio 500  
         65 = /dev/usb/usblcd   USBLCD Interface ↪  
 ↪(info@usblcd.de)  
         66 = /dev/usb/cpad0     Synaptics cPad (mouse/LCD)  
         96 = /dev/usb/hiddev0   1st USB HID device  
         ...  
         111 = /dev/usb/hiddev15 16th USB HID device  
         112 = /dev/usb/auer0     1st auerswald ISDN device  
         ...  
         127 = /dev/usb/auer15   16th auerswald ISDN device  
         128 = /dev/usb/brlvgr0   First Braille Voyager device  
         ...  
         131 = /dev/usb/brlvgr3   Fourth Braille Voyager ↪  
 ↪device  
         132 = /dev/usb/idmouse   ID Mouse (fingerprint ↪  
 ↪scanner) device  
         133 = /dev/usb/sisusbvga1       First SiSUSB VGA ↪  
 ↪device  
         ...  
         140 = /dev/usb/sisusbvga8       Eighth SISUSB VGA ↪  
 ↪device  
         144 = /dev/usb/lcd       USB LCD device  
         160 = /dev/usb/legousbtower0   1st USB Legotower ↪  
 ↪device  
         ...  
         175 = /dev/usb/legousbtower15   16th USB Legotower ↪  
 ↪device  
         176 = /dev/usb/usbtmc1   First USB TMC device  
         ...  
         191 = /dev/usb/usbtmc16 16th USB TMC device  
         192 = /dev/usb/yurex1   First USB Yurex device  
         ...  
         209 = /dev/usb/yurex16   16th USB Yurex device

180 block USB block devices

	0 = /dev/uba	First USB block device
	8 = /dev/ubb	Second USB block device
	16 = /dev/ubc	Third USB block device
	...	
181 char	Conrad Electronic parallel port radio clocks	
	0 = /dev/pcfclock0	First Conrad radio clock
	1 = /dev/pcfclock1	Second Conrad radio clock
	...	
182 char	Picture Elements THR2 binarizer	
	0 = /dev/pethr0	First THR2 board
	1 = /dev/pethr1	Second THR2 board
	...	
183 char	SST 5136-DN DeviceNet interface	
	0 = /dev/ss5136dn0	First DeviceNet interface
	1 = /dev/ss5136dn1	Second DeviceNet interface
	...	
	This device used to be assigned to major number 144. It had to be moved due to an unfortunate conflict.	
184 char	Picture Elements' video simulator/sender	
	0 = /dev/pevss0	First sender board
	1 = /dev/pevss1	Second sender board
	...	
185 char	InterMezzo high availability file system	
	0 = /dev/intermezzo0	First cache manager
	1 = /dev/intermezzo1	Second cache manager
	...	
	See <a href="http://web.archive.org/web/20080115195241/http://inter-mezzo.org/index.html">http://web.archive.org/web/20080115195241/ http://inter-mezzo.org/index.html</a>	
186 char	Object-based storage control device	
	0 = /dev/obd0	First obd control device
	1 = /dev/obd1	Second obd control device
	...	
	See <a href="ftp://ftp.lustre.org/pub/obd">ftp://ftp.lustre.org/pub/obd</a> for code and <a href="#">information</a> .	
187 char	DESkey hardware encryption device	
	0 = /dev/deskey0	First DES key
	1 = /dev/deskey1	Second DES key
	...	
188 char	USB serial converters	
	0 = /dev/ttyUSB0	First USB serial converter

```

        1 = /dev/ttyUSB1      Second USB serial converter
        ...

189 char    USB serial converters - alternate devices
        0 = /dev/cuusb0      Callout device for ttyUSB0
        1 = /dev/cuusb1      Callout device for ttyUSB1
        ...

190 char    Kansas City tracker/tuner card
        0 = /dev/kctt0       First KCT/T card
        1 = /dev/kctt1       Second KCT/T card
        ...

191 char    Reserved for PCMCIA

192 char    Kernel profiling interface
        0 = /dev/profile      Profiling control device
        1 = /dev/profile0     Profiling device for CPU 0
        2 = /dev/profile1     Profiling device for CPU 1
        ...

193 char    Kernel event-tracing interface
        0 = /dev/trace        Tracing control device
        1 = /dev/trace0       Tracing device for CPU 0
        2 = /dev/trace1       Tracing device for CPU 1
        ...

194 char    linVideoStreams (LIVS)
        0 = /dev/mvideo/status0  Video compression_
↳status
        1 = /dev/mvideo/stream0   Video stream
        2 = /dev/mvideo/frame0    Single compressed_
↳frame
        3 = /dev/mvideo/rawframe0  Raw uncompressed_
↳frame
        4 = /dev/mvideo/codec0     Direct codec access
        5 = /dev/mvideo/video4linux0 Video4Linux_
↳compatibility

        16 = /dev/mvideo/status1    Second device
        ...
        32 = /dev/mvideo/status2    Third device
        ...
        240 = /dev/mvideo/status15  16th device
        ...

195 char    Nvidia graphics devices
        0 = /dev/nvidia0          First Nvidia card
        1 = /dev/nvidia1          Second Nvidia card
        ...

```

	255 = /dev/nvidiactl	Nvidia card control
↪device		
196 char	Tormenta T1 card	
	0 = /dev/tor/0	Master control
↪channel for all cards		
	1 = /dev/tor/1	First DS0
	2 = /dev/tor/2	Second DS0
	...	
	48 = /dev/tor/48	48th DS0
	49 = /dev/tor/49	First pseudo-channel
	50 = /dev/tor/50	Second
↪pseudo-channel		
	...	
197 char	OpenTNF tracing facility	
	0 = /dev/tnf/t0	Trace 0 data
↪extraction		
	1 = /dev/tnf/t1	Trace 1 data
↪extraction		
	...	
	128 = /dev/tnf/status	Tracing facility
↪status		
	130 = /dev/tnf/trace	Tracing device
198 char	Total Impact TPMP2 quad coprocessor PCI card	
	0 = /dev/tpmp2/0	First card
	1 = /dev/tpmp2/1	Second card
	...	
199 char	Veritas volume manager (VxVM) volumes	
	0 = /dev/vx/rdisk/*/*	First volume
	1 = /dev/vx/rdisk/*/*	Second volume
	...	
199 block	Veritas volume manager (VxVM) volumes	
	0 = /dev/vx/dsk/*/*	First volume
	1 = /dev/vx/dsk/*/*	Second volume
	...	
	The namespace in these directories is maintained by the user space VxVM software.	
200 char	Veritas VxVM configuration interface	
	0 = /dev/vx/config	Configuration
↪access node		
	1 = /dev/vx/trace	Volume i/o trace
↪access node		
	2 = /dev/vx/iod	Volume i/o daemon
↪access node		
	3 = /dev/vx/info	Volume information

↪access node	4 = /dev/vx/task	Volume tasks access <a href="#">↵</a>
↪node	5 = /dev/vx/taskmon	Volume tasks <a href="#">↵</a>
↪monitor daemon		
201 char	Veritas VxVM dynamic multipathing driver	
↪device	0 = /dev/vx/rdmp/*	First multipath <a href="#">↵</a>
↪device	1 = /dev/vx/rdmp/*	Second multipath <a href="#">↵</a>
	...	
201 block	Veritas VxVM dynamic multipathing driver	
↪device	0 = /dev/vx/dmp/*	First multipath <a href="#">↵</a>
↪device	1 = /dev/vx/dmp/*	Second multipath <a href="#">↵</a>
	...	
	The namespace in these directories is maintained by the user space VxVM software.	
202 char	CPU model-specific registers	
	0 = /dev/cpu/0/msr	MSRs on CPU 0
	1 = /dev/cpu/1/msr	MSRs on CPU 1
	...	
202 block	Xen Virtual Block Device	
	0 = /dev/xvda	First Xen VBD whole disk
	16 = /dev/xvdb	Second Xen VBD whole disk
	32 = /dev/xvdc	Third Xen VBD whole disk
	...	
	240 = /dev/xvdp	Sixteenth Xen VBD whole disk
	Partitions are handled in the same way as for IDE disks (see major number 3) except that the limit on partitions is 15.	
203 char	CPU CPUID information	
	0 = /dev/cpu/0/cpuid	CPUID on CPU 0
	1 = /dev/cpu/1/cpuid	CPUID on CPU 1
	...	
204 char	Low-density serial ports	
↪L72xx UART - port 0	0 = /dev/ttyLU0	LinkUp Systems <a href="#">↵</a>
↪L72xx UART - port 1	1 = /dev/ttyLU1	LinkUp Systems <a href="#">↵</a>
↪L72xx UART - port 2	2 = /dev/ttyLU2	LinkUp Systems <a href="#">↵</a>
↪L72xx UART - port 3	3 = /dev/ttyLU3	LinkUp Systems <a href="#">↵</a>

↪L72xx UART - port 3	4 = /dev/ttyFB0	Intel Footbridge <a href="#">↵</a>
↪(ARM)	5 = /dev/ttySA0	StrongARM builtin <a href="#">↵</a>
↪serial port 0	6 = /dev/ttySA1	StrongARM builtin <a href="#">↵</a>
↪serial port 1	7 = /dev/ttySA2	StrongARM builtin <a href="#">↵</a>
↪serial port 2	8 = /dev/ttySC0	SCI serial port <a href="#">↵</a>
↪(SuperH) - port 0	9 = /dev/ttySC1	SCI serial port <a href="#">↵</a>
↪(SuperH) - port 1	10 = /dev/ttySC2	SCI serial port <a href="#">↵</a>
↪(SuperH) - port 2	11 = /dev/ttySC3	SCI serial port <a href="#">↵</a>
↪(SuperH) - port 3	12 = /dev/ttyFW0	Firmware console - <a href="#">↵</a>
↪port 0	13 = /dev/ttyFW1	Firmware console - <a href="#">↵</a>
↪port 1	14 = /dev/ttyFW2	Firmware console - <a href="#">↵</a>
↪port 2	15 = /dev/ttyFW3	Firmware console - <a href="#">↵</a>
↪port 3	16 = /dev/ttyAM0	ARM "AMBA" serial <a href="#">↵</a>
↪port 0	...	
↪port 15	31 = /dev/ttyAM15	ARM "AMBA" serial <a href="#">↵</a>
↪port 0	32 = /dev/ttyDB0	DataBooster serial <a href="#">↵</a>
	...	
↪port 7	39 = /dev/ttyDB7	DataBooster serial <a href="#">↵</a>
↪port	40 = /dev/ttySG0	SGI Altix console <a href="#">↵</a>
↪port 0	41 = /dev/ttySMX0	Motorola i.MX - <a href="#">↵</a>
↪port 1	42 = /dev/ttySMX1	Motorola i.MX - <a href="#">↵</a>
↪port 2	43 = /dev/ttySMX2	Motorola i.MX - <a href="#">↵</a>
↪0 (obsolete unused)	44 = /dev/ttyMM0	Marvell MPSC - port <a href="#">↵</a>
↪1 (obsolete unused)	45 = /dev/ttyMM1	Marvell MPSC - port <a href="#">↵</a>
↪SMC) - port 0	46 = /dev/ttyCPM0	PPC CPM (SCC or <a href="#">↵</a>
	...	
	47 = /dev/ttyCPM5	PPC CPM (SCC or <a href="#">↵</a>

↪SMC) - port 5	50 = /dev/ttyIOC0	Altix serial card
	...	
	81 = /dev/ttyIOC31	Altix serial card
	82 = /dev/ttyVR0	NEC VR4100 series <a href="#">↵</a>
↪SIU		
	83 = /dev/ttyVR1	NEC VR4100 series <a href="#">↵</a>
↪DSIU		
	84 = /dev/ttyIOC84	Altix ioc4 serial <a href="#">↵</a>
↪card		
	...	
	115 = /dev/ttyIOC115	Altix ioc4 serial <a href="#">↵</a>
↪card		
	116 = /dev/ttySIOC0	Altix ioc3 serial <a href="#">↵</a>
↪card		
	...	
	147 = /dev/ttySIOC31	Altix ioc3 serial <a href="#">↵</a>
↪card		
	148 = /dev/ttyPSC0	PPC PSC - port 0
	...	
	153 = /dev/ttyPSC5	PPC PSC - port 5
	154 = /dev/ttyAT0	ATMEL serial port 0
	...	
	169 = /dev/ttyAT15	ATMEL serial port 15
	170 = /dev/ttyNX0	Hilscher netX <a href="#">↵</a>
↪serial port 0		
	...	
	185 = /dev/ttyNX15	Hilscher netX <a href="#">↵</a>
↪serial port 15		
	186 = /dev/ttyJ0	JTAG1 DCC protocol <a href="#">↵</a>
↪based serial port emulation		
	187 = /dev/ttyUL0	Xilinx uartlite - <a href="#">↵</a>
↪port 0		
	...	
	190 = /dev/ttyUL3	Xilinx uartlite - <a href="#">↵</a>
↪port 3		
	191 = /dev/xvc0	Xen virtual console <a href="#">↵</a>
↪- port 0		
	192 = /dev/ttyPZ0	pmac_zilog - port 0
	...	
	195 = /dev/ttyPZ3	pmac_zilog - port 3
	196 = /dev/ttyTX0	TX39/49 serial port <a href="#">↵</a>
↪0		
	...	
	204 = /dev/ttyTX7	TX39/49 serial port <a href="#">↵</a>
↪7		
	205 = /dev/ttySC0	SC26xx serial port 0
	206 = /dev/ttySC1	SC26xx serial port 1
	207 = /dev/ttySC2	SC26xx serial port 2
	208 = /dev/ttySC3	SC26xx serial port 3
	209 = /dev/ttyMAX0	MAX3100 serial port <a href="#">↵</a>

↪0	210 = /dev/ttyMAX1	MAX3100 serial port
↪1	211 = /dev/ttyMAX2	MAX3100 serial port
↪2	212 = /dev/ttyMAX3	MAX3100 serial port
↪3		
205 char	Low-density serial ports (alternate device)	
↪ttyLU0	0 = /dev/culu0	Callout device for
↪ttyLU1	1 = /dev/culu1	Callout device for
↪ttyLU2	2 = /dev/culu2	Callout device for
↪ttyLU3	3 = /dev/culu3	Callout device for
↪ttyFB0	4 = /dev/cufb0	Callout device for
↪ttySA0	5 = /dev/cusa0	Callout device for
↪ttySA1	6 = /dev/cusa1	Callout device for
↪ttySA2	7 = /dev/cusa2	Callout device for
↪ttySC0	8 = /dev/cusc0	Callout device for
↪ttySC1	9 = /dev/cusc1	Callout device for
↪ttySC2	10 = /dev/cusc2	Callout device for
↪ttySC3	11 = /dev/cusc3	Callout device for
↪ttyFW0	12 = /dev/cufw0	Callout device for
↪ttyFW1	13 = /dev/cufw1	Callout device for
↪ttyFW2	14 = /dev/cufw2	Callout device for
↪ttyFW3	15 = /dev/cufw3	Callout device for
↪ttyAM0	16 = /dev/cuam0	Callout device for
	...	
↪ttyAM15	31 = /dev/cuam15	Callout device for
↪ttyDB0	32 = /dev/cudb0	Callout device for
	...	
↪ttyDB7	39 = /dev/cudb7	Callout device for

<code>↪ttySG0</code>	40 = /dev/cusg0	Callout device for <a href="#">u</a>
<code>↪ttySMX0</code>	41 = /dev/ttycusmx0	Callout device for <a href="#">u</a>
<code>↪ttySMX1</code>	42 = /dev/ttycusmx1	Callout device for <a href="#">u</a>
<code>↪ttySMX2</code>	43 = /dev/ttycusmx2	Callout device for <a href="#">u</a>
<code>↪ttyCPM0</code>	46 = /dev/cucpm0	Callout device for <a href="#">u</a>
	...	
<code>↪ttyCPM5</code>	49 = /dev/cucpm5	Callout device for <a href="#">u</a>
<code>↪ttyIOC40</code>	50 = /dev/cuioc40	Callout device for <a href="#">u</a>
	...	
<code>↪ttyIOC431</code>	81 = /dev/cuioc431	Callout device for <a href="#">u</a>
<code>↪ttyVR0</code>	82 = /dev/cuvr0	Callout device for <a href="#">u</a>
<code>↪ttyVR1</code>	83 = /dev/cuvr1	Callout device for <a href="#">u</a>
206 char	OnStream SC-x0 tape devices	
<code>↪tape, mode 0</code>	0 = /dev/osst0	First OnStream SCSI <a href="#">u</a>
<code>↪SCSI tape, mode 0</code>	1 = /dev/osst1	Second OnStream <a href="#">u</a>
	...	
<code>↪tape, mode 1</code>	32 = /dev/osst0l	First OnStream SCSI <a href="#">u</a>
<code>↪SCSI tape, mode 1</code>	33 = /dev/osst1l	Second OnStream <a href="#">u</a>
	...	
<code>↪tape, mode 2</code>	64 = /dev/osst0m	First OnStream SCSI <a href="#">u</a>
<code>↪SCSI tape, mode 2</code>	65 = /dev/osst1m	Second OnStream <a href="#">u</a>
	...	
<code>↪tape, mode 3</code>	96 = /dev/osst0a	First OnStream SCSI <a href="#">u</a>
<code>↪SCSI tape, mode 3</code>	97 = /dev/osst1a	Second OnStream <a href="#">u</a>
	...	
<code>↪of /dev/osst0</code>	128 = /dev/nosst0	No rewind version <a href="#">u</a>
<code>↪of /dev/osst1</code>	129 = /dev/nosst1	No rewind version <a href="#">u</a>
	...	
<code>↪of /dev/osst0l</code>	160 = /dev/nosst0l	No rewind version <a href="#">u</a>

```

    161 = /dev/nosst1l          No rewind version_
↳of /dev/osst1l
    ...
    192 = /dev/nosst0m        No rewind version_
↳of /dev/osst0m
    193 = /dev/nosst1m        No rewind version_
↳of /dev/osst1m
    ...
    224 = /dev/nosst0a        No rewind version_
↳of /dev/osst0a
    225 = /dev/nosst1a        No rewind version_
↳of /dev/osst1a
    ...

```

The OnStream SC-x0 SCSI tapes do not support the standard SCSI SASD command set and therefore need their own driver "osst". Note that the IDE, USB (and maybe ParPort) versions may be driven via ide-scsi\_

```

↳or
    usb-storage SCSI emulation and this osst device and
    driver as well. The ADR-x0 drives are QIC-157
    compliant and don't need osst.

```

```

207 char    Compaq ProLiant health feature indicate
            0 = /dev/cpqhealth/cpqw      Redirector interface
            1 = /dev/cpqhealth/crom      EISA CROM
            2 = /dev/cpqhealth/cdt       Data Table
            3 = /dev/cpqhealth/cevt      Event Log
            4 = /dev/cpqhealth/casr      Automatic Server_
↳Recovery
            5 = /dev/cpqhealth/cecc      ECC Memory
            6 = /dev/cpqhealth/cmca      Machine Check_
↳Architecture
            7 = /dev/cpqhealth/ccsm      Deprecated CDT
            8 = /dev/cpqhealth/cnmi      NMI Handling
            9 = /dev/cpqhealth/css       Sideshow Management
            10 = /dev/cpqhealth/cram     CMOS interface
            11 = /dev/cpqhealth/cpci     PCI IRQ interface

```

```

208 char    User space serial ports
            0 = /dev/ttyU0              First user space_
↳serial port
            1 = /dev/ttyU1              Second user space_
↳serial port
            ...

```

```

209 char    User space serial ports (alternate devices)
            0 = /dev/cuu0                Callout device for_
↳ttyU0
            1 = /dev/cuu1                Callout device for_
↳ttyU1

```

```

...
210 char      SBE, Inc. sync/async serial card
               0 = /dev/sbei/wxcfg0      Configuration_
↳device for board 0
               1 = /dev/sbei/dld0       Download device for_
↳board 0
               2 = /dev/sbei/wan00     WAN device, port 0,_
↳board 0
               3 = /dev/sbei/wan01     WAN device, port 1,_
↳board 0
               4 = /dev/sbei/wan02     WAN device, port 2,_
↳board 0
               5 = /dev/sbei/wan03     WAN device, port 3,_
↳board 0
               6 = /dev/sbei/wanc00    WAN clone device,_
↳port 0, board 0
               7 = /dev/sbei/wanc01    WAN clone device,_
↳port 1, board 0
               8 = /dev/sbei/wanc02    WAN clone device,_
↳port 2, board 0
               9 = /dev/sbei/wanc03    WAN clone device,_
↳port 3, board 0
               10 = /dev/sbei/wxcfg1   Configuration_
↳device for board 1
               11 = /dev/sbei/dld1     Download device for_
↳board 1
               12 = /dev/sbei/wan10    WAN device, port 0,_
↳board 1
               13 = /dev/sbei/wan11    WAN device, port 1,_
↳board 1
               14 = /dev/sbei/wan12    WAN device, port 2,_
↳board 1
               15 = /dev/sbei/wan13    WAN device, port 3,_
↳board 1
               16 = /dev/sbei/wanc10    WAN clone device,_
↳port 0, board 1
               17 = /dev/sbei/wanc11    WAN clone device,_
↳port 1, board 1
               18 = /dev/sbei/wanc12    WAN clone device,_
↳port 2, board 1
               19 = /dev/sbei/wanc13    WAN clone device,_
↳port 3, board 1
...

```

Yes, each board is really spaced 10 (decimal) apart.

```

211 char      Addinun CPCI1500 digital I/O card
               0 = /dev/addinum/cpci1500/0  First CPCI1500 card
               1 = /dev/addinum/cpci1500/1  Second CPCI1500 card
...

```

212 char          LinuxTV.org DVB driver subsystem  
                  0 = /dev/dvb/adapter0/video0      first video\_

↳decoder of first card  
                  1 = /dev/dvb/adapter0/audio0      first audio\_

↳decoder of first card  
                  2 = /dev/dvb/adapter0/sec0          (obsolete/unused)  
                  3 = /dev/dvb/adapter0/frontend0    first frontend\_

↳device of first card  
                  4 = /dev/dvb/adapter0/demux0      first demux\_

↳device of first card  
                  5 = /dev/dvb/adapter0/dvr0        first digital\_

↳video recoder device of first card  
                  6 = /dev/dvb/adapter0/ca0        first common\_

↳access port of first card  
                  7 = /dev/dvb/adapter0/net0        first network\_

↳device of first card  
                  8 = /dev/dvb/adapter0/osd0        first\_

↳on-screen-display device of first card  
                  9 = /dev/dvb/adapter0/video1      second video\_

↳decoder of first card  
                  ...  
                  64 = /dev/dvb/adapter1/video0      first video\_

↳decoder of second card  
                  ...  
                  128 = /dev/dvb/adapter2/video0    first video\_

↳decoder of third card  
                  ...  
                  196 = /dev/dvb/adapter3/video0    first video\_

↳decoder of fourth card

216 char          Bluetooth RFCOMM TTY devices  
                  0 = /dev/rfcomm0                  First Bluetooth\_

↳RFCOMM TTY device  
                  1 = /dev/rfcomm1                  Second Bluetooth\_

↳RFCOMM TTY device  
                  ...

217 char          Bluetooth RFCOMM TTY devices (alternate devices)  
                  0 = /dev/curf0                    Callout device for\_

↳rfcomm0  
                  1 = /dev/curf1                    Callout device for\_

↳rfcomm1  
                  ...

218 char          The Logical Company bus Unibus/Qbus adapters  
                  0 = /dev/logicalco/bci/0          First bus adapter  
                  1 = /dev/logicalco/bci/1          First bus adapter  
                  ...

219 char          The Logical Company DCI-1300 digital I/O card

```

    0 = /dev/logicalco/dci1300/0  First DCI-1300 card
    1 = /dev/logicalco/dci1300/1  Second DCI-1300 card
    ...

220 char      Myricom Myrinet "GM" board
↳board       0 = /dev/myricom/gm0      First Myrinet GM
↳access"     1 = /dev/myricom/gmp0   First board "root
↳board       2 = /dev/myricom/gm1   Second Myrinet GM
↳access"     3 = /dev/myricom/gmp1   Second board "root
    ...

221 char      VME bus
    0 = /dev/bus/vme/m0      First master image
    1 = /dev/bus/vme/m1     Second master image
    2 = /dev/bus/vme/m2     Third master image
    3 = /dev/bus/vme/m3     Fourth master image
    4 = /dev/bus/vme/s0     First slave image
    5 = /dev/bus/vme/s1     Second slave image
    6 = /dev/bus/vme/s2     Third slave image
    7 = /dev/bus/vme/s3     Fourth slave image
    8 = /dev/bus/vme/ctl     Control

It is expected that all VME bus drivers will use the
same interface.  For interface documentation see
http://www.vmlinux.org/.

224 char      A2232 serial card
    0 = /dev/ttyY0          First A2232 port
    1 = /dev/ttyY1          Second A2232 port
    ...

225 char      A2232 serial card (alternate devices)
↳ttyY0       0 = /dev/cuy0          Callout device for
↳ttyY1       1 = /dev/cuy1          Callout device for
    ...

226 char      Direct Rendering Infrastructure (DRI)
    0 = /dev/dri/card0      First graphics card
    1 = /dev/dri/card1      Second graphics card
    ...

227 char      IBM 3270 terminal Unix tty access
↳terminal    1 = /dev/3270/tty1     First 3270 terminal
            2 = /dev/3270/tty2     Seconds 3270

```

```

    ...
228 char      IBM 3270 terminal block-mode access
              0 = /dev/3270/tub           Controlling_
↳interface
              1 = /dev/3270/tub1        First 3270 terminal
              2 = /dev/3270/tub2        Second 3270 terminal
    ...

229 char      IBM iSeries/pSeries virtual console
              0 = /dev/hvc0             First console port
              1 = /dev/hvc1             Second console port
    ...

230 char      IBM iSeries virtual tape
↳mode 0       0 = /dev/iseriess/vt0      First virtual tape,_
↳ mode 0      1 = /dev/iseriess/vt1      Second virtual tape,
    ...
↳mode 1       32 = /dev/iseriess/vt0l    First virtual tape,_
↳ mode 1      33 = /dev/iseriess/vt1l    Second virtual tape,
    ...
↳mode 2       64 = /dev/iseriess/vt0m    First virtual tape,_
↳ mode 2      65 = /dev/iseriess/vt1m    Second virtual tape,
    ...
↳mode 3       96 = /dev/iseriess/vt0a    First virtual tape,_
↳ mode 3      97 = /dev/iseriess/vt1a    Second virtual tape,
    ...
↳mode 0, no rewind 128 = /dev/iseriess/nvt0  First virtual tape,_
↳ mode 0, no rewind 129 = /dev/iseriess/nvt1  Second virtual tape,
    ...
↳mode 1, no rewind 160 = /dev/iseriess/nvt0l  First virtual tape,_
↳ mode 1, no rewind 161 = /dev/iseriess/nvt1l  Second virtual tape,
    ...
↳mode 2, no rewind 192 = /dev/iseriess/nvt0m  First virtual tape,_
↳ mode 2, no rewind 193 = /dev/iseriess/nvt1m  Second virtual tape,
    ...
↳ mode 2, no rewind 224 = /dev/iseriess/nvt0a  First virtual tape,_
```

```

↪mode 3, no rewind
    225 = /dev/iseries/nvt1a        Second virtual tape,
↪ mode 3, no rewind
    ...

    "No rewind" refers to the omission of the default
    automatic rewind on device close. The MTREW or
↪MTOFFL
    ioctl()'s can be used to rewind the tape regardless
↪of
    the device used to access it.

231 char    InfiniBand
    0 = /dev/infiniband/umad0
    1 = /dev/infiniband/umad1
    ...
↪device    63 = /dev/infiniband/umad63    63rd InfiniBandMad
↪IsSM device    64 = /dev/infiniband/issm0    First InfiniBand
↪IsSM device    65 = /dev/infiniband/issm1    Second InfiniBand
    ...
↪IsSM device    127 = /dev/infiniband/issm63    63rd InfiniBand
↪verbs device    128 = /dev/infiniband/uverbs0    First InfiniBand
↪verbs device    129 = /dev/infiniband/uverbs1    Second InfiniBand
    ...
↪verbs device    159 = /dev/infiniband/uverbs31    31st InfiniBand

232 char    Biometric Devices
    0 = /dev/biometric/sensor0/fingerprint    first
↪fingerprint sensor on first device
    1 = /dev/biometric/sensor0/iris          first iris
↪sensor on first device
    2 = /dev/biometric/sensor0/retina        first
↪retina sensor on first device
    3 = /dev/biometric/sensor0/voiceprint    first
↪voiceprint sensor on first device
    4 = /dev/biometric/sensor0/facial        first
↪facial sensor on first device
    5 = /dev/biometric/sensor0/hand          first hand
↪sensor on first device
    ...
↪fingerprint sensor on second device    10 = /dev/biometric/sensor1/fingerprint    first
    ...
    20 = /dev/biometric/sensor2/fingerprint    first

```

```
↳fingerprint sensor on third device
    ...
233 char      PathScale InfiniPath interconnect
              0 = /dev/ipath      Primary device for programs
↳(any unit)   1 = /dev/ipath0      Access specifically to unit 0
              2 = /dev/ipath1      Access specifically to unit 1
              ...
              4 = /dev/ipath3      Access specifically to unit 3
              129 = /dev/ipath_sma  Device used by Subnet
↳Management Agent
              130 = /dev/ipath_diag Device used by diagnostics
↳programs

234-254      char      RESERVED FOR DYNAMIC ASSIGNMENT
              Character devices that request a dynamic allocation
↳of major number will
              take numbers starting from 254 and downward.

240-254      block    LOCAL/EXPERIMENTAL USE
              Allocated for local/experimental use. For devices
↳not
              assigned official numbers, these ranges should be
              used in order to avoid conflicting with future
↳assignments.

255 char      RESERVED

255 block    RESERVED

              This major is reserved to assist the expansion to a
              larger number space. No device nodes with this
↳major
              should ever be created on the filesystem.
              (This is probably not true anymore, but I'll leave
↳it
              for now /Torben)

---LARGE MAJORS!!!!!---

256 char      Equinox SST multi-port serial boards
              0 = /dev/ttyEQ0      First serial port on first
↳Equinox SST board
              127 = /dev/ttyEQ127  Last serial port on first
↳Equinox SST board
              128 = /dev/ttyEQ128  First serial port on second
↳Equinox SST board
              ...
              1027 = /dev/ttyEQ1027 Last serial port on eighth
↳Equinox SST board
```

256 block	Resident Flash Disk Flash Translation Layer
	0 = /dev/rfda First RFD FTL layer
	16 = /dev/rfdb Second RFD FTL layer
	...
	240 = /dev/rfdp 16th RFD FTL layer
257 char	Phoenix Technologies Cryptographic Services Driver
	0 = /dev/ptlsec Crypto Services Driver
257 block	SSFDC Flash Translation Layer filesystem
	0 = /dev/ssfdca First SSFDC layer
	8 = /dev/ssfdb Second SSFDC layer
	16 = /dev/ssfdcc Third SSFDC layer
	24 = /dev/ssfdcd 4th SSFDC layer
	32 = /dev/ssfdce 5th SSFDC layer
	40 = /dev/ssfdcf 6th SSFDC layer
	48 = /dev/ssfdcg 7th SSFDC layer
	56 = /dev/ssfdch 8th SSFDC layer
258 block	ROM/Flash read-only translation layer
	0 = /dev/blockrom0 First ROM card's
↳translation layer interface	
	1 = /dev/blockrom1 Second ROM card's
↳translation layer interface	
	...
259 block	Block Extended Major
	Used dynamically to hold additional partition
↳minor	
	numbers and allow large numbers of partitions per
↳device	
259 char	FPGA configuration interfaces
	0 = /dev/icap0 First Xilinx internal
↳configuration	
	1 = /dev/icap1 Second Xilinx internal
↳configuration	
260 char	OSD (Object-based-device) SCSI Device
	0 = /dev/osd0 First OSD Device
	1 = /dev/osd1 Second OSD Device
	...
	255 = /dev/osd255 256th OSD Device
384-511 char	RESERVED FOR DYNAMIC ASSIGNMENT
	Character devices that request a dynamic allocation
↳of major	
	number will take numbers starting from 511 and
↳downward,	
	once the 234-254 range is full.

## 3.1 Additional /dev/ directory entries

This section details additional entries that should or may exist in the /dev directory. It is preferred that symbolic links use the same form (absolute or relative) as is indicated here. Links are classified as “hard” or “symbolic” depending on the preferred type of link; if possible, the indicated type of link should be used.

### 3.1.1 Compulsory links

These links should exist on all systems:

/dev/fd	/proc/self/fd	symbolic	File descriptors
/dev/stdin	fd/0	symbolic	stdin file descriptor
/dev/stdout	fd/1	symbolic	stdout file descriptor
/dev/stderr	fd/2	symbolic	stderr file descriptor
/dev/nfsd	socksys	symbolic	Required by iBCS-2
/dev/X0R	null	symbolic	Required by iBCS-2

Note: /dev/X0R is <letter X>-<digit 0>-<letter R>.

### 3.1.2 Recommended links

It is recommended that these links exist on all systems:

/dev/core	/proc/kcore	symbolic	Backward compatibility
/dev/ramdisk	ram0	symbolic	Backward compatibility
/dev/ftape	qft0	symbolic	Backward compatibility
/dev/bttv0	video0	symbolic	Backward compatibility
/dev/radio	radio0	symbolic	Backward compatibility
/dev/i2o*	/dev/i2o/*	symbolic	Backward compatibility
/dev/scd?	sr?	hard	Alternate SCSI CD-ROM name

### 3.1.3 Locally defined links

The following links may be established locally to conform to the configuration of the system. This is merely a tabulation of existing practice, and does not constitute a recommendation. However, if they exist, they should have the following uses.

/dev/mouse	mouse port	symbolic	Current mouse device
/dev/tape	tape device	symbolic	Current tape device
/dev/cdrom	CD-ROM device	symbolic	Current CD-ROM device
/dev/cdwriter	CD-writer	symbolic	Current CD-writer device
/dev/scanner	scanner	symbolic	Current scanner device
/dev/modem	modem port	symbolic	Current dialout device
/dev/root	root device	symbolic	Current root filesystem
/dev/swap	swap device	symbolic	Current swap device

/dev/modem should not be used for a modem which supports dialin as well as dialout, as it tends to cause lock file problems. If it exists, /dev/modem should point to the appropriate primary TTY device (the use of the alternate callout devices is deprecated).

For SCSI devices, /dev/tape and /dev/cdrom should point to the cooked devices (/dev/st\* and /dev/sr\*, respectively), whereas /dev/cdwriter and /dev/scanner should point to the appropriate generic SCSI devices (/dev/sg\*).

/dev/mouse may point to a primary serial TTY device, a hardware mouse device, or a socket for a mouse driver program (e.g. /dev/gpmdata).

### 3.1.4 Sockets and pipes

Non-transient sockets and named pipes may exist in /dev. Common entries are:

/dev/printer	socket	lpd local socket
/dev/log	socket	syslog local socket
/dev/gpmdata	socket	gpm mouse multiplexer

### 3.1.5 Mount points

The following names are reserved for mounting special filesystems under /dev. These special filesystems provide kernel interfaces that cannot be provided with standard device nodes.

/dev/pts	devpts	PTY slave filesystem
/dev/shm	tmpfs	POSIX shared memory maintenance access

## 3.2 Terminal devices

Terminal, or TTY devices are a special class of character devices. A terminal device is any device that could act as a controlling terminal for a session; this includes virtual consoles, serial ports, and pseudoterminals (PTYs).

All terminal devices share a common set of capabilities known as line disciplines; these include the common terminal line discipline as well as SLIP and PPP modes.

All terminal devices are named similarly; this section explains the naming and use of the various types of TTYs. Note that the naming conventions include several historical warts; some of these are Linux-specific, some were inherited from other systems, and some reflect Linux outgrowing a borrowed convention.

A hash mark (#) in a device name is used here to indicate a decimal number without leading zeroes.

### 3.2.1 Virtual consoles and the console device

Virtual consoles are full-screen terminal displays on the system video monitor. Virtual consoles are named `/dev/tty#`, with numbering starting at `/dev/tty1`; `/dev/tty0` is the current virtual console. `/dev/tty0` is the device that should be used to access the system video card on those architectures for which the frame buffer devices (`/dev/fb*`) are not applicable. Do not use `/dev/console` for this purpose.

The console device, `/dev/console`, is the device to which system messages should be sent, and on which logins should be permitted in single-user mode. Starting with Linux 2.1.71, `/dev/console` is managed by the kernel; for previous versions it should be a symbolic link to either `/dev/tty0`, a specific virtual console such as `/dev/tty1`, or to a serial port primary (`tty*`, not `cu*`) device, depending on the configuration of the system.

### 3.2.2 Serial ports

Serial ports are RS-232 serial ports and any device which simulates one, either in hardware (such as internal modems) or in software (such as the ISDN driver.) Under Linux, each serial ports has two device names, the primary or callin device and the alternate or callout one. Each kind of device is indicated by a different letter. For any letter X, the names of the devices are `/dev/ttyX#` and `/dev/cux#`, respectively; for historical reasons, `/dev/ttyS#` and `/dev/ttyC#` correspond to `/dev/cua#` and `/dev/cub#`. In the future, it should be expected that multiple letters will be used; all letters will be upper case for the “tty” device (e.g. `/dev/ttyDP#`) and lower case for the “cu” device (e.g. `/dev/cudp#`).

The names `/dev/ttyQ#` and `/dev/cuq#` are reserved for local use.

The alternate devices provide for kernel-based exclusion and somewhat different defaults than the primary devices. Their main purpose is to allow the use of serial ports with programs with no inherent or broken support for serial ports. Their use is deprecated, and they may be removed from a future version of Linux.

Arbitration of serial ports is provided by the use of lock files with the names `/var/lock/LCK..ttyX#`. The contents of the lock file should be the PID of the locking process as an ASCII number.

It is common practice to install links such as `/dev/modem` which point to serial ports. In order to ensure proper locking in the presence of these links, it is recommended that software chase symlinks and lock all possible names; additionally, it is recommended that a lock file be installed with the corresponding alternate device. In order to avoid deadlocks, it is recommended that the locks are acquired in the following order, and released in the reverse:

1. The symbolic link name, if any (`/var/lock/LCK..modem`)
2. The “tty” name (`/var/lock/LCK..ttyS2`)
3. The alternate device name (`/var/lock/LCK..cua2`)

In the case of nested symbolic links, the lock files should be installed in the order the symlinks are resolved.

Under no circumstances should an application hold a lock while waiting for another to be released. In addition, applications which attempt to create lock files for the corresponding alternate device names should take into account the possibility of being used on a non-serial port TTY, for which no alternate device would exist.

### **3.2.3 Pseudoterminals (PTYs)**

Pseudoterminals, or PTYs, are used to create login sessions or provide other capabilities requiring a TTY line discipline (including SLIP or PPP capability) to arbitrary data-generation processes. Each PTY has a master side, named `/dev/pty[p-za-e][0-9a-f]`, and a slave side, named `/dev/tty[p-za-e][0-9a-f]`. The kernel arbitrates the use of PTYs by allowing each master side to be opened only once.

Once the master side has been opened, the corresponding slave device can be used in the same manner as any TTY device. The master and slave devices are connected by the kernel, generating the equivalent of a bidirectional pipe with TTY capabilities.

Recent versions of the Linux kernels and GNU libc contain support for the System V/Unix98 naming scheme for PTYs, which assigns a common device, `/dev/ptmx`, to all the masters (opening it will automatically give you a previously unassigned PTY) and a subdirectory, `/dev/pts`, for the slaves; the slaves are named with decimal integers (`/dev/pts/#` in our notation). This removes the problem of exhausting the namespace and enables the kernel to automatically create the device nodes for the slaves on demand using the “devpts” filesystem.



## **DOCUMENTATION FOR /PROC/SYS**

Copyright (c) 1998, 1999, Rik van Riel <[riel@nl.linux.org](mailto:riel@nl.linux.org)>

---

‘Why’ , I hear you ask, ‘would anyone even `_want_` documentation for them `sysctl` files? If anybody really needs it, it’ s all in the source...’

Well, this documentation is written because some people either don’ t know they need to tweak something, or because they don’ t have the time or knowledge to read the source code.

Furthermore, the programmers who built `sysctl` have built it to be actually used, not just for the fun of programming it :-)

---

Legal blurb:

As usual, there are two main things to consider:

1. you get what you pay for
2. it’ s free

The consequences are that I won’ t guarantee the correctness of this document, and if you come to me complaining about how you screwed up your system because of wrong documentation, I won’ t feel sorry for you. I might even laugh at you...

But of course, if you `_do_` manage to screw up your system using only the `sysctl` options used in this file, I’ d like to hear of it. Not only to have a great laugh, but also to make sure that you’ re the last RTFMing person to screw up.

In short, e-mail your suggestions, corrections and / or horror stories to: <[riel@nl.linux.org](mailto:riel@nl.linux.org)>

Rik van Riel.

---

### 4.1 Introduction

Sysctl is a means of configuring certain aspects of the kernel at run-time, and the `/proc/sys/` directory is there so that you don't even need special tools to do it! In fact, there are only four things needed to use these config facilities:

- a running Linux system
- root access
- common sense (this is especially hard to come by these days)
- knowledge of what all those values mean

As a quick `ls /proc/sys` will show, the directory consists of several (arch-dependent?) subdirs. Each subdir is mainly about one part of the kernel, so you can do configuration on a piece by piece basis, or just some 'thematic frobbing'.

This documentation is about:

abi/	execution domains & personalities
de- bug/	<empty>
dev/	device specific information (eg dev/cdrom/info)
fs/	specific filesystems filehandle, inode, dentry and quota tuning binfmt_misc <Documentation/admin-guide/binfmt-misc.rst>
ker- nel/	global kernel info / tuning miscellaneous stuff
net/	networking stuff, for documentation look in: <Documentation/networking/>
proc/	<empty>
sun- rpc/	SUN Remote Procedure Call (NFS)
vm/	memory management tuning buffer and cache management
user/	Per user per user namespace limits

These are the subdirs I have on my system. There might be more or other subdirs in another setup. If you see another dir, I'd really like to hear about it :-)

#### 4.1.1 Documentation for `/proc/sys/abi/`

kernel version 2.6.0.test2

Copyright (c) 2003, Fabian Frederick <[ffrederick@users.sourceforge.net](mailto:ffrederick@users.sourceforge.net)>

For general info: `index.rst`.

---

This path is binary emulation relevant aka personality types aka abi. When a process is executed, it's linked to an `exec_domain` whose personality is defined using values available from `/proc/sys/abi`. You can find further details about abi in `include/linux/personality.h`.

Here are the files featuring in 2.6 kernel:

- defhandler\_coff
- defhandler\_elf
- defhandler\_lcall7
- defhandler\_libcso
- fake\_utsname
- trace

### defhandler\_coff

**defined value:** PER\_SCOSVR3:

0x0003 | STICKY\_TIMEOUTS | WHOLE\_SECONDS | SHORT\_INODE

### defhandler\_elf

**defined value:** PER\_LINUX:

0

### defhandler\_lcall7

**defined value :** PER\_SVR4:

0x0001 | STICKY\_TIMEOUTS | MMAP\_PAGE\_ZERO,

### defhandler\_libsco

**defined value:** PER\_SVR4:

0x0001 | STICKY\_TIMEOUTS | MMAP\_PAGE\_ZERO,

### fake\_utsname

Unused

### trace

Unused

### 4.1.2 Documentation for /proc/sys/fs/

kernel version 2.2.10

Copyright (c) 1998, 1999, Rik van Riel <riel@nl.linux.org>

Copyright (c) 2009, Shen Feng <shen@cn.fujitsu.com>

For general info and legal blurb, please look in intro.rst.

---

This file contains documentation for the sysctl files in /proc/sys/fs/ and is valid for Linux kernel version 2.2.

The files in this directory can be used to tune and monitor miscellaneous and general things in the operation of the Linux kernel. Since some of the files can be used to screw up your system, it is advisable to read both documentation and source before actually making adjustments.

#### 1. /proc/sys/fs

Currently, these files are in /proc/sys/fs:

- aio-max-nr
- aio-nr
- dentry-state
- dquot-max
- dquot-nr
- file-max
- file-nr
- inode-max
- inode-nr
- inode-state
- nr\_open
- overflowuid
- overflowgid
- pipe-user-pages-hard
- pipe-user-pages-soft
- protected\_fifos
- protected\_hardlinks
- protected\_regular
- protected\_symlinks
- suid\_dumpable

- super-max
- super-nr

### aio-nr & aio-max-nr

aio-nr is the running total of the number of events specified on the `io_setup` system call for all currently active aio contexts. If aio-nr reaches aio-max-nr then `io_setup` will fail with EAGAIN. Note that raising aio-max-nr does not result in the pre-allocation or re-sizing of any kernel data structures.

### dentry-state

From `linux/include/linux/dcache.h`:

```
struct dentry_stat_t dentry_stat {
    int nr_dentry;
    int nr_unused;
    int age_limit;           /* age in seconds */
    int want_pages;        /* pages requested by system */
    int nr_negative;       /* # of unused negative dentries */
    int dummy;             /* Reserved for future use */
};
```

Dentries are dynamically allocated and deallocated.

`nr_dentry` shows the total number of dentries allocated (active + unused). `nr_unused` shows the number of dentries that are not actively used, but are saved in the LRU list for future reuse.

`Age_limit` is the age in seconds after which dcache entries can be reclaimed when memory is short and `want_pages` is nonzero when `shrink_dcache_pages()` has been called and the dcache isn't pruned yet.

`nr_negative` shows the number of unused dentries that are also negative dentries which do not map to any files. Instead, they help speeding up rejection of non-existing files provided by the users.

### dquot-max & dquot-nr

The file `dquot-max` shows the maximum number of cached disk quota entries.

The file `dquot-nr` shows the number of allocated disk quota entries and the number of free disk quota entries.

If the number of free cached disk quotas is very low and you have some awesome number of simultaneous system users, you might want to raise the limit.

### file-max & file-nr

The value in file-max denotes the maximum number of file- handles that the Linux kernel will allocate. When you get lots of error messages about running out of file handles, you might want to increase this limit.

Historically, the kernel was able to allocate file handles dynamically, but not to free them again. The three values in file-nr denote the number of allocated file handles, the number of allocated but unused file handles, and the maximum number of file handles. Linux 2.6 always reports 0 as the number of free file handles - this is not an error, it just means that the number of allocated file handles exactly matches the number of used file handles.

Attempts to allocate more file descriptors than file-max are reported with printk, look for “VFS: file-max limit <number> reached” .

### nr\_open

This denotes the maximum number of file-handles a process can allocate. Default value is 1024\*1024 (1048576) which should be enough for most machines. Actual limit depends on RLIMIT\_NOFILE resource limit.

### inode-max, inode-nr & inode-state

As with file handles, the kernel allocates the inode structures dynamically, but can't free them yet.

The value in inode-max denotes the maximum number of inode handlers. This value should be 3-4 times larger than the value in file-max, since stdin, stdout and network sockets also need an inode struct to handle them. When you regularly run out of inodes, you need to increase this value.

The file inode-nr contains the first two items from inode-state, so we' ll skip to that file...

Inode-state contains three actual numbers and four dummies. The actual numbers are, in order of appearance, nr\_inodes, nr\_free\_inodes and preshrink.

Nr\_inodes stands for the number of inodes the system has allocated, this can be slightly more than inode-max because Linux allocates them one pageful at a time.

Nr\_free\_inodes represents the number of free inodes (?) and preshrink is nonzero when the nr\_inodes > inode-max and the system needs to prune the inode list instead of allocating more.

### **overflowgid & overflowuid**

Some filesystems only support 16-bit UIDs and GIDs, although in Linux UIDs and GIDs are 32 bits. When one of these filesystems is mounted with writes enabled, any UID or GID that would exceed 65535 is translated to a fixed value before being written to disk.

These sysctls allow you to change the value of the fixed UID and GID. The default is 65534.

### **pipe-user-pages-hard**

Maximum total number of pages a non-privileged user may allocate for pipes. Once this limit is reached, no new pipes may be allocated until usage goes below the limit again. When set to 0, no limit is applied, which is the default setting.

### **pipe-user-pages-soft**

Maximum total number of pages a non-privileged user may allocate for pipes before the pipe size gets limited to a single page. Once this limit is reached, new pipes will be limited to a single page in size for this user in order to limit total memory usage, and trying to increase them using `fcntl()` will be denied until usage goes below the limit again. The default value allows to allocate up to 1024 pipes at their default size. When set to 0, no limit is applied.

### **protected\_fifos**

The intent of this protection is to avoid unintentional writes to an attacker-controlled FIFO, where a program expected to create a regular file.

When set to “0” , writing to FIFOs is unrestricted.

When set to “1” don’ t allow `O_CREAT` open on FIFOs that we don’ t own in world writable sticky directories, unless they are owned by the owner of the directory.

When set to “2” it also applies to group writable sticky directories.

This protection is based on the restrictions in Openwall.

### **protected\_hardlinks**

A long-standing class of security issues is the hardlink-based time-of-check-time-of-use race, most commonly seen in world-writable directories like `/tmp`. The common method of exploitation of this flaw is to cross privilege boundaries when following a given hardlink (i.e. a root process follows a hardlink created by another user). Additionally, on systems without separated partitions, this stops unauthorized users from “pinning” vulnerable `setuid/setgid` files against being upgraded by the administrator, or linking to special files.

When set to “0” , hardlink creation behavior is unrestricted.

When set to “1” hardlinks cannot be created by users if they do not already own the source file, or do not have read/write access to it.

This protection is based on the restrictions in Openwall and grsecurity.

### **protected\_regular**

This protection is similar to `protected_fifos`, but it avoids writes to an attacker-controlled regular file, where a program expected to create one.

When set to “0” , writing to regular files is unrestricted.

When set to “1” don’ t allow `O_CREAT` open on regular files that we don’ t own in world writable sticky directories, unless they are owned by the owner of the directory.

When set to “2” it also applies to group writable sticky directories.

### **protected\_symlinks**

A long-standing class of security issues is the symlink-based time-of-check-time-of-use race, most commonly seen in world-writable directories like `/tmp`. The common method of exploitation of this flaw is to cross privilege boundaries when following a given symlink (i.e. a root process follows a symlink belonging to another user). For a likely incomplete list of hundreds of examples across the years, please see: <http://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=/tmp>

When set to “0” , symlink following behavior is unrestricted.

When set to “1” symlinks are permitted to be followed only when outside a sticky world-writable directory, or when the uid of the symlink and follower match, or when the directory owner matches the symlink’ s owner.

This protection is based on the restrictions in Openwall and grsecurity.

**suid\_dumpable:**

This value can be used to query and set the core dump mode for setuid or otherwise protected/tainted binaries. The modes are

0	(default)	traditional behaviour. Any process which has changed privilege levels or is execute only will not be dumped.
1	(debug)	all processes dump core when possible. The core dump is owned by the current user and no security is applied. This is intended for system debugging situations only. Ptrace is unchecked. This is insecure as it allows regular users to examine the memory contents of privileged processes.
2	(suid safe)	any binary which normally would not be dumped is dumped anyway, but only if the "core_pattern" kernel sysctl is set to either a pipe handler or a fully qualified path. (For more details on this limitation, see CVE-2006-2451.) This mode is appropriate when administrators are attempting to debug problems in a normal environment, and either have a core dump pipe handler that knows to treat privileged core dumps with care, or specific directory defined for catching core dumps. If a core dump happens without a pipe handler or fully qualified path, a message will be emitted to syslog warning about the lack of a correct setting.

**super-max & super-nr**

These numbers control the maximum number of superblocks, and thus the maximum number of mounted filesystems the kernel can have. You only need to increase super-max if you need to mount more filesystems than the current value in super-max allows you to.

**aio-nr & aio-max-nr**

aio-nr shows the current system-wide number of asynchronous io requests. aio-max-nr allows you to change the maximum value aio-nr can grow to.

### mount-max

This denotes the maximum number of mounts that may exist in a mount namespace.

### 2. /proc/sys/fs/binfmt\_misc

Documentation for the files in /proc/sys/fs/binfmt\_misc is in Documentation/admin-guide/binfmt-misc.rst.

### 3. /proc/sys/fs/mqueue - POSIX message queues filesystem

The “mqueue” filesystem provides the necessary kernel features to enable the creation of a user space library that implements the POSIX message queues API (as noted by the MSG tag in the POSIX 1003.1-2001 version of the System Interfaces specification.)

The “mqueue” filesystem contains values for determining/setting the amount of resources used by the file system.

/proc/sys/fs/mqueue/queues\_max is a read/write file for setting/getting the maximum number of message queues allowed on the system.

/proc/sys/fs/mqueue/msg\_max is a read/write file for setting/getting the maximum number of messages in a queue value. In fact it is the limiting value for another (user) limit which is set in mq\_open invocation. This attribute of a queue must be less or equal then msg\_max.

/proc/sys/fs/mqueue/msgsize\_max is a read/write file for setting/getting the maximum message size value (it is every message queue's attribute set during its creation).

/proc/sys/fs/mqueue/msg\_default is a read/write file for setting/getting the default number of messages in a queue value if attr parameter of mq\_open(2) is NULL. If it exceed msg\_max, the default value is initialized msg\_max.

/proc/sys/fs/mqueue/msgsize\_default is a read/write file for setting/getting the default message size value if attr parameter of mq\_open(2) is NULL. If it exceed msgsize\_max, the default value is initialized msgsize\_max.

### 4. /proc/sys/fs/epoll - Configuration options for the epoll interface

This directory contains configuration options for the epoll(7) interface.

## **max\_user\_watches**

Every epoll file descriptor can store a number of files to be monitored for event readiness. Each one of these monitored files constitutes a “watch” . This configuration option sets the maximum number of “watches” that are allowed for each user. Each “watch” costs roughly 90 bytes on a 32bit kernel, and roughly 160 bytes on a 64bit one. The current default value for max\_user\_watches is the 1/32 of the available low memory, divided for the “watch” cost in bytes.

### **4.1.3 Documentation for /proc/sys/kernel/**

Copyright (c) 1998, 1999, Rik van Riel <riel@nl.linux.org>

Copyright (c) 2009, Shen Feng<shen@cn.fujitsu.com>

For general info and legal blurb, please look in Documentation for /proc/sys.

---

This file contains documentation for the sysctl files in /proc/sys/kernel/ and is valid for Linux kernel version 2.2.

The files in this directory can be used to tune and monitor miscellaneous and general things in the operation of the Linux kernel. Since some of the files can be used to screw up your system, it is advisable to read both documentation and source before actually making adjustments.

Currently, these files might (depending on your configuration) show up in /proc/sys/kernel:

- acct
- acpi\_video\_flags
- auto\_msgmni
- bootloader\_type (x86 only)
- bootloader\_version (x86 only)
- bpf\_stats\_enabled
- cad\_pid
- cap\_last\_cap
- core\_pattern
- core\_pipe\_limit
- core\_uses\_pid
- ctrl-alt-del
- dmesg\_restrict
- domainname & hostname
- firmware\_config

- `ftrace_dump_on_oops`
- `ftrace_enabled`, `stack_tracer_enabled`
- `hardlockup_all_cpu_backtrace`
- `hardlockup_panic`
- `hotplug`
- `hung_task_all_cpu_backtrace`:
- `hung_task_panic`
- `hung_task_check_count`
- `hung_task_timeout_secs`
- `hung_task_check_interval_secs`
- `hung_task_warnings`
- `hyperv_record_panic_msg`
- `ignore-unaligned-usertrap`
- `kexec_load_disabled`
- `kptr_restrict`
- `modprobe`
- `modules_disabled`
- `msgmax`, `msgmnb`, and `msgmni`
- `msg_next_id`, `sem_next_id`, and `shm_next_id` (System V IPC)
- `ngroups_max`
- `nmi_watchdog`
- `numa_balancing`
- `numa_balancing_scan_period_min_ms`, `numa_balancing_scan_delay_ms`,  
`numa_balancing_scan_period_max_ms`, `numa_balancing_scan_size_mb`
- `oops_all_cpu_backtrace`:
- `osrelease`, `ostype` & `version`
- `overflowgid` & `overflowuid`
- `panic`
- `panic_on_io_nmi`
- `panic_on_oops`
- `panic_on_stackoverflow`
- `panic_on_unrecovered_nmi`
- `panic_on_warn`
- `panic_print`

- panic\_on\_rcu\_stall
- perf\_cpu\_time\_max\_percent
- perf\_event\_paranoid
- perf\_event\_max\_stack
- perf\_event\_mlock\_kb
- perf\_event\_max\_contexts\_per\_stack
- pid\_max
- ns\_last\_pid
- powersave-nap (PPC only)
- printk
- printk\_delay
- printk\_ratelimit
- printk\_ratelimit\_burst
- printk\_devkmsg
- pty
- randomize\_va\_space
- real-root-dev
- reboot-cmd (SPARC only)
- sched\_energy\_aware
- sched\_schedstats
- seccomp
- sg-big-buff
- shmall
- shmmax
- shmmni
- shm\_rmid\_forced
- sysctl\_writes\_strict
- softlockup\_all\_cpu\_backtrace
- softlockup\_panic
- soft\_watchdog
- stack\_erasing
- stop-a (SPARC only)
- sysrq
- tainted

- threads-max
- traceoff\_on\_warning
- tracepoint\_printk
- unaligned-dump-stack (ia64)
- unaligned-trap
- unknown\_nmi\_panic
- unprivileged\_bpf\_disabled
- watchdog
- watchdog\_cpumask
- watchdog\_thresh

### acct

highwater lowwater frequency

If BSD-style process accounting is enabled these values control its behaviour. If free space on filesystem where the log lives goes below lowwater ``% accounting suspends. If free space gets above ``highwater``% accounting resumes. ``frequency determines how often do we check the amount of free space (value is in seconds). Default:

4 2 30

That is, suspend accounting if free space drops below 2%; resume it if it increases to at least 4%; consider information about amount of free space valid for 30 seconds.

### acpi\_video\_flags

See /power/video. This allows the video resume mode to be set, in a similar fashion to the acpi\_sleep kernel parameter, by combining the following values:

1	s3_bios
2	s3_mode
4	s3_beep

## auto\_msgmni

This variable has no effect and may be removed in future kernel releases. Reading it always returns 0. Up to Linux 3.17, it enabled/disabled automatic recomputing of msgmni upon memory add/remove or upon IPC namespace creation/removal. Echoing “1” into this file enabled msgmni automatic recomputing. Echoing “0” turned it off. The default value was 1.

## bootloader\_type (x86 only)

This gives the bootloader type number as indicated by the bootloader, shifted left by 4, and OR'd with the low four bits of the bootloader version. The reason for this encoding is that this used to match the `type_of_loader` field in the kernel header; the encoding is kept for backwards compatibility. That is, if the full bootloader type number is 0x15 and the full version number is 0x234, this file will contain the value  $340 = 0x154$ .

See the `type_of_loader` and `ext_loader_type` fields in `/x86/boot` for additional information.

## bootloader\_version (x86 only)

The complete bootloader version number. In the example above, this file will contain the value  $564 = 0x234$ .

See the `type_of_loader` and `ext_loader_ver` fields in `/x86/boot` for additional information.

## bpf\_stats\_enabled

Controls whether the kernel should collect statistics on BPF programs (total time spent running, number of times run...). Enabling statistics causes a slight reduction in performance on each program run. The statistics can be seen using `bpftool`.

0	Don't collect statistics (default).
1	Collect statistics.

## cad\_pid

This is the pid which will be signalled on reboot (notably, by Ctrl-Alt-Delete). Writing a value to this file which doesn't correspond to a running process will result in `-ESRCH`.

See also `ctrl-alt-del`.

### cap\_last\_cap

Highest valid capability of the running kernel. Exports CAP\_LAST\_CAP from the kernel.

### core\_pattern

core\_pattern is used to specify a core dumpfile pattern name.

- max length 127 characters; default value is “core”
- core\_pattern is used as a pattern template for the output filename; certain string patterns (beginning with ‘%’ ) are substituted with their actual values.
- backward compatibility with core\_uses\_pid:
  - If core\_pattern does not include “%p” (default does not) and core\_uses\_pid is set, then .PID will be appended to the filename.
- corename format specifiers

%<NUL>	‘%’ is dropped
%%	output one ‘%’
%p	pid
%P	global pid (init PID namespace)
%i	tid
%I	global tid (init PID namespace)
%u	uid (in initial user namespace)
%g	gid (in initial user namespace)
%d	dump mode, matches PR_SET_DUMPABLE and /proc/sys/fs/suid_dumpable
%s	signal number
%t	UNIX time of dump
%h	hostname
%e	executable filename (may be shortened)
%E	executable path
%c	maximum size of core file by resource limit RLIMIT_CORE
%<OTHER>	both are dropped

- If the first character of the pattern is a ‘|’ , the kernel will treat the rest of the pattern as a command to run. The core dump will be written to the standard input of that program instead of to a file.

### **core\_pipe\_limit**

This sysctl is only applicable when `core_pattern` is configured to pipe core files to a user space helper (when the first character of `core_pattern` is a '|', see above). When collecting cores via a pipe to an application, it is occasionally useful for the collecting application to gather data about the crashing process from its `/proc/pid` directory. In order to do this safely, the kernel must wait for the collecting process to exit, so as not to remove the crashing processes proc files prematurely. This in turn creates the possibility that a misbehaving userspace collecting process can block the reaping of a crashed process simply by never exiting. This sysctl defends against that. It defines how many concurrent crashing processes may be piped to user space applications in parallel. If this value is exceeded, then those crashing processes above that value are noted via the kernel log and their cores are skipped. 0 is a special value, indicating that unlimited processes may be captured in parallel, but that no waiting will take place (i.e. the collecting process is not guaranteed access to `/proc/<crashing pid>/`). This value defaults to 0.

### **core\_uses\_pid**

The default coredump filename is "core". By setting `core_uses_pid` to 1, the coredump filename becomes `core.PID`. If `core_pattern` does not include "%p" (default does not) and `core_uses_pid` is set, then `.PID` will be appended to the filename.

### **ctrl-alt-del**

When the value in this file is 0, `ctrl-alt-del` is trapped and sent to the `init(1)` program to handle a graceful restart. When, however, the value is > 0, Linux's reaction to a Vulcan Nerve Pinch (tm) will be an immediate reboot, without even syncing its dirty buffers.

**Note:** when a program (like `dosemu`) has the keyboard in 'raw' mode, the `ctrl-alt-del` is intercepted by the program before it ever reaches the kernel tty layer, and it's up to the program to decide what to do with it.

### **dmesg\_restrict**

This toggle indicates whether unprivileged users are prevented from using `dmesg(8)` to view messages from the kernel's log buffer. When `dmesg_restrict` is set to 0 there are no restrictions. When `dmesg_restrict` is set set to 1, users must have `CAP_SYSLOG` to use `dmesg(8)`.

The kernel config option `CONFIG_SECURITY_DMESG_RESTRICT` sets the default value of `dmesg_restrict`.

### domainname & hostname

These files can be used to set the NIS/YP domainname and the hostname of your box in exactly the same way as the commands domainname and hostname, i.e.:

```
# echo "darkstar" > /proc/sys/kernel/hostname
# echo "mydomain" > /proc/sys/kernel/domainname
```

has the same effect as:

```
# hostname "darkstar"
# domainname "mydomain"
```

Note, however, that the classic darkstar.frop.org has the hostname “darkstar” and DNS (Internet Domain Name Server) domainname “frop.org”, not to be confused with the NIS (Network Information Service) or YP (Yellow Pages) domainname. These two domain names are in general different. For a detailed discussion see the hostname(1) man page.

### firmware\_config

See /driver-api/firmware/fallback-mechanisms.

The entries in this directory allow the firmware loader helper fallback to be controlled:

- force\_sysfs\_fallback, when set to 1, forces the use of the fallback;
- ignore\_sysfs\_fallback, when set to 1, ignores any fallback.

### ftrace\_dump\_on\_oops

Determines whether ftrace\_dump() should be called on an oops (or kernel panic). This will output the contents of the ftrace buffers to the console. This is very useful for capturing traces that lead to crashes and outputting them to a serial console.

0	Disabled (default).
1	Dump buffers of all CPUs.
2	Dump the buffer of the CPU that triggered the oops.

### ftrace\_enabled, stack\_tracer\_enabled

See /trace/ftrace.

### hardlockup\_all\_cpu\_backtrace

This value controls the hard lockup detector behavior when a hard lockup condition is detected as to whether or not to gather further debug information. If enabled, arch-specific all-CPU stack dumping will be initiated.

0	Do nothing. This is the default behavior.
1	On detection capture more debug information.

### hardlockup\_panic

This parameter can be used to control whether the kernel panics when a hard lockup is detected.

0	Don' t panic on hard lockup.
1	Panic on hard lockup.

See Softlockup detector and hardlockup detector (aka nmi\_watchdog) for more information. This can also be set using the nmi\_watchdog kernel parameter.

### hotplug

Path for the hotplug policy agent. Default value is “/sbin/hotplug” .

### hung\_task\_all\_cpu\_backtrace:

If this option is set, the kernel will send an NMI to all CPUs to dump their backtraces when a hung task is detected. This file shows up if CONFIG\_DETECT\_HUNG\_TASK and CONFIG\_SMP are enabled.

0: Won' t show all CPUs backtraces when a hung task is detected. This is the default behavior.

1: Will non-maskably interrupt all CPUs and dump their backtraces when a hung task is detected.

### hung\_task\_panic

Controls the kernel' s behavior when a hung task is detected. This file shows up if CONFIG\_DETECT\_HUNG\_TASK is enabled.

0	Continue operation. This is the default behavior.
1	Panic immediately.

### hung\_task\_check\_count

The upper bound on the number of tasks that are checked. This file shows up if CONFIG\_DETECT\_HUNG\_TASK is enabled.

### hung\_task\_timeout\_secs

When a task in D state did not get scheduled for more than this value report a warning. This file shows up if CONFIG\_DETECT\_HUNG\_TASK is enabled.

0 means infinite timeout, no checking is done.

Possible values to set are in range {0:LONG\_MAX/HZ}.

### hung\_task\_check\_interval\_secs

Hung task check interval. If hung task checking is enabled (see hung\_task\_timeout\_secs), the check is done every hung\_task\_check\_interval\_secs seconds. This file shows up if CONFIG\_DETECT\_HUNG\_TASK is enabled.

0 (default) means use hung\_task\_timeout\_secs as checking interval.

Possible values to set are in range {0:LONG\_MAX/HZ}.

### hung\_task\_warnings

The maximum number of warnings to report. During a check interval if a hung task is detected, this value is decreased by 1. When this value reaches 0, no more warnings will be reported. This file shows up if CONFIG\_DETECT\_HUNG\_TASK is enabled.

-1: report an infinite number of warnings.

### hyperv\_record\_panic\_msg

Controls whether the panic kmsg data should be reported to Hyper-V.

0	Do not report panic kmsg data.
1	Report the panic kmsg data. This is the default behavior.

### ignore-unaligned-usertrap

On architectures where unaligned accesses cause traps, and where this feature is supported (`CONFIG_SYSCTL_ARCH_UNALIGN_NO_WARN`; currently, `arc` and `ia64`), controls whether all unaligned traps are logged.

0	Log all unaligned accesses.
1	Only warn the first time a process traps. This is the default setting.

See also `unaligned-trap` and `unaligned-dump-stack`. On `ia64`, this allows system administrators to override the `IA64_THREAD_UAC_NOPRINT` `prctl` and avoid logs being flooded.

### kexec\_load\_disabled

A toggle indicating if the `kexec_load` syscall has been disabled. This value defaults to 0 (false: `kexec_load` enabled), but can be set to 1 (true: `kexec_load` disabled). Once true, `kexec` can no longer be used, and the toggle cannot be set back to false. This allows a `kexec` image to be loaded before disabling the syscall, allowing a system to set up (and later use) an image without it being altered. Generally used together with the `modules_disabled` `sysctl`.

### kptr\_restrict

This toggle indicates whether restrictions are placed on exposing kernel addresses via `/proc` and other interfaces.

When `kptr_restrict` is set to 0 (the default) the address is hashed before printing. (This is the equivalent to `%p`.)

When `kptr_restrict` is set to 1, kernel pointers printed using the `%pK` format specifier will be replaced with 0s unless the user has `CAP_SYSLOG` and effective user and group ids are equal to the real ids. This is because `%pK` checks are done at `read()` time rather than `open()` time, so if permissions are elevated between the `open()` and the `read()` (e.g via a `setuid` binary) then `%pK` will not leak kernel pointers to unprivileged users. Note, this is a temporary solution only. The correct long-term solution is to do the permission checks at `open()` time. Consider removing world read permissions from files that use `%pK`, and using `dmesg_restrict` to protect against uses of `%pK` in `dmesg(8)` if leaking kernel pointer values to unprivileged users is a concern.

When `kptr_restrict` is set to 2, kernel pointers printed using `%pK` will be replaced with 0s regardless of privileges.

### modprobe

The full path to the usermode helper for autoloading kernel modules, by default “/sbin/modprobe” . This binary is executed when the kernel requests a module. For example, if userspace passes an unknown filesystem type to mount(), then the kernel will automatically request the corresponding filesystem module by executing this usermode helper. This usermode helper should insert the needed module into the kernel.

This sysctl only affects module autoloading. It has no effect on the ability to explicitly insert modules.

This sysctl can be used to debug module loading requests:

```
echo '#! /bin/sh' > /tmp/modprobe
echo 'echo "$@" >> /tmp/modprobe.log' >> /tmp/modprobe
echo 'exec /sbin/modprobe "$@"' >> /tmp/modprobe
chmod a+x /tmp/modprobe
echo /tmp/modprobe > /proc/sys/kernel/modprobe
```

Alternatively, if this sysctl is set to the empty string, then module autoloading is completely disabled. The kernel will not try to execute a usermode helper at all, nor will it call the kernel\_module\_request LSM hook.

If CONFIG\_STATIC\_USERMODEHELPER=y is set in the kernel configuration, then the configured static usermode helper overrides this sysctl, except that the empty string is still accepted to completely disable module autoloading as described above.

### modules\_disabled

A toggle value indicating if modules are allowed to be loaded in an otherwise modular kernel. This toggle defaults to off (0), but can be set true (1). Once true, modules can be neither loaded nor unloaded, and the toggle cannot be set back to false. Generally used with the kexec\_load\_disabled toggle.

### msgmax, msgmnb, and msgmni

msgmax is the maximum size of an IPC message, in bytes. 8192 by default (MSGMAX).

msgmnb is the maximum size of an IPC queue, in bytes. 16384 by default (MSGMNB).

msgmni is the maximum number of IPC queues. 32000 by default (MSGMNI).

## msg\_next\_id, sem\_next\_id, and shm\_next\_id (System V IPC)

These three toggles allows to specify desired id for next allocated IPC object: message, semaphore or shared memory respectively.

By default they are equal to -1, which means generic allocation logic. Possible values to set are in range {0:INT\_MAX}.

### Notes:

- 1) kernel doesn't guarantee, that new object will have desired id. So, it's up to userspace, how to handle an object with "wrong" id.
- 2) Toggle with non-default value will be set back to -1 by kernel after successful IPC object allocation. If an IPC object allocation syscall fails, it is undefined if the value remains unmodified or is reset to -1.

## ngroups\_max

Maximum number of supplementary groups, *i.e.* the maximum size which setgroups will accept. Exports NGROUPS\_MAX from the kernel.

## nmi\_watchdog

This parameter can be used to control the NMI watchdog (*i.e.* the hard lockup detector) on x86 systems.

0	Disable the hard lockup detector.
1	Enable the hard lockup detector.

The hard lockup detector monitors each CPU for its ability to respond to timer interrupts. The mechanism utilizes CPU performance counter registers that are programmed to generate Non-Maskable Interrupts (NMIs) periodically while a CPU is busy. Hence, the alternative name 'NMI watchdog'.

The NMI watchdog is disabled by default if the kernel is running as a guest in a KVM virtual machine. This default can be overridden by adding:

```
nmi_watchdog=1
```

to the guest kernel command line (see The kernel's command-line parameters).

## numa\_balancing

Enables/disables automatic page fault based NUMA memory balancing. Memory is moved automatically to nodes that access it often.

Enables/disables automatic NUMA memory balancing. On NUMA machines, there is a performance penalty if remote memory is accessed by a CPU. When this feature is enabled the kernel samples what task thread is accessing memory by periodically unmapping pages and later trapping a page fault. At the time of the page

fault, it is determined if the data being accessed should be migrated to a local memory node.

The unmapping of pages and trapping faults incur additional overhead that ideally is offset by improved memory locality but there is no universal guarantee. If the target workload is already bound to NUMA nodes then this feature should be disabled. Otherwise, if the system overhead from the feature is too high then the rate the kernel samples for NUMA hinting faults may be controlled by the `numa_balancing_scan_period_min_ms`, `numa_balancing_scan_delay_ms`, `numa_balancing_scan_period_max_ms`, `numa_balancing_scan_size_mb`, and `numa_balancing_settle_count` sysctls.

### **`numa_balancing_scan_period_min_ms`, `numa_balancing_scan_delay_ms`, `numa_balancing_scan_period_max_ms`, `numa_balancing_scan_size_mb`**

Automatic NUMA balancing scans tasks address space and unmaps pages to detect if pages are properly placed or if the data should be migrated to a memory node local to where the task is running. Every “scan delay” the task scans the next “scan size” number of pages in its address space. When the end of the address space is reached the scanner restarts from the beginning.

In combination, the “scan delay” and “scan size” determine the scan rate. When “scan delay” decreases, the scan rate increases. The scan delay and hence the scan rate of every task is adaptive and depends on historical behaviour. If pages are properly placed then the scan delay increases, otherwise the scan delay decreases. The “scan size” is not adaptive but the higher the “scan size”, the higher the scan rate.

Higher scan rates incur higher system overhead as page faults must be trapped and potentially data must be migrated. However, the higher the scan rate, the more quickly a tasks memory is migrated to a local node if the workload pattern changes and minimises performance impact due to remote memory accesses. These sysctls control the thresholds for scan delays and the number of pages scanned.

`numa_balancing_scan_period_min_ms` is the minimum time in milliseconds to scan a tasks virtual memory. It effectively controls the maximum scanning rate for each task.

`numa_balancing_scan_delay_ms` is the starting “scan delay” used for a task when it initially forks.

`numa_balancing_scan_period_max_ms` is the maximum time in milliseconds to scan a tasks virtual memory. It effectively controls the minimum scanning rate for each task.

`numa_balancing_scan_size_mb` is how many megabytes worth of pages are scanned for a given scan.

### **oops\_all\_cpu\_backtrace:**

If this option is set, the kernel will send an NMI to all CPUs to dump their backtraces when an oops event occurs. It should be used as a last resort in case a panic cannot be triggered (to protect VMs running, for example) or kdump can't be collected. This file shows up if CONFIG\_SMP is enabled.

0: Won't show all CPUs backtraces when an oops is detected. This is the default behavior.

1: Will non-maskably interrupt all CPUs and dump their backtraces when an oops event is detected.

### **osrelease, ostype & version**

```
# cat osrelease
2.1.88
# cat ostype
Linux
# cat version
#5 Wed Feb 25 21:49:24 MET 1998
```

The files `osrelease` and `ostype` should be clear enough. `version` needs a little more clarification however. The '#5' means that this is the fifth kernel built from this source base and the date behind it indicates the time the kernel was built. The only way to tune these values is to rebuild the kernel :-)

### **overflowgid & overflowuid**

if your architecture did not always support 32-bit UIDs (i.e. arm, i386, m68k, sh, and sparc32), a fixed UID and GID will be returned to applications that use the old 16-bit UID/GID system calls, if the actual UID or GID would exceed 65535.

These sysctls allow you to change the value of the fixed UID and GID. The default is 65534.

### **panic**

The value in this file determines the behaviour of the kernel on a panic:

- if zero, the kernel will loop forever;
- if negative, the kernel will reboot immediately;
- if positive, the kernel will reboot after the corresponding number of seconds.

When you use the software watchdog, the recommended setting is 60.

### panic\_on\_io\_nmi

Controls the kernel's behavior when a CPU receives an NMI caused by an IO error.

0	Try to continue operation (default).
1	Panic immediately. The IO error triggered an NMI. This indicates a serious system condition which could result in IO data corruption. Rather than continuing, panicking might be a better choice. Some servers issue this sort of NMI when the dump button is pushed, and you can use this option to take a crash dump.

### panic\_on\_oops

Controls the kernel's behaviour when an oops or BUG is encountered.

0	Try to continue operation.
1	Panic immediately. If the panic sysctl is also non-zero then the machine will be rebooted.

### panic\_on\_stackoverflow

Controls the kernel's behavior when detecting the overflows of kernel, IRQ and exception stacks except a user stack. This file shows up if CONFIG\_DEBUG\_STACKOVERFLOW is enabled.

0	Try to continue operation.
1	Panic immediately.

### panic\_on\_unrecovered\_nmi

The default Linux behaviour on an NMI of either memory or unknown is to continue operation. For many environments such as scientific computing it is preferable that the box is taken out and the error dealt with than an uncorrected parity/ECC error get propagated.

A small number of systems do generate NMIs for bizarre random reasons such as power management so the default is off. That sysctl works like the existing panic controls already in that directory.

### panic\_on\_warn

Calls panic() in the WARN() path when set to 1. This is useful to avoid a kernel rebuild when attempting to kdump at the location of a WARN().

0	Only WARN(), default behaviour.
1	Call panic() after printing out WARN() location.

### panic\_print

Bitmask for printing system info when panic happens. User can chose combination of the following bits:

bit 0	print all tasks info
bit 1	print system memory info
bit 2	print timer info
bit 3	print locks info if CONFIG_LOCKDEP is on
bit 4	print ftrace buffer

So for example to print tasks and memory info on panic, user can:

```
echo 3 > /proc/sys/kernel/panic_print
```

### panic\_on\_rcu\_stall

When set to 1, calls panic() after RCU stall detection messages. This is useful to define the root cause of RCU stalls using a vmcore.

0	Do not panic() when RCU stall takes place, default behavior.
1	panic() after printing RCU stall messages.

### perf\_cpu\_time\_max\_percent

Hints to the kernel how much CPU time it should be allowed to use to handle perf sampling events. If the perf subsystem is informed that its samples are exceeding this limit, it will drop its sampling frequency to attempt to reduce its CPU usage.

Some perf sampling happens in NMIs. If these samples unexpectedly take too long to execute, the NMIs can become stacked up next to each other so much that nothing else is allowed to execute.

0	Disable the mechanism. Do not monitor or correct perf's sampling rate no matter how CPU time it takes.
1-100	Attempt to throttle perf's sample rate to this percentage of CPU. Note: the kernel calculates an "expected" length of each sample event. 100 here means 100% of that expected length. Even if this is set to 100, you may still see sample throttling if this length is exceeded. Set to 0 if you truly do not care how much CPU is consumed.

### perf\_event\_paranoid

Controls use of the performance events system by unprivileged users (without CAP\_PERFMON). The default value is 2.

For backward compatibility reasons access to system performance monitoring and observability remains open for CAP\_SYS\_ADMIN privileged processes but CAP\_SYS\_ADMIN usage for secure system performance monitoring and observability operations is discouraged with respect to CAP\_PERFMON use cases.

-1	Allow use of (almost) all events by all users. Ignore mlock limit after perf_event_mlock_kb without CAP_IPC_LOCK.
>=0	Disallow ftrace function tracepoint by users without CAP_PERFMON. Disallow raw tracepoint access by users without CAP_PERFMON.
>=1	Disallow CPU event access by users without CAP_PERFMON.
>=2	Disallow kernel profiling by users without CAP_PERFMON.

### perf\_event\_max\_stack

Controls maximum number of stack frames to copy for (attr.sample\_type & PERF\_SAMPLE\_CALLCHAIN) configured events, for instance, when using 'perf record -g' or 'perf trace --call-graph fp' .

This can only be done when no events are in use that have callchains enabled, otherwise writing to this file will return -EBUSY.

The default value is 127.

### perf\_event\_mlock\_kb

Control size of per-cpu ring buffer not counted against mlock limit.

The default value is 512 + 1 page

### perf\_event\_max\_contexts\_per\_stack

Controls maximum number of stack frame context entries for (attr.sample\_type & PERF\_SAMPLE\_CALLCHAIN) configured events, for instance, when using 'perf record -g' or 'perf trace --call-graph fp' .

This can only be done when no events are in use that have callchains enabled, otherwise writing to this file will return -EBUSY.

The default value is 8.

## pid\_max

PID allocation wrap value. When the kernel's next PID value reaches this value, it wraps back to a minimum PID value. PIDs of value pid\_max or larger are not allocated.

## ns\_last\_pid

The last pid allocated in the current (the one task using this sysctl lives in) pid namespace. When selecting a pid for a next task on fork kernel tries to allocate a number starting from this one.

## powersave-nap (PPC only)

If set, Linux-PPC will use the 'nap' mode of powersaving, otherwise the 'doze' mode will be used.

---

## printk

The four values in printk denote: console\_loglevel, default\_message\_loglevel, minimum\_console\_loglevel and default\_console\_loglevel respectively.

These values influence printk() behavior when printing or logging error messages. See 'man 2 syslog' for more info on the different loglevels.

console_loglevel	messages with a higher priority than this will be printed to the console
de- fault_message_loglevel	messages without an explicit priority will be printed with this priority
mini- mum_console_loglevel	minimum (highest) value to which console_loglevel can be set
de- fault_console_loglevel	default value for console_loglevel

## printk\_delay

Delay each printk message in printk\_delay milliseconds

Value from 0 - 10000 is allowed.

### **printk\_ratelimit**

Some warning messages are rate limited. `printk_ratelimit` specifies the minimum length of time between these messages (in seconds). The default value is 5 seconds.

A value of 0 will disable rate limiting.

### **printk\_ratelimit\_burst**

While long term we enforce one message per `printk_ratelimit` seconds, we do allow a burst of messages to pass through. `printk_ratelimit_burst` specifies the number of messages we can send before ratelimiting kicks in.

The default value is 10 messages.

### **printk\_devkmsg**

Control the logging to `/dev/kmsg` from userspace:

<code>ratelimit</code>	default, ratelimited
<code>on</code>	unlimited logging to <code>/dev/kmsg</code> from userspace
<code>off</code>	logging to <code>/dev/kmsg</code> disabled

The kernel command line parameter `printk.devkmsg=` overrides this and is a one-time setting until next reboot: once set, it cannot be changed by this `sysctl` interface anymore.

---

### **pty**

See [Documentation/filesystems/devpts.rst](#).

### **randomize\_va\_space**

This option can be used to select the type of process address space randomization that is used in the system, for architectures that support this feature.

0	Turn the process address space randomization off. This is the default for architectures that do not support this feature anyways, and kernels that are booted with the “norandmaps” parameter.
1	Make the addresses of mmap base, stack and VDSO page randomized. This, among other things, implies that shared libraries will be loaded to random addresses. Also for PIE-linked binaries, the location of code start is randomized. This is the default if the CONFIG_COMPAT_BRK option is enabled.
2	<p>Additionally enable heap randomization. This is the default if CONFIG_COMPAT_BRK is disabled.</p> <p>There are a few legacy applications out there (such as some ancient versions of libc.so.5 from 1996) that assume that brk area starts just after the end of the code+bss. These applications break when start of the brk area is randomized. There are however no known non-legacy applications that would be broken this way, so for most systems it is safe to choose full randomization.</p> <p>Systems with ancient and/or broken binaries should be configured with CONFIG_COMPAT_BRK enabled, which excludes the heap from process address space randomization.</p>

### real-root-dev

See Using the initial RAM disk (initrd).

### reboot-cmd (SPARC only)

??? This seems to be a way to give an argument to the Sparc ROM/Flash boot loader. Maybe to tell it what to do after rebooting. ???

### sched\_energy\_aware

Enables/disables Energy Aware Scheduling (EAS). EAS starts automatically on platforms where it can run (that is, platforms with asymmetric CPU topologies and having an Energy Model available). If your platform happens to meet the requirements for EAS but you do not want to use it, change this value to 0.

### sched\_schedstats

Enables/disables scheduler statistics. Enabling this feature incurs a small amount of overhead in the scheduler but is useful for debugging and performance tuning.

### seccomp

See `/userspace-api/seccomp_filter`.

### sg-big-buff

This file shows the size of the generic SCSI (sg) buffer. You can't tune it just yet, but you could change it on compile time by editing `include/scsi/sg.h` and changing the value of `SG_BIG_BUFF`.

There shouldn't be any reason to change this value. If you can come up with one, you probably know what you are doing anyway :)

### shmall

This parameter sets the total amount of shared memory pages that can be used system wide. Hence, `shmall` should always be at least `ceil(shmmax/PAGE_SIZE)`.

If you are not sure what the default `PAGE_SIZE` is on your Linux system, you can run the following command:

```
# getconf PAGE_SIZE
```

### shmmax

This value can be used to query and set the run time limit on the maximum shared memory segment size that can be created. Shared memory segments up to 1Gb are now supported in the kernel. This value defaults to `SHMMAX`.

### shmmni

This value determines the maximum number of shared memory segments. 4096 by default (`SHMMNI`).

### shm\_rmid\_forced

Linux lets you set resource limits, including how much memory one process can consume, via `setrlimit(2)`. Unfortunately, shared memory segments are allowed to exist without association with any process, and thus might not be counted against any resource limits. If enabled, shared memory segments are automatically destroyed when their attach count becomes zero after a detach or a process termination. It will also destroy segments that were created, but never attached to, on exit from the process. The only use left for `IPC_RMID` is to immediately destroy an unattached segment. Of course, this breaks the way things are defined, so some applications might stop working. Note that this feature will do you no good unless you also configure your resource limits (in particular, `RLIMIT_AS` and `RLIMIT_NPROC`). Most systems don't need this.

Note that if you change this from 0 to 1, already created segments without users and with a dead originative process will be destroyed.

### sysctl\_writes\_strict

Control how file position affects the behavior of updating sysctl values via the /proc/sys interface:

-1	Legacy per-write sysctl value handling, with no printk warnings. Each write syscall must fully contain the sysctl value to be written, and multiple writes on the same sysctl file descriptor will rewrite the sysctl value, regardless of file position.
0	Same behavior as above, but warn about processes that perform writes to a sysctl file descriptor when the file position is not 0.
1	(default) Respect file position when writing sysctl strings. Multiple writes will append to the sysctl value buffer. Anything past the max length of the sysctl value buffer will be ignored. Writes to numeric sysctl entries must always be at file position 0 and the value must be fully contained in the buffer sent in the write syscall.

### softlockup\_all\_cpu\_backtrace

This value controls the soft lockup detector thread's behavior when a soft lockup condition is detected as to whether or not to gather further debug information. If enabled, each cpu will be issued an NMI and instructed to capture stack trace.

This feature is only applicable for architectures which support NMI.

0	Do nothing. This is the default behavior.
1	On detection capture more debug information.

### softlockup\_panic

This parameter can be used to control whether the kernel panics when a soft lockup is detected.

0	Don't panic on soft lockup.
1	Panic on soft lockup.

This can also be set using the softlockup\_panic kernel parameter.

### soft\_watchdog

This parameter can be used to control the soft lockup detector.

0	Disable the soft lockup detector.
1	Enable the soft lockup detector.

The soft lockup detector monitors CPUs for threads that are hogging the CPUs without rescheduling voluntarily, and thus prevent the 'watchdog/N' threads from

running. The mechanism depends on the CPUs ability to respond to timer interrupts which are needed for the ‘watchdog/N’ threads to be woken up by the watchdog timer function, otherwise the NMI watchdog –if enabled –can detect a hard lockup condition.

### stack\_erasing

This parameter can be used to control kernel stack erasing at the end of syscalls for kernels built with CONFIG\_GCC\_PLUGIN\_STACKLEAK.

That erasing reduces the information which kernel stack leak bugs can reveal and blocks some uninitialized stack variable attacks. The tradeoff is the performance impact: on a single CPU system kernel compilation sees a 1% slowdown, other systems and workloads may vary.

0	Kernel stack erasing is disabled, STACKLEAK_METRICS are not updated.
1	Kernel stack erasing is enabled (default), it is performed before returning to the userspace at the end of syscalls.

### stop-a (SPARC only)

Controls Stop-A:

0	Stop-A has no effect.
1	Stop-A breaks to the PROM (default).

Stop-A is always enabled on a panic, so that the user can return to the boot PROM.

### sysrq

See Linux Magic System Request Key Hacks.

### tainted

Non-zero if the kernel has been tainted. Numeric values, which can be ORed together. The letters are seen in “Tainted” line of Oops reports.

1	(P)	proprietary module was loaded
2	(F)	module was force loaded
4	(S)	SMP kernel oops on an officially SMP incapable processor
8	(R)	module was force unloaded
16	(M)	processor reported a Machine Check Exception (MCE)
32	(B)	bad page referenced or some unexpected page flags
64	(U)	taint requested by userspace application
128	(D)	kernel died recently, i.e. there was an OOPS or BUG
256	(A)	an ACPI table was overridden by user
512	(W)	kernel issued warning
1024	(C)	staging driver was loaded
2048	(I)	workaround for bug in platform firmware applied
4096	(O)	externally-built ( "out-of-tree" ) module was loaded
8192	(E)	unsigned module was loaded
16384	(L)	soft lockup occurred
32768	(K)	kernel has been live patched
65536	(X)	Auxiliary taint, defined and used by for distros
131072	(T)	The kernel was built with the struct randomization plugin

See Tainted kernels for more information.

**Note:** writes to this sysctl interface will fail with EINVAL if the kernel is booted with the command line option `panic_on_taint=<bitmask>`, `nousertaint` and any of the ORed together values being written to `tainted` match with the bitmask declared on `panic_on_taint`. See The kernel' s command-line parameters for more details on that particular kernel command line option and its optional `nousertaint` switch.

### threads-max

This value controls the maximum number of threads that can be created using `fork()`.

During initialization the kernel sets this value such that even if the maximum number of threads is created, the thread structures occupy only a part (1/8th) of the available RAM pages.

The minimum value that can be written to `threads-max` is 1.

The maximum value that can be written to `threads-max` is given by the constant `FUTEX_TID_MASK` (0x3fffffff).

If a value outside of this range is written to `threads-max` an EINVAL error occurs.

### traceoff\_on\_warning

When set, disables tracing (see /trace/ftrace) when a WARN( ) is hit.

### tracepoint\_printk

When tracepoints are sent to printk() (enabled by the tp\_printk boot parameter), this entry provides runtime control:

```
echo 0 > /proc/sys/kernel/tracepoint_printk
```

will stop tracepoints from being sent to printk(), and:

```
echo 1 > /proc/sys/kernel/tracepoint_printk
```

will send them to printk() again.

This only works if the kernel was booted with tp\_printk enabled.

See The kernel' s command-line parameters and /trace/boottime-trace.

### unaligned-dump-stack (ia64)

When logging unaligned accesses, controls whether the stack is dumped.

0	Do not dump the stack. This is the default setting.
1	Dump the stack.

See also ignore-unaligned-usertrap.

### unaligned-trap

On architectures where unaligned accesses cause traps, and where this feature is supported (CONFIG\_SYSCTL\_ARCH\_UNALIGN\_ALLOW; currently, arc and parisc), controls whether unaligned traps are caught and emulated (instead of failing).

0	Do not emulate unaligned accesses.
1	Emulate unaligned accesses. This is the default setting.

See also ignore-unaligned-usertrap.

### unknown\_nmi\_panic

The value in this file affects behavior of handling NMI. When the value is non-zero, unknown NMI is trapped and then panic occurs. At that time, kernel debugging information is displayed on console.

NMI switch that most IA32 servers have fires unknown NMI up, for example. If a system hangs up, try pressing the NMI switch.

### unprivileged\_bpf\_disabled

Writing 1 to this entry will disable unprivileged calls to `bpf()`; once disabled, calling `bpf()` without `CAP_SYS_ADMIN` will return `-EPERM`.

Once set, this can't be cleared.

### watchdog

This parameter can be used to disable or enable the soft lockup detector and the NMI watchdog (i.e. the hard lockup detector) at the same time.

0	Disable both lockup detectors.
1	Enable both lockup detectors.

The soft lockup detector and the NMI watchdog can also be disabled or enabled individually, using the `soft_watchdog` and `nmi_watchdog` parameters. If the `watchdog` parameter is read, for example by executing:

```
cat /proc/sys/kernel/watchdog
```

the output of this command (0 or 1) shows the logical OR of `soft_watchdog` and `nmi_watchdog`.

### watchdog\_cpumask

This value can be used to control on which cpus the watchdog may run. The default cpumask is all possible cores, but if `NO_HZ_FULL` is enabled in the kernel config, and cores are specified with the `nohz_full=boot` argument, those cores are excluded by default. Offline cores can be included in this mask, and if the core is later brought online, the watchdog will be started based on the mask value.

Typically this value would only be touched in the `nohz_full` case to re-enable cores that by default were not running the watchdog, if a kernel lockup was suspected on those cores.

The argument value is the standard cpulist format for cpumasks, so for example to enable the watchdog on cores 0, 2, 3, and 4 you might say:

```
echo 0,2-4 > /proc/sys/kernel/watchdog_cpumask
```

### watchdog\_thresh

This value can be used to control the frequency of hrtimer and NMI events and the soft and hard lockup thresholds. The default threshold is 10 seconds.

The softlockup threshold is (2 \* watchdog\_thresh). Setting this tunable to zero will disable lockup detection altogether.

#### 4.1.4 Documentation for /proc/sys/net/

Copyright

Copyright (c) 1999

- Terrehon Bowden <terrehon@pacbell.net>
- Bodo Bauer <bb@ricochet.net>

Copyright (c) 2000

- Jorge Nerin <comandante@zaralinux.com>

Copyright (c) 2009

- Shen Feng <shen@cn.fujitsu.com>

For general info and legal blurb, please look in index.rst.

---

This file contains the documentation for the sysctl files in /proc/sys/net

The interface to the networking parts of the kernel is located in /proc/sys/net. The following table shows all possible subdirectories. You may see only some of them, depending on your kernel's configuration.

Table : Subdirectories in /proc/sys/net

Directory	Content	Directory	Content
core	General parameter	appletalk	Appletalk protocol
unix	Unix domain sockets	netrom	NET/ROM
802	E802 protocol	ax25	AX25
ethernet	Ethernet protocol	rose	X.25 PLP layer
ipv4	IP version 4	x25	X.25 protocol
bridge	Bridging	decnet	DEC net
ipv6	IP version 6	tipc	TIPC

## 1. /proc/sys/net/core - Network core options

### **bpf\_jit\_enable**

This enables the BPF Just in Time (JIT) compiler. BPF is a flexible and efficient infrastructure allowing to execute bytecode at various hook points. It is used in a number of Linux kernel subsystems such as networking (e.g. XDP, tc), tracing (e.g. kprobes, uprobes, tracepoints) and security (e.g. seccomp). LLVM has a BPF back end that can compile restricted C into a sequence of BPF instructions. After program load through `bpf(2)` and passing a verifier in the kernel, a JIT will then translate these BPF proglets into native CPU instructions. There are two flavors of JITs, the newer eBPF JIT currently supported on:

- x86\_64
- x86\_32
- arm64
- arm32
- ppc64
- sparc64
- mips64
- s390x
- riscv64
- riscv32

And the older cBPF JIT supported on the following archs:

- mips
- ppc
- sparc

eBPF JITs are a superset of cBPF JITs, meaning the kernel will migrate cBPF instructions into eBPF instructions and then JIT compile them transparently. Older cBPF JITs can only translate `tcpdump` filters, `seccomp` rules, etc, but not mentioned eBPF programs loaded through `bpf(2)`.

Values:

- 0 - disable the JIT (default value)
- 1 - enable the JIT
- 2 - enable the JIT and ask the compiler to emit traces on kernel log.

### **bpf\_jit\_harden**

This enables hardening for the BPF JIT compiler. Supported are eBPF JIT backends. Enabling hardening trades off performance, but can mitigate JIT spraying.

Values:

- 0 - disable JIT hardening (default value)
- 1 - enable JIT hardening for unprivileged users only
- 2 - enable JIT hardening for all users

### **bpf\_jit\_kallsyms**

When BPF JIT compiler is enabled, then compiled images are unknown addresses to the kernel, meaning they neither show up in traces nor in `/proc/kallsyms`. This enables export of these addresses, which can be used for debugging/tracing. If `bpf_jit_harden` is enabled, this feature is disabled.

Values :

- 0 - disable JIT kallsyms export (default value)
- 1 - enable JIT kallsyms export for privileged users only

### **bpf\_jit\_limit**

This enforces a global limit for memory allocations to the BPF JIT compiler in order to reject unprivileged JIT requests once it has been surpassed. `bpf_jit_limit` contains the value of the global limit in bytes.

### **dev\_weight**

The maximum number of packets that kernel can handle on a NAPI interrupt, it's a Per-CPU variable. For drivers that support LRO or GRO\_HW, a hardware aggregated packet is counted as one packet in this context.

Default: 64

### **dev\_weight\_rx\_bias**

RPS (e.g. RFS, aRFS) processing is competing with the registered NAPI poll function of the driver for the per softirq cycle `netdev_budget`. This parameter influences the proportion of the configured `netdev_budget` that is spent on RPS based packet processing during RX softirq cycles. It is further meant for making current `dev_weight` adaptable for asymmetric CPU needs on RX/TX side of the network stack. (see `dev_weight_tx_bias`) It is effective on a per CPU basis. Determination is based on `dev_weight` and is calculated multiplicative (`dev_weight * dev_weight_rx_bias`).

Default: 1

### **dev\_weight\_tx\_bias**

Scales the maximum number of packets that can be processed during a TX softirq cycle. Effective on a per CPU basis. Allows scaling of current dev\_weight for asymmetric net stack processing needs. Be careful to avoid making TX softirq processing a CPU hog.

Calculation is based on dev\_weight ( $\text{dev\_weight} * \text{dev\_weight\_tx\_bias}$ ).

Default: 1

### **default\_qdisc**

The default queuing discipline to use for network devices. This allows overriding the default of pfifo\_fast with an alternative. Since the default queuing discipline is created without additional parameters so is best suited to queuing disciplines that work well without configuration like stochastic fair queue (sfq), CoDel (codell) or fair queue CoDel (fq\_codel). Don't use queuing disciplines like Hierarchical Token Bucket or Deficit Round Robin which require setting up classes and bandwidths. Note that physical multiqueue interfaces still use mq as root qdisc, which in turn uses this default for its leaves. Virtual devices (like e.g. lo or veth) ignore this setting and instead default to noqueue.

Default: pfifo\_fast

### **busy\_read**

Low latency busy poll timeout for socket reads. (needs CONFIG\_NET\_RX\_BUSY\_POLL) Approximate time in us to busy loop waiting for packets on the device queue. This sets the default value of the SO\_BUSY\_POLL socket option. Can be set or overridden per socket by setting socket option SO\_BUSY\_POLL, which is the preferred method of enabling. If you need to enable the feature globally via sysctl, a value of 50 is recommended.

Will increase power usage.

Default: 0 (off)

### **busy\_poll**

Low latency busy poll timeout for poll and select. (needs CONFIG\_NET\_RX\_BUSY\_POLL) Approximate time in us to busy loop waiting for events. Recommended value depends on the number of sockets you poll on. For several sockets 50, for several hundreds 100. For more than that you probably want to use epoll. Note that only sockets with SO\_BUSY\_POLL set will be busy polled, so you want to either selectively set SO\_BUSY\_POLL on those sockets or set sysctl.net.busy\_read globally.

Will increase power usage.

Default: 0 (off)

### **rmem\_default**

The default setting of the socket receive buffer in bytes.

### **rmem\_max**

The maximum receive socket buffer size in bytes.

### **tstamp\_allow\_data**

Allow processes to receive tx timestamps looped together with the original packet contents. If disabled, transmit timestamp requests from unprivileged processes are dropped unless socket option `SOF_TIMESTAMPING_OPT_TSONLY` is set.

Default: 1 (on)

### **wmem\_default**

The default setting (in bytes) of the socket send buffer.

### **wmem\_max**

The maximum send socket buffer size in bytes.

### **message\_burst and message\_cost**

These parameters are used to limit the warning messages written to the kernel log from the networking code. They enforce a rate limit to make a denial-of-service attack impossible. A higher `message_cost` factor, results in fewer messages that will be written. `Message_burst` controls when messages will be dropped. The default settings limit warning messages to one every five seconds.

### **warnings**

This `sysctl` is now unused.

This was used to control console messages from the networking stack that occur because of problems on the network like duplicate address or bad checksums.

These messages are now emitted at `KERN_DEBUG` and can generally be enabled and controlled by the `dynamic_debug` facility.

## netdev\_budget

Maximum number of packets taken from all interfaces in one polling cycle (NAPI poll). In one polling cycle interfaces which are registered to polling are probed in a round-robin manner. Also, a polling cycle may not exceed `netdev_budget_usecs` microseconds, even if `netdev_budget` has not been exhausted.

## netdev\_budget\_usecs

Maximum number of microseconds in one NAPI polling cycle. Polling will exit when either `netdev_budget_usecs` have elapsed during the poll cycle or the number of packets processed reaches `netdev_budget`.

## netdev\_max\_backlog

Maximum number of packets, queued on the INPUT side, when the interface receives packets faster than kernel can process them.

## netdev\_rss\_key

RSS (Receive Side Scaling) enabled drivers use a 40 bytes host key that is randomly generated. Some user space might need to gather its content even if drivers do not provide `ethtool -x` support yet.

```
myhost:~# cat /proc/sys/net/core/netdev_rss_key
84:50:f4:00:a8:15:d1:a7:e9:7f:1d:60:35:c7:47:25:42:97:74:ca:56:bb:b6:a1:d8:
↪... (52 bytes total)
```

File contains nul bytes if no driver ever called `netdev_rss_key_fill()` function.

**Note:** `/proc/sys/net/core/netdev_rss_key` contains 52 bytes of key, but most drivers only use 40 bytes of it.

```
myhost:~# ethtool -x eth0
RX flow hash indirection table for eth0 with 8 RX ring(s):
  0:  0   1   2   3   4   5   6   7
RSS hash key:
84:50:f4:00:a8:15:d1:a7:e9:7f:1d:60:35:c7:47:25:42:97:74:ca:56:bb:b6:a1:d8:43:e3:c9:0c:
```

## netdev\_tstamp\_prequeue

If set to 0, RX packet timestamps can be sampled after RPS processing, when the target CPU processes packets. It might give some delay on timestamps, but permit to distribute the load on several cpus.

If set to 1 (default), timestamps are sampled as soon as possible, before queueing.

### **optmem\_max**

Maximum ancillary buffer size allowed per socket. Ancillary data is a sequence of struct cmsghdr structures with appended data.

### **fb\_tunnels\_only\_for\_init\_net**

Controls if fallback tunnels (like tunl0, gre0, gretap0, erspan0, sit0, ip6tnl0, ip6gre0) are automatically created when a new network namespace is created, if corresponding tunnel is present in initial network namespace. If set to 1, these devices are not automatically created, and user space is responsible for creating them if needed.

Default : 0 (for compatibility reasons)

### **devconf\_inherit\_init\_net**

Controls if a new network namespace should inherit all current settings under /proc/sys/net/{ipv4,ipv6}/conf/{all,default}/. By default, we keep the current behavior: for IPv4 we inherit all current settings from init\_net and for IPv6 we reset all settings to default.

If set to 1, both IPv4 and IPv6 settings are forced to inherit from current ones in init\_net. If set to 2, both IPv4 and IPv6 settings are forced to reset to their default values. If set to 3, both IPv4 and IPv6 settings are forced to inherit from current ones in the netns where this new netns has been created.

Default : 0 (for compatibility reasons)

## **2. /proc/sys/net/unix - Parameters for Unix domain sockets**

There is only one file in this directory. unix\_dgram\_qlen limits the max number of datagrams queued in Unix domain socket's buffer. It will not take effect unless PF\_UNIX flag is specified.

## **3. /proc/sys/net/ipv4 - IPV4 settings**

Please see: Documentation/networking/ip-sysctl.rst and Documentation/admin-guide/sysctl/net.rst for descriptions of these entries.

## 4. Appletalk

The `/proc/sys/net/appletalk` directory holds the Appletalk configuration data when Appletalk is loaded. The configurable parameters are:

### **aarp-expiry-time**

The amount of time we keep an ARP entry before expiring it. Used to age out old hosts.

### **aarp-resolve-time**

The amount of time we will spend trying to resolve an Appletalk address.

### **aarp-retransmit-limit**

The number of times we will retransmit a query before giving up.

### **aarp-tick-time**

Controls the rate at which expires are checked.

The directory `/proc/net/appletalk` holds the list of active Appletalk sockets on a machine.

The fields indicate the DDP type, the local address (in `network:node` format) the remote address, the size of the transmit pending queue, the size of the received queue (bytes waiting for applications to read) the state and the uid owning the socket.

`/proc/net/atalk_iface` lists all the interfaces configured for appletalk. It shows the name of the interface, its Appletalk address, the network range on that address (or network number for phase 1 networks), and the status of the interface.

`/proc/net/atalk_route` lists each known network route. It lists the target (network) that the route leads to, the router (may be directly connected), the route flags, and the device the route is using.

## 5. TIPC

### **tipc\_rmem**

The TIPC protocol now has a tunable for the receive memory, similar to the `tcp_rmem` - i.e. a vector of 3 INTEGERS: (min, default, max)

```
# cat /proc/sys/net/tipc/tipc_rmem
4252725 34021800      68043600
#
```

The max value is set to `CONN_OVERLOAD_LIMIT`, and the default and min values are scaled (shifted) versions of that same value. Note that the min value is not at this point in time used in any meaningful way, but the triplet is preserved in order to be consistent with things like `tcp_rmem`.

### **named\_timeout**

TIPC name table updates are distributed asynchronously in a cluster, without any form of transaction handling. This means that different race scenarios are possible. One such is that a name withdrawal sent out by one node and received by another node may arrive after a second, overlapping name publication already has been accepted from a third node, although the conflicting updates originally may have been issued in the correct sequential order. If `named_timeout` is nonzero, failed topology updates will be placed on a defer queue until another event arrives that clears the error, or until the timeout expires. Value is in milliseconds.

### **4.1.5 Documentation for /proc/sys/sunrpc/**

kernel version 2.2.10

Copyright (c) 1998, 1999, Rik van Riel <[riel@nl.linux.org](mailto:riel@nl.linux.org)>

For general info and legal blurb, please look in `index.rst`.

---

This file contains the documentation for the `sysctl` files in `/proc/sys/sunrpc` and is valid for Linux kernel version 2.2.

The files in this directory can be used to (re)set the debug flags of the SUN Remote Procedure Call (RPC) subsystem in the Linux kernel. This stuff is used for NFS, KNFSD and maybe a few other things as well.

The files in there are used to control the debugging flags: `rpc_debug`, `nfs_debug`, `nfsd_debug` and `nlm_debug`.

These flags are for kernel hackers only. You should read the source code in `net/sunrpc/` for more information.

### **4.1.6 Documentation for /proc/sys/user/**

kernel version 4.9.0

Copyright (c) 2016 Eric Biederman <[ebiederm@xmission.com](mailto:ebiederm@xmission.com)>

---

This file contains the documentation for the `sysctl` files in `/proc/sys/user`.

The files in this directory can be used to override the default limits on the number of namespaces and other objects that have per user per user namespace limits.

The primary purpose of these limits is to stop programs that malfunction and attempt to create a ridiculous number of objects, before the malfunction becomes a

system wide problem. It is the intention that the defaults of these limits are set high enough that no program in normal operation should run into these limits.

The creation of per user per user namespace objects are charged to the user in the user namespace who created the object and verified to be below the per user limit in that user namespace.

The creation of objects is also charged to all of the users who created user namespaces the creation of the object happens in (user namespaces can be nested) and verified to be below the per user limits in the user namespaces of those users.

This recursive counting of created objects ensures that creating a user namespace does not allow a user to escape their current limits.

Currently, these files are in `/proc/sys/user`:

### **max\_cgroup\_namespaces**

The maximum number of cgroup namespaces that any user in the current user namespace may create.

### **max\_ipc\_namespaces**

The maximum number of ipc namespaces that any user in the current user namespace may create.

### **max\_mnt\_namespaces**

The maximum number of mount namespaces that any user in the current user namespace may create.

### **max\_net\_namespaces**

The maximum number of network namespaces that any user in the current user namespace may create.

### **max\_pid\_namespaces**

The maximum number of pid namespaces that any user in the current user namespace may create.

### max\_time\_namespaces

The maximum number of time namespaces that any user in the current user namespace may create.

### max\_user\_namespaces

The maximum number of user namespaces that any user in the current user namespace may create.

### max\_uts\_namespaces

The maximum number of user namespaces that any user in the current user namespace may create.

## 4.1.7 Documentation for /proc/sys/vm/

kernel version 2.6.29

Copyright (c) 1998, 1999, Rik van Riel <[riel@nl.linux.org](mailto:riel@nl.linux.org)>

Copyright (c) 2008 Peter W. Morreale <[pmorreale@novell.com](mailto:pmorreale@novell.com)>

For general info and legal blurb, please look in index.rst.

---

This file contains the documentation for the sysctl files in /proc/sys/vm and is valid for Linux kernel version 2.6.29.

The files in this directory can be used to tune the operation of the virtual memory (VM) subsystem of the Linux kernel and the writeout of dirty data to disk.

Default values and initialization routines for most of these files can be found in mm/swap.c.

Currently, these files are in /proc/sys/vm:

- admin\_reserve\_kbytes
- block\_dump
- compact\_memory
- compact\_unevictable\_allowed
- dirty\_background\_bytes
- dirty\_background\_ratio
- dirty\_bytes
- dirty\_expire\_centisecs
- dirty\_ratio
- dirtytime\_expire\_seconds
- dirty\_writeback\_centisecs

- drop\_caches
- extfrag\_threshold
- hugetlb\_shm\_group
- laptop\_mode
- legacy\_va\_layout
- lowmem\_reserve\_ratio
- max\_map\_count
- memory\_failure\_early\_kill
- memory\_failure\_recovery
- min\_free\_kbytes
- min\_slab\_ratio
- min\_unmapped\_ratio
- mmap\_min\_addr
- mmap\_rnd\_bits
- mmap\_rnd\_compat\_bits
- nr\_hugepages
- nr\_hugepages\_mempolicy
- nr\_overcommit\_hugepages
- nr\_trim\_pages (only if CONFIG\_MMU=n)
- numa\_zonelist\_order
- oom\_dump\_tasks
- oom\_kill\_allocating\_task
- overcommit\_kbytes
- overcommit\_memory
- overcommit\_ratio
- page-cluster
- panic\_on\_oom
- percpu\_pagelist\_fraction
- stat\_interval
- stat\_refresh
- numa\_stat
- swappiness
- unprivileged\_userfaultfd
- user\_reserve\_kbytes

- `vfs_cache_pressure`
- `watermark_boost_factor`
- `watermark_scale_factor`
- `zone_reclaim_mode`

### **admin\_reserve\_kbytes**

The amount of free memory in the system that should be reserved for users with the capability `cap_sys_admin`.

`admin_reserve_kbytes` defaults to `min(3% of free pages, 8MB)`

That should provide enough for the admin to log in and kill a process, if necessary, under the default overcommit 'guess' mode.

Systems running under overcommit 'never' should increase this to account for the full Virtual Memory Size of programs used to recover. Otherwise, root may not be able to log in to recover the system.

How do you calculate a minimum useful reserve?

`sshd` or `login` + `bash` (or some other shell) + `top` (or `ps`, `kill`, etc.)

For overcommit 'guess', we can sum resident set sizes (RSS). On `x86_64` this is about 8MB.

For overcommit 'never', we can take the max of their virtual sizes (VSZ) and add the sum of their RSS. On `x86_64` this is about 128MB.

Changing this takes effect whenever an application requests memory.

### **block\_dump**

`block_dump` enables block I/O debugging when set to a nonzero value. More information on block I/O debugging is in `Documentation/admin-guide/laptops/laptop-mode.rst`.

### **compact\_memory**

Available only when `CONFIG_COMPACTION` is set. When 1 is written to the file, all zones are compacted such that free memory is available in contiguous blocks where possible. This can be important for example in the allocation of huge pages although processes will also directly compact memory as required.

### **compact\_unevictable\_allowed**

Available only when CONFIG\_COMPACTION is set. When set to 1, compaction is allowed to examine the unevictable lru (mlocked pages) for pages to compact. This should be used on systems where stalls for minor page faults are an acceptable trade for large contiguous free memory. Set to 0 to prevent compaction from moving pages that are unevictable. Default value is 1. On CONFIG\_PREEMPT\_RT the default value is 0 in order to avoid a page fault, due to compaction, which would block the task from becoming active until the fault is resolved.

### **dirty\_background\_bytes**

Contains the amount of dirty memory at which the background kernel flusher threads will start writeback.

**Note:** dirty\_background\_bytes is the counterpart of dirty\_background\_ratio. Only one of them may be specified at a time. When one sysctl is written it is immediately taken into account to evaluate the dirty memory limits and the other appears as 0 when read.

### **dirty\_background\_ratio**

Contains, as a percentage of total available memory that contains free pages and reclaimable pages, the number of pages at which the background kernel flusher threads will start writing out dirty data.

The total available memory is not equal to total system memory.

### **dirty\_bytes**

Contains the amount of dirty memory at which a process generating disk writes will itself start writeback.

Note: dirty\_bytes is the counterpart of dirty\_ratio. Only one of them may be specified at a time. When one sysctl is written it is immediately taken into account to evaluate the dirty memory limits and the other appears as 0 when read.

Note: the minimum value allowed for dirty\_bytes is two pages (in bytes); any value lower than this limit will be ignored and the old configuration will be retained.

### **dirty\_expire\_centisecs**

This tunable is used to define when dirty data is old enough to be eligible for writeout by the kernel flusher threads. It is expressed in 100' ths of a second. Data which has been dirty in-memory for longer than this interval will be written out next time a flusher thread wakes up.

### dirty\_ratio

Contains, as a percentage of total available memory that contains free pages and reclaimable pages, the number of pages at which a process which is generating disk writes will itself start writing out dirty data.

The total available memory is not equal to total system memory.

### dirtytime\_expire\_seconds

When a lazytime inode is constantly having its pages dirtied, the inode with an updated timestamp will never get chance to be written out. And, if the only thing that has happened on the file system is a dirtytime inode caused by an atime update, a worker will be scheduled to make sure that inode eventually gets pushed out to disk. This tunable is used to define when dirty inode is old enough to be eligible for writeback by the kernel flusher threads. And, it is also used as the interval to wakeup dirtytime\_writeback thread.

### dirty\_writeback\_centisecs

The kernel flusher threads will periodically wake up and write old data out to disk. This tunable expresses the interval between those wakeups, in 100'ths of a second.

Setting this to zero disables periodic writeback altogether.

### drop\_caches

Writing to this will cause the kernel to drop clean caches, as well as reclaimable slab objects like dentries and inodes. Once dropped, their memory becomes free.

To free pagecache:

```
echo 1 > /proc/sys/vm/drop_caches
```

To free reclaimable slab objects (includes dentries and inodes):

```
echo 2 > /proc/sys/vm/drop_caches
```

To free slab objects and pagecache:

```
echo 3 > /proc/sys/vm/drop_caches
```

This is a non-destructive operation and will not free any dirty objects. To increase the number of objects freed by this operation, the user may run sync prior to writing to /proc/sys/vm/drop\_caches. This will minimize the number of dirty objects on the system and create more candidates to be dropped.

This file is not a means to control the growth of the various kernel caches (inodes, dentries, pagecache, etc...) These objects are automatically reclaimed by the kernel when memory is needed elsewhere on the system.

Use of this file can cause performance problems. Since it discards cached objects, it may cost a significant amount of I/O and CPU to recreate the dropped objects,

especially if they were under heavy use. Because of this, use outside of a testing or debugging environment is not recommended.

You may see informational messages in your kernel log when this file is used:

```
cat (1234): drop_caches: 3
```

These are informational only. They do not mean that anything is wrong with your system. To disable them, echo 4 (bit 2) into `drop_caches`.

### **extfrag\_threshold**

This parameter affects whether the kernel will compact memory or direct reclaim to satisfy a high-order allocation. The `extfrag/extfrag_index` file in `debugfs` shows what the fragmentation index for each order is in each zone in the system. Values tending towards 0 imply allocations would fail due to lack of memory, values towards 1000 imply failures are due to fragmentation and -1 implies that the allocation will succeed as long as watermarks are met.

The kernel will not compact memory in a zone if the fragmentation index is  $\leq$  `extfrag_threshold`. The default value is 500.

### **highmem\_is\_dirtyable**

Available only for systems with `CONFIG_HIGHMEM` enabled (32b systems).

This parameter controls whether the high memory is considered for dirty writers throttling. This is not the case by default which means that only the amount of memory directly visible/usable by the kernel can be dirtied. As a result, on systems with a large amount of memory and `lowmem` basically depleted writers might be throttled too early and streaming writes can get very slow.

Changing the value to non zero would allow more memory to be dirtied and thus allow writers to write more data which can be flushed to the storage more effectively. Note this also comes with a risk of pre-mature OOM killer because some writers (e.g. direct block device writes) can only use the low memory and they can fill it up with dirty data without any throttling.

### **hugetlb\_shm\_group**

`hugetlb_shm_group` contains group id that is allowed to create SysV shared memory segment using `hugetlb` page.

### laptop\_mode

`laptop_mode` is a knob that controls “laptop mode” . All the things that are controlled by this knob are discussed in `Documentation/admin-guide/laptops/laptop-mode.rst`.

### legacy\_va\_layout

If non-zero, this sysctl disables the new 32-bit mmap layout - the kernel will use the legacy (2.4) layout for all processes.

### lowmem\_reserve\_ratio

For some specialised workloads on highmem machines it is dangerous for the kernel to allow process memory to be allocated from the “lowmem” zone. This is because that memory could then be pinned via the `mlock()` system call, or by unavailability of swap space.

And on large highmem machines this lack of reclaimable lowmem memory can be fatal.

So the Linux page allocator has a mechanism which prevents allocations which could use highmem from using too much lowmem. This means that a certain amount of lowmem is defended from the possibility of being captured into pinned user memory.

(The same argument applies to the old 16 megabyte ISA DMA region. This mechanism will also defend that region from allocations which could use highmem or lowmem).

The `lowmem_reserve_ratio` tunable determines how aggressive the kernel is in defending these lower zones.

If you have a machine which uses highmem or ISA DMA and your applications are using `mlock()`, or if you are running with no swap then you probably should change the `lowmem_reserve_ratio` setting.

The `lowmem_reserve_ratio` is an array. You can see them by reading this file:

```
% cat /proc/sys/vm/lowmem_reserve_ratio
256    256    32
```

But, these values are not used directly. The kernel calculates # of protection pages for each zones from them. These are shown as array of protection pages in `/proc/zoneinfo` like followings. (This is an example of x86-64 box). Each zone has an array of protection pages like this:

```
Node 0, zone      DMA
  pages free     1355
        min       3
        low       3
        high      4
        :
        :
```

(continues on next page)

(continued from previous page)

```

numa_other  0
  protection: (0, 2004, 2004, 2004)
  ~~~~~
pagesets
  cpu: 0 pcp: 0
  :
```

These protections are added to score to judge whether this zone should be used for page allocation or should be reclaimed.

In this example, if normal pages (index=2) are required to this DMA zone and watermark[WMARK\_HIGH] is used for watermark, the kernel judges this zone should not be used because pages\_free(1355) is smaller than watermark + protection[2] (4 + 2004 = 2008). If this protection value is 0, this zone would be used for normal page requirement. If requirement is DMA zone(index=0), protection[0] (=0) is used.

zone[i]'s protection[j] is calculated by following expression:

```

(i < j):
  zone[i]->protection[j]
  = (total sums of managed_pages from zone[i+1] to zone[j] on the node)
    / lowmem_reserve_ratio[i];
(i = j):
  (should not be protected. = 0);
(i > j):
  (not necessary, but looks 0)
```

The default values of lowmem\_reserve\_ratio[i] are

256	(if zone[i] means DMA or DMA32 zone)
32	(others)

As above expression, they are reciprocal number of ratio. 256 means 1/256. # of protection pages becomes about “0.39%” of total managed pages of higher zones on the node.

If you would like to protect more pages, smaller values are effective. The minimum value is 1 (1/1 -> 100%). The value less than 1 completely disables protection of the pages.

### max\_map\_count:

This file contains the maximum number of memory map areas a process may have. Memory map areas are used as a side-effect of calling malloc, directly by mmap, mprotect, and madvise, and also when loading shared libraries.

While most applications need less than a thousand maps, certain programs, particularly malloc debuggers, may consume lots of them, e.g., up to one or two maps per allocation.

The default value is 65536.

### **memory\_failure\_early\_kill:**

Control how to kill processes when uncorrected memory error (typically a 2bit error in a memory module) is detected in the background by hardware that cannot be handled by the kernel. In some cases (like the page still having a valid copy on disk) the kernel will handle the failure transparently without affecting any applications. But if there is no other uptodate copy of the data it will kill to prevent any data corruptions from propagating.

1: Kill all processes that have the corrupted and not reloadable page mapped as soon as the corruption is detected. Note this is not supported for a few types of pages, like kernel internally allocated data or the swap cache, but works for the majority of user pages.

0: Only unmap the corrupted page from all processes and only kill a process who tries to access it.

The kill is done using a catchable SIGBUS with BUS\_MCEERR\_AO, so processes can handle this if they want to.

This is only active on architectures/platforms with advanced machine check handling and depends on the hardware capabilities.

Applications can override this setting individually with the PR\_MCE\_KILL prctl

### **memory\_failure\_recovery**

Enable memory failure recovery (when supported by the platform)

1: Attempt recovery.

0: Always panic on a memory failure.

### **min\_free\_kbytes**

This is used to force the Linux VM to keep a minimum number of kilobytes free. The VM uses this number to compute a watermark[WMARK\_MIN] value for each lowmem zone in the system. Each lowmem zone gets a number of reserved free pages based proportionally on its size.

Some minimal amount of memory is needed to satisfy PF\_MEMALLOC allocations; if you set this to lower than 1024KB, your system will become subtly broken, and prone to deadlock under high loads.

Setting this too high will OOM your machine instantly.

### **min\_slab\_ratio**

This is available only on NUMA kernels.

A percentage of the total pages in each zone. On Zone reclaim (fallback from the local zone occurs) slabs will be reclaimed if more than this percentage of pages in a zone are reclaimable slab pages. This insures that the slab growth stays under control even in NUMA systems that rarely perform global reclaim.

The default is 5 percent.

Note that slab reclaim is triggered in a per zone / node fashion. The process of reclaiming slab memory is currently not node specific and may not be fast.

### **min\_unmapped\_ratio**

This is available only on NUMA kernels.

This is a percentage of the total pages in each zone. Zone reclaim will only occur if more than this percentage of pages are in a state that `zone_reclaim_mode` allows to be reclaimed.

If `zone_reclaim_mode` has the value 4 OR' d, then the percentage is compared against all file-backed unmapped pages including swapcache pages and tmpfs files. Otherwise, only unmapped pages backed by normal files but not tmpfs files and similar are considered.

The default is 1 percent.

### **mmap\_min\_addr**

This file indicates the amount of address space which a user process will be restricted from mmaping. Since kernel null dereference bugs could accidentally operate based on the information in the first couple of pages of memory userspace processes should not be allowed to write to them. By default this value is set to 0 and no protections will be enforced by the security module. Setting this value to something like 64k will allow the vast majority of applications to work correctly and provide defense in depth against future potential kernel bugs.

### **mmap\_rnd\_bits**

This value can be used to select the number of bits to use to determine the random offset to the base address of vma regions resulting from mmap allocations on architectures which support tuning address space randomization. This value will be bounded by the architecture' s minimum and maximum supported values.

This value can be changed after boot using the `/proc/sys/vm/mmap_rnd_bits` tunable

### **mmap\_rnd\_compat\_bits**

This value can be used to select the number of bits to use to determine the random offset to the base address of vma regions resulting from mmap allocations for applications run in compatibility mode on architectures which support tuning address space randomization. This value will be bounded by the architecture's minimum and maximum supported values.

This value can be changed after boot using the `/proc/sys/vm/mmap_rnd_compat_bits` tunable

### **nr\_hugepages**

Change the minimum size of the hugepage pool.

See [Documentation/admin-guide/mm/hugetlbpage.rst](#)

### **nr\_hugepages\_mempolicy**

Change the size of the hugepage pool at run-time on a specific set of NUMA nodes.

See [Documentation/admin-guide/mm/hugetlbpage.rst](#)

### **nr\_overcommit\_hugepages**

Change the maximum size of the hugepage pool. The maximum is `nr_hugepages + nr_overcommit_hugepages`.

See [Documentation/admin-guide/mm/hugetlbpage.rst](#)

### **nr\_trim\_pages**

This is available only on NOMMU kernels.

This value adjusts the excess page trimming behaviour of power-of-2 aligned NOMMU mmap allocations.

A value of 0 disables trimming of allocations entirely, while a value of 1 trims excess pages aggressively. Any value  $\geq 1$  acts as the watermark where trimming of allocations is initiated.

The default value is 1.

See [Documentation/nommu-mmap.txt](#) for more information.

## numa\_zonelist\_order

This sysctl is only for NUMA and it is deprecated. Anything but Node order will fail!

‘where the memory is allocated from’ is controlled by zonelists.

(This documentation ignores ZONE\_HIGHMEM/ZONE\_DMA32 for simple explanation. you may be able to read ZONE\_DMA as ZONE\_DMA32···)

In non-NUMA case, a zonelist for GFP\_KERNEL is ordered as following. ZONE\_NORMAL -> ZONE\_DMA This means that a memory allocation request for GFP\_KERNEL will get memory from ZONE\_DMA only when ZONE\_NORMAL is not available.

In NUMA case, you can think of following 2 types of order. Assume 2 node NUMA and below is zonelist of Node(0)’ s GFP\_KERNEL:

(A) Node(0) ZONE_NORMAL -> Node(0) ZONE_DMA -> Node(1) ZONE_NORMAL
(B) Node(0) ZONE_NORMAL -> Node(1) ZONE_NORMAL -> Node(0) ZONE_DMA.

Type(A) offers the best locality for processes on Node(0), but ZONE\_DMA will be used before ZONE\_NORMAL exhaustion. This increases possibility of out-of-memory(OOM) of ZONE\_DMA because ZONE\_DMA is tend to be small.

Type(B) cannot offer the best locality but is more robust against OOM of the DMA zone.

Type(A) is called as “Node” order. Type (B) is “Zone” order.

“Node order” orders the zonelists by node, then by zone within each node. Specify “[Nn]ode” for node order

“Zone Order” orders the zonelists by zone type, then by node within each zone. Specify “[Zz]one” for zone order.

Specify “[Dd]efault” to request automatic configuration.

On 32-bit, the Normal zone needs to be preserved for allocations accessible by the kernel, so “zone” order will be selected.

On 64-bit, devices that require DMA32/DMA are relatively rare, so “node” order will be selected.

Default order is recommended unless this is causing problems for your system/application.

## oom\_dump\_tasks

Enables a system-wide task dump (excluding kernel threads) to be produced when the kernel performs an OOM-killing and includes such information as pid, uid, tgid, vm size, rss, pgtables\_bytes, swapents, oom\_score\_adj score, and name. This is helpful to determine why the OOM killer was invoked, to identify the rogue task that caused it, and to determine why the OOM killer chose the task it did to kill.

If this is set to zero, this information is suppressed. On very large systems with thousands of tasks it may not be feasible to dump the memory state information

for each one. Such systems should not be forced to incur a performance penalty in OOM conditions when the information may not be desired.

If this is set to non-zero, this information is shown whenever the OOM killer actually kills a memory-hogging task.

The default value is 1 (enabled).

### **oom\_kill\_allocating\_task**

This enables or disables killing the OOM-triggering task in out-of-memory situations.

If this is set to zero, the OOM killer will scan through the entire tasklist and select a task based on heuristics to kill. This normally selects a rogue memory-hogging task that frees up a large amount of memory when killed.

If this is set to non-zero, the OOM killer simply kills the task that triggered the out-of-memory condition. This avoids the expensive tasklist scan.

If `panic_on_oom` is selected, it takes precedence over whatever value is used in `oom_kill_allocating_task`.

The default value is 0.

### **overcommit\_kbytes**

When `overcommit_memory` is set to 2, the committed address space is not permitted to exceed swap plus this amount of physical RAM. See below.

Note: `overcommit_kbytes` is the counterpart of `overcommit_ratio`. Only one of them may be specified at a time. Setting one disables the other (which then appears as 0 when read).

### **overcommit\_memory**

This value contains a flag that enables memory overcommitment.

When this flag is 0, the kernel attempts to estimate the amount of free memory left when userspace requests more memory.

When this flag is 1, the kernel pretends there is always enough memory until it actually runs out.

When this flag is 2, the kernel uses a “never overcommit” policy that attempts to prevent any overcommit of memory. Note that `user_reserve_kbytes` affects this policy.

This feature can be very useful because there are a lot of programs that `malloc()` huge amounts of memory “just-in-case” and don’t use much of it.

The default value is 0.

See [Documentation/vm/overcommit-accounting.rst](#) and `mm/util.c::__vm_enough_memory()` for more information.

### **overcommit\_ratio**

When `overcommit_memory` is set to 2, the committed address space is not permitted to exceed swap plus this percentage of physical RAM. See above.

### **page-cluster**

`page-cluster` controls the number of pages up to which consecutive pages are read in from swap in a single attempt. This is the swap counterpart to page cache readahead. The mentioned consecutivity is not in terms of virtual/physical addresses, but consecutive on swap space - that means they were swapped out together.

It is a logarithmic value - setting it to zero means “1 page”, setting it to 1 means “2 pages”, setting it to 2 means “4 pages”, etc. Zero disables swap readahead completely.

The default value is three (eight pages at a time). There may be some small benefits in tuning this to a different value if your workload is swap-intensive.

Lower values mean lower latencies for initial faults, but at the same time extra faults and I/O delays for following faults if they would have been part of that consecutive pages readahead would have brought in.

### **panic\_on\_oom**

This enables or disables panic on out-of-memory feature.

If this is set to 0, the kernel will kill some rogue process, called `oom_killer`. Usually, `oom_killer` can kill rogue processes and system will survive.

If this is set to 1, the kernel panics when out-of-memory happens. However, if a process limits using nodes by `mempolicy/cpusets`, and those nodes become memory exhaustion status, one process may be killed by oom-killer. No panic occurs in this case. Because other nodes' memory may be free. This means system total status may be not fatal yet.

If this is set to 2, the kernel panics compulsorily even on the above-mentioned. Even oom happens under memory cgroup, the whole system panics.

The default value is 0.

1 and 2 are for failover of clustering. Please select either according to your policy of failover.

`panic_on_oom=2+kdump` gives you very strong tool to investigate why oom happens. You can get snapshot.

### percpu\_pagelist\_fraction

This is the fraction of pages at most (high mark `pcp->high`) in each zone that are allocated for each per cpu page list. The min value for this is 8. It means that we don't allow more than 1/8th of pages in each zone to be allocated in any single `per_cpu_pagelist`. This entry only changes the value of hot per cpu pagelists. User can specify a number like 100 to allocate 1/100th of each zone to each per cpu page list.

The batch value of each per cpu pagelist is also updated as a result. It is set to `pcp->high/4`. The upper limit of batch is `(PAGE_SHIFT * 8)`

The initial value is zero. Kernel does not use this value at boot time to set the high water marks for each per cpu page list. If the user writes '0' to this `sysctl`, it will revert to this default behavior.

### stat\_interval

The time interval between which vm statistics are updated. The default is 1 second.

### stat\_refresh

Any read or write (by root only) flushes all the per-cpu vm statistics into their global totals, for more accurate reports when testing e.g. `cat /proc/sys/vm/stat_refresh /proc/meminfo`

As a side-effect, it also checks for negative totals (elsewhere reported as 0) and "fails" with `EINVAL` if any are found, with a warning in `dmesg`. (At time of writing, a few stats are known sometimes to be found negative, with no ill effects: errors and warnings on these stats are suppressed.)

### numa\_stat

This interface allows runtime configuration of numa statistics.

When page allocation performance becomes a bottleneck and you can tolerate some possible tool breakage and decreased numa counter precision, you can do:

```
echo 0 > /proc/sys/vm/numa_stat
```

When page allocation performance is not a bottleneck and you want all tooling to work, you can do:

```
echo 1 > /proc/sys/vm/numa_stat
```

## **swappiness**

This control is used to define the rough relative IO cost of swapping and filesystem paging, as a value between 0 and 200. At 100, the VM assumes equal IO cost and will thus apply memory pressure to the page cache and swap-backed pages equally; lower values signify more expensive swap IO, higher values indicates cheaper.

Keep in mind that filesystem IO patterns under memory pressure tend to be more efficient than swap's random IO. An optimal value will require experimentation and will also be workload-dependent.

The default value is 60.

For in-memory swap, like zram or zswap, as well as hybrid setups that have swap on faster devices than the filesystem, values beyond 100 can be considered. For example, if the random IO against the swap device is on average 2x faster than IO from the filesystem, swappiness should be 133 ( $x + 2x = 200$ ,  $2x = 133.33$ ).

At 0, the kernel will not initiate swap until the amount of free and file-backed pages is less than the high watermark in a zone.

## **unprivileged\_userfaultfd**

This flag controls whether unprivileged users can use the userfaultfd system calls. Set this to 1 to allow unprivileged users to use the userfaultfd system calls, or set this to 0 to restrict userfaultfd to only privileged users (with `SYS_CAP_PTRACE` capability).

The default value is 1.

## **user\_reserve\_kbytes**

When `overcommit_memory` is set to 2, “never overcommit” mode, reserve  $\min(3\%$  of current process size, `user_reserve_kbytes`) of free memory. This is intended to prevent a user from starting a single memory hogging process, such that they cannot recover (kill the hog).

`user_reserve_kbytes` defaults to  $\min(3\%$  of the current process size, 128MB).

If this is reduced to zero, then the user will be allowed to allocate all free memory with a single process, minus `admin_reserve_kbytes`. Any subsequent attempts to execute a command will result in “fork: Cannot allocate memory” .

Changing this takes effect whenever an application requests memory.

### **vfs\_cache\_pressure**

This percentage value controls the tendency of the kernel to reclaim the memory which is used for caching of directory and inode objects.

At the default value of `vfs_cache_pressure=100` the kernel will attempt to reclaim dentries and inodes at a “fair” rate with respect to pagecache and swapcache reclaim. Decreasing `vfs_cache_pressure` causes the kernel to prefer to retain dentry and inode caches. When `vfs_cache_pressure=0`, the kernel will never reclaim dentries and inodes due to memory pressure and this can easily lead to out-of-memory conditions. Increasing `vfs_cache_pressure` beyond 100 causes the kernel to prefer to reclaim dentries and inodes.

Increasing `vfs_cache_pressure` significantly beyond 100 may have negative performance impact. Reclaim code needs to take various locks to find freeable directory and inode objects. With `vfs_cache_pressure=1000`, it will look for ten times more freeable objects than there are.

### **watermark\_boost\_factor**

This factor controls the level of reclaim when memory is being fragmented. It defines the percentage of the high watermark of a zone that will be reclaimed if pages of different mobility are being mixed within pageblocks. The intent is that compaction has less work to do in the future and to increase the success rate of future high-order allocations such as SLUB allocations, THP and hugetlbfs pages.

To make it sensible with respect to the `watermark_scale_factor` parameter, the unit is in fractions of 10,000. The default value of 15,000 on !DISCONTIGMEM configurations means that up to 150% of the high watermark will be reclaimed in the event of a pageblock being mixed due to fragmentation. The level of reclaim is determined by the number of fragmentation events that occurred in the recent past. If this value is smaller than a pageblock then a pageblocks worth of pages will be reclaimed (e.g. 2MB on 64-bit x86). A boost factor of 0 will disable the feature.

### **watermark\_scale\_factor**

This factor controls the aggressiveness of `kswapd`. It defines the amount of memory left in a node/system before `kswapd` is woken up and how much memory needs to be free before `kswapd` goes back to sleep.

The unit is in fractions of 10,000. The default value of 10 means the distances between watermarks are 0.1% of the available memory in the node/system. The maximum value is 1000, or 10% of memory.

A high rate of threads entering direct reclaim (allocstall) or `kswapd` going to sleep prematurely (`kswapd_low_wmark_hit_quickly`) can indicate that the number of free pages `kswapd` maintains for latency reasons is too small for the allocation bursts occurring in the system. This knob can then be used to tune `kswapd` aggressiveness accordingly.

## zone\_reclaim\_mode

Zone\_reclaim\_mode allows someone to set more or less aggressive approaches to reclaim memory when a zone runs out of memory. If it is set to zero then no zone reclaim occurs. Allocations will be satisfied from other zones / nodes in the system.

This is value OR' ed together of

1	Zone reclaim on
2	Zone reclaim writes dirty pages out
4	Zone reclaim swaps pages

zone\_reclaim\_mode is disabled by default. For file servers or workloads that benefit from having their data cached, zone\_reclaim\_mode should be left disabled as the caching effect is likely to be more important than data locality.

zone\_reclaim may be enabled if it' s known that the workload is partitioned such that each partition fits within a NUMA node and that accessing remote memory would cause a measurable performance reduction. The page allocator will then reclaim easily reusable pages (those page cache pages that are currently not used) before allocating off node pages.

Allowing zone reclaim to write out pages stops processes that are writing large amounts of data from dirtying pages on other nodes. Zone reclaim will write out dirty pages if a zone fills up and so effectively throttle the process. This may decrease the performance of a single process since it cannot use all of system memory to buffer the outgoing writes anymore but it preserve the memory on other nodes so that the performance of other processes running on other nodes will not be affected.

Allowing regular swap effectively restricts allocations to the local node unless explicitly overridden by memory policies or cpuset configurations.

This section describes CPU vulnerabilities and their mitigations.



## HARDWARE VULNERABILITIES

This section describes CPU vulnerabilities and provides an overview of the possible mitigations along with guidance for selecting mitigations if they are configurable at compile, boot or run time.

### 5.1 Spectre Side Channels

Spectre is a class of side channel attacks that exploit branch prediction and speculative execution on modern CPUs to read memory, possibly bypassing access controls. Speculative execution side channel exploits do not modify memory but attempt to infer privileged data in the memory.

This document covers Spectre variant 1 and Spectre variant 2.

#### 5.1.1 Affected processors

Speculative execution side channel methods affect a wide range of modern high performance processors, since most modern high speed processors use branch prediction and speculative execution.

The following CPUs are vulnerable:

- Intel Core, Atom, Pentium, and Xeon processors
- AMD Phenom, EPYC, and Zen processors
- IBM POWER and zSeries processors
- Higher end ARM processors
- Apple CPUs
- Higher end MIPS CPUs
- Likely most other high performance CPUs. Contact your CPU vendor for details.

Whether a processor is affected or not can be read out from the Spectre vulnerability files in sysfs. See Spectre system information.

### 5.1.2 Related CVEs

The following CVE entries describe Spectre variants:

CVE-2017-5753	Bounds check bypass	Spectre variant 1
CVE-2017-5715	Branch target injection	Spectre variant 2
CVE-2019-1125	Spectre v1 swapgs	Spectre variant 1 (swapgs)

### 5.1.3 Problem

CPUs use speculative operations to improve performance. That may leave traces of memory accesses or computations in the processor's caches, buffers, and branch predictors. Malicious software may be able to influence the speculative execution paths, and then use the side effects of the speculative execution in the CPUs' caches and buffers to infer privileged data touched during the speculative execution.

Spectre variant 1 attacks take advantage of speculative execution of conditional branches, while Spectre variant 2 attacks use speculative execution of indirect branches to leak privileged memory. See [1] [5] [7] [10] [11].

### 5.1.4 Spectre variant 1 (Bounds Check Bypass)

The bounds check bypass attack [2] takes advantage of speculative execution that bypasses conditional branch instructions used for memory access bounds check (e.g. checking if the index of an array results in memory access within a valid range). This results in memory accesses to invalid memory (with out-of-bound index) that are done speculatively before validation checks resolve. Such speculative memory accesses can leave side effects, creating side channels which leak information to the attacker.

There are some extensions of Spectre variant 1 attacks for reading data over the network, see [12]. However such attacks are difficult, low bandwidth, fragile, and are considered low risk.

Note that, despite "Bounds Check Bypass" name, Spectre variant 1 is not only about user-controlled array bounds checks. It can affect any conditional checks. The kernel entry code interrupt, exception, and NMI handlers all have conditional swapgs checks. Those may be problematic in the context of Spectre v1, as kernel code can speculatively run with a user GS.

### 5.1.5 Spectre variant 2 (Branch Target Injection)

The branch target injection attack takes advantage of speculative execution of indirect branches [3]. The indirect branch predictors inside the processor used to guess the target of indirect branches can be influenced by an attacker, causing gadget code to be speculatively executed, thus exposing sensitive data touched by the victim. The side effects left in the CPU's caches during speculative execution can be measured to infer data values.

In Spectre variant 2 attacks, the attacker can steer speculative indirect branches in the victim to gadget code by poisoning the branch target buffer of a CPU used for predicting indirect branch addresses. Such poisoning could be done by indirect branching into existing code, with the address offset of the indirect branch under the attacker's control. Since the branch prediction on impacted hardware does not fully disambiguate branch address and uses the offset for prediction, this could cause privileged code's indirect branch to jump to a gadget code with the same offset.

The most useful gadgets take an attacker-controlled input parameter (such as a register value) so that the memory read can be controlled. Gadgets without input parameters might be possible, but the attacker would have very little control over what memory can be read, reducing the risk of the attack revealing useful data.

One other variant 2 attack vector is for the attacker to poison the return stack buffer (RSB) [13] to cause speculative subroutine return instruction execution to go to a gadget. An attacker's imbalanced subroutine call instructions might "poison" entries in the return stack buffer which are later consumed by a victim's subroutine return instructions. This attack can be mitigated by flushing the return stack buffer on context switch, or virtual machine (VM) exit.

On systems with simultaneous multi-threading (SMT), attacks are possible from the sibling thread, as level 1 cache and branch target buffer (BTB) may be shared between hardware threads in a CPU core. A malicious program running on the sibling thread may influence its peer's BTB to steer its indirect branch speculations to gadget code, and measure the speculative execution's side effects left in level 1 cache to infer the victim's data.

### 5.1.6 Attack scenarios

The following list of attack scenarios have been anticipated, but may not cover all possible attack vectors.

#### 1. A user process attacking the kernel

##### Spectre variant 1

The attacker passes a parameter to the kernel via a register or via a known address in memory during a syscall. Such parameter may be used later by the kernel as an index to an array or to derive a pointer for a Spectre variant 1 attack. The index or pointer is invalid, but bound checks are bypassed in the code branch taken for speculative execution. This could cause privileged memory to be accessed and leaked.

For kernel code that has been identified where data pointers could potentially be influenced for Spectre attacks, new “nospec” accessor macros are used to prevent speculative loading of data.

### Spectre variant 1 (swapgs)

An attacker can train the branch predictor to speculatively skip the swapgs path for an interrupt or exception. If they initialize the GS register to a user-space value, if the swapgs is speculatively skipped, subsequent GS-related percpu accesses in the speculation window will be done with the attacker-controlled GS value. This could cause privileged memory to be accessed and leaked.

For example:

```
if (coming from user space)
    swapgs
mov %gs:<percpu_offset>, %reg
mov (%reg), %reg1
```

When coming from user space, the CPU can speculatively skip the swapgs, and then do a speculative percpu load using the user GS value. So the user can speculatively force a read of any kernel value. If a gadget exists which uses the percpu value as an address in another load/store, then the contents of the kernel value may become visible via an L1 side channel attack.

A similar attack exists when coming from kernel space. The CPU can speculatively do the swapgs, causing the user GS to get used for the rest of the speculative window.

### Spectre variant 2

A spectre variant 2 attacker can poison the branch target buffer (BTB) before issuing syscall to launch an attack. After entering the kernel, the kernel could use the poisoned branch target buffer on indirect jump and jump to gadget code in speculative execution.

If an attacker tries to control the memory addresses leaked during speculative execution, he would also need to pass a parameter to the gadget, either through a register or a known address in memory. After the gadget has executed, he can measure the side effect.

The kernel can protect itself against consuming poisoned branch target buffer entries by using return trampolines (also known as “retpoline”) [3] [9] for all indirect branches. Return trampolines trap speculative execution paths to prevent jumping to gadget code during speculative execution. x86 CPUs with Enhanced Indirect Branch Restricted Speculation (Enhanced IBRS) available in hardware should use the feature to mitigate Spectre variant 2 instead of retpoline. Enhanced IBRS is more efficient than retpoline.

There may be gadget code in firmware which could be exploited with Spectre variant 2 attack by a rogue user process. To mitigate such attacks on x86, Indirect Branch Restricted Speculation (IBRS) feature is turned on before the kernel invokes any firmware code.

## **2. A user process attacking another user process**

A malicious user process can try to attack another user process, either via a context switch on the same hardware thread, or from the sibling hyperthread sharing a physical processor core on simultaneous multi-threading (SMT) system.

Spectre variant 1 attacks generally require passing parameters between the processes, which needs a data passing relationship, such as remote procedure calls (RPC). Those parameters are used in gadget code to derive invalid data pointers accessing privileged memory in the attacked process.

Spectre variant 2 attacks can be launched from a rogue process by poisoning the branch target buffer. This can influence the indirect branch targets for a victim process that either runs later on the same hardware thread, or running concurrently on a sibling hardware thread sharing the same physical core.

A user process can protect itself against Spectre variant 2 attacks by using the `prctl()` syscall to disable indirect branch speculation for itself. An administrator can also cordon off an unsafe process from polluting the branch target buffer by disabling the process' s indirect branch speculation. This comes with a performance cost from not using indirect branch speculation and clearing the branch target buffer. When SMT is enabled on x86, for a process that has indirect branch speculation disabled, Single Threaded Indirect Branch Predictors (STIBP) [4] are turned on to prevent the sibling thread from controlling branch target buffer. In addition, the Indirect Branch Prediction Barrier (IBPB) is issued to clear the branch target buffer when context switching to and from such process.

On x86, the return stack buffer is stuffed on context switch. This prevents the branch target buffer from being used for branch prediction when the return stack buffer underflows while switching to a deeper call stack. Any poisoned entries in the return stack buffer left by the previous process will also be cleared.

User programs should use address space randomization to make attacks more difficult (Set `/proc/sys/kernel/randomize_va_space = 1` or `2`).

### 3. A virtualized guest attacking the host

The attack mechanism is similar to how user processes attack the kernel. The kernel is entered via hyper-calls or other virtualization exit paths.

For Spectre variant 1 attacks, rogue guests can pass parameters (e.g. in registers) via hyper-calls to derive invalid pointers to speculate into privileged memory after entering the kernel. For places where such kernel code has been identified, nospec accessor macros are used to stop speculative memory access.

For Spectre variant 2 attacks, rogue guests can poison the branch target buffer or return stack buffer, causing the kernel to jump to gadget code in the speculative execution paths.

To mitigate variant 2, the host kernel can use return trampolines for indirect branches to bypass the poisoned branch target buffer, and flushing the return stack buffer on VM exit. This prevents rogue guests from affecting indirect branching in the host kernel.

To protect host processes from rogue guests, host processes can have indirect branch speculation disabled via `prctl()`. The branch target buffer is cleared before context switching to such processes.

### 4. A virtualized guest attacking other guest

A rogue guest may attack another guest to get data accessible by the other guest.

Spectre variant 1 attacks are possible if parameters can be passed between guests. This may be done via mechanisms such as shared memory or message passing. Such parameters could be used to derive data pointers to privileged data in guest. The privileged data could be accessed by gadget code in the victim's speculation paths.

Spectre variant 2 attacks can be launched from a rogue guest by poisoning the branch target buffer or the return stack buffer. Such poisoned entries could be used to influence speculation execution paths in the victim guest.

Linux kernel mitigates attacks to other guests running in the same CPU hardware thread by flushing the return stack buffer on VM exit, and clearing the branch target buffer before switching to a new guest.

If SMT is used, Spectre variant 2 attacks from an untrusted guest in the sibling hyperthread can be mitigated by the administrator, by turning off the unsafe guest's indirect branch speculation via `prctl()`. A guest can also protect itself by turning on microcode based mitigations (such as IBPB or STIBP on x86) within the guest.

### 5.1.7 Spectre system information

The Linux kernel provides a sysfs interface to enumerate the current mitigation status of the system for Spectre: whether the system is vulnerable, and which mitigations are active.

The sysfs file showing Spectre variant 1 mitigation status is:

```
/sys/devices/system/cpu/vulnerabilities/spectre_v1
```

The possible values in this file are:

'Not affected'	The processor is not vulnerable.
'Vulnerable: __user pointer sanitization and usercopy barriers only; no swapgs barriers'	The swapgs protections are disabled; otherwise it has protection in the kernel on a case by case base with explicit pointer sanitation and usercopy LFENCE barriers.
'Mitigation: usercopy/swapgs barriers and __user pointer sanitization'	Protection in the kernel on a case by case base with explicit pointer sanitation, usercopy LFENCE barriers, and swapgs LFENCE barriers.

However, the protections are put in place on a case by case basis, and there is no guarantee that all possible attack vectors for Spectre variant 1 are covered.

The spectre\_v2 kernel file reports if the kernel has been compiled with retpoline mitigation or if the CPU has hardware mitigation, and if the CPU has support for additional process-specific mitigation.

This file also reports CPU features enabled by microcode to mitigate attack between user processes:

1. Indirect Branch Prediction Barrier (IBPB) to add additional isolation between processes of different users.
2. Single Thread Indirect Branch Predictors (STIBP) to add additional isolation between CPU threads running on the same core.

These CPU features may impact performance when used and can be enabled per process on a case-by-case base.

The sysfs file showing Spectre variant 2 mitigation status is:

```
/sys/devices/system/cpu/vulnerabilities/spectre_v2
```

The possible values in this file are:

- Kernel status:

'Not affected'	The processor is not vulnerable
'Vulnerable'	Vulnerable, no mitigation
'Mitigation: Full generic retpoline'	Software-focused mitigation
'Mitigation: Full AMD retpoline'	AMD-specific software mitigation
'Mitigation: Enhanced IBRS'	Hardware-focused mitigation

- Firmware status: Show if Indirect Branch Restricted Speculation (IBRS) is used to protect against Spectre variant 2 attacks when calling firmware (x86 only).

'IBRS_FW'	Protection against user program attacks when calling firmware
-----------	---

- Indirect branch prediction barrier (IBPB) status for protection between processes of different users. This feature can be controlled through `prctl()` per process, or through kernel command line options. This is an x86 only feature. For more details see below.

'IBPB: disabled'	IBPB unused
'IBPB: always-on'	Use IBPB on all tasks
'IBPB: conditional'	Use IBPB on SECCOMP or indirect branch restricted tasks

- Single threaded indirect branch prediction (STIBP) status for protection between different hyper threads. This feature can be controlled through `prctl` per process, or through kernel command line options. This is x86 only feature. For more details see below.

'STIBP: disabled'	STIBP unused
'STIBP: forced'	Use STIBP on all tasks
'STIBP: conditional'	Use STIBP on SECCOMP or indirect branch restricted tasks

- Return stack buffer (RSB) protection status:

'RSB filling'	Protection of RSB on context switch enabled
---------------	---

Full mitigation might require a microcode update from the CPU vendor. When the necessary microcode is not available, the kernel will report vulnerability.

### 5.1.8 Turning on mitigation for Spectre variant 1 and Spectre variant 2

#### 1. Kernel mitigation

##### Spectre variant 1

For the Spectre variant 1, vulnerable kernel code (as determined by code audit or scanning tools) is annotated on a case by case basis to use `nospec` accessor macros for bounds clipping [2] to avoid any usable disclosure gadgets. However, it may not cover all attack vectors for Spectre variant 1.

Copy-from-user code has an `LFENCE` barrier to prevent the `access_ok()` check from being mis-speculated. The barrier is done by the `barrier_nospec()` macro.

For the swapgs variant of Spectre variant 1, LFENCE barriers are added to interrupt, exception and NMI entry where needed. These barriers are done by the `FENCE_SWAPGS_KERNEL_ENTRY` and `FENCE_SWAPGS_USER_ENTRY` macros.

## Spectre variant 2

For Spectre variant 2 mitigation, the compiler turns indirect calls or jumps in the kernel into equivalent return trampolines (retpolines) [3] [9] to go to the target addresses. Speculative execution paths under retpolines are trapped in an infinite loop to prevent any speculative execution jumping to a gadget.

To turn on retpoline mitigation on a vulnerable CPU, the kernel needs to be compiled with a gcc compiler that supports the `-mindirect-branch=thunk-extern` `-mindirect-branch-register` options. If the kernel is compiled with a Clang compiler, the compiler needs to support `-mretpoline-external-thunk` option. The kernel config `CONFIG_RETPOLINE` needs to be turned on, and the CPU needs to run with the latest updated microcode.

On Intel Skylake-era systems the mitigation covers most, but not all, cases. See [3] for more details.

On CPUs with hardware mitigation for Spectre variant 2 (e.g. Enhanced IBRS on x86), retpoline is automatically disabled at run time.

The retpoline mitigation is turned on by default on vulnerable CPUs. It can be forced on or off by the administrator via the kernel command line and sysfs control files. See Mitigation control on the kernel command line.

On x86, indirect branch restricted speculation is turned on by default before invoking any firmware code to prevent Spectre variant 2 exploits using the firmware.

Using kernel address space randomization (`CONFIG_RANDOMIZE_SLAB=y` and `CONFIG_SLAB_FREELIST_RANDOM=y` in the kernel configuration) makes attacks on the kernel generally more difficult.

## 2. User program mitigation

User programs can mitigate Spectre variant 1 using LFENCE or “bounds clipping” . For more details see [2].

For Spectre variant 2 mitigation, individual user programs can be compiled with return trampolines for indirect branches. This protects them from consuming poisoned entries in the branch target buffer left by malicious software. Alternatively, the programs can disable their indirect branch speculation via `prctl()` (See Documentation/userspace-api/spec\_ctrl.rst). On x86, this will turn on STIBP to guard against attacks from the sibling thread when the user program is running, and

use IBPB to flush the branch target buffer when switching to/from the program.

Restricting indirect branch speculation on a user program will also prevent the program from launching a variant 2 attack on x86. All sandboxed SECCOMP programs have indirect branch speculation restricted by default. Administrators can change that behavior via the kernel command line and sysfs control files. See Mitigation control on the kernel command line.

Programs that disable their indirect branch speculation will have more overhead and run slower.

User programs should use address space randomization (`/proc/sys/kernel/randomize_va_space = 1` or `2`) to make attacks more difficult.

### 3. VM mitigation

Within the kernel, Spectre variant 1 attacks from rogue guests are mitigated on a case by case basis in VM exit paths. Vulnerable code uses `nospec` accessor macros for “bounds clipping”, to avoid any usable disclosure gadgets. However, this may not cover all variant 1 attack vectors.

For Spectre variant 2 attacks from rogue guests to the kernel, the Linux kernel uses `retpoline` or Enhanced IBRS to prevent consumption of poisoned entries in branch target buffer left by rogue guests. It also flushes the return stack buffer on every VM exit to prevent a return stack buffer underflow so poisoned branch target buffer could be used, or attacker guests leaving poisoned entries in the return stack buffer.

To mitigate guest-to-guest attacks in the same CPU hardware thread, the branch target buffer is sanitized by flushing before switching to a new guest on a CPU.

The above mitigations are turned on by default on vulnerable CPUs.

To mitigate guest-to-guest attacks from sibling thread when SMT is in use, an untrusted guest running in the sibling thread can have its indirect branch speculation disabled by administrator via `prctl()`.

The kernel also allows guests to use any microcode based mitigation they choose to use (such as IBPB or STIBP on x86) to protect themselves.

### 5.1.9 Mitigation control on the kernel command line

Spectre variant 2 mitigation can be disabled or force enabled at the kernel command line.

`nospectre_v1`

[X86,PPC] Disable mitigations for Spectre Variant 1 (bounds check bypass). With this option data leaks are possible in the system.

`nospectre_v2`

[X86] Disable all mitigations for the Spectre variant 2 (indirect branch prediction) vulnerability. System may allow data leaks with this option, which is equivalent to `spectre_v2=off`.

`spectre_v2=`

[X86] Control mitigation of Spectre variant 2 (indirect branch speculation) vulnerability. The default operation protects the kernel from user space attacks.

**on** unconditionally enable, implies `spectre_v2_user=on`

**off** unconditionally disable, implies `spectre_v2_user=off`

**auto** kernel detects whether your CPU model is vulnerable

Selecting ‘on’ will, and ‘auto’ may, choose a mitigation method at run time according to the CPU, the available microcode, the setting of the `CONFIG_RETPOLINE` configuration option, and the compiler with which the kernel was built.

Selecting ‘on’ will also enable the mitigation against user space to user space task attacks.

Selecting ‘off’ will disable both the kernel and the user space protections.

Specific mitigations can also be selected manually:

**retpoline** replace indirect branches

**retpoline,generic** google’ s original retpoline

**retpoline,amd** AMD-specific minimal thunk

Not specifying this option is equivalent to `spectre_v2=auto`.

For user space mitigation:

`spectre_v2_user=`

[X86] Control mitigation of Spectre variant 2 (indirect branch speculation) vulnerability between user space tasks

**on** Unconditionally enable mitigations. Is enforced by `spectre_v2=on`

**off** Unconditionally disable mitigations. Is enforced by `spectre_v2=off`

**prctl** Indirect branch speculation is enabled, but mitigation can be enabled via `prctl` per thread. The mitigation control state is inherited on fork.

**prctl,ibpb** Like “`prctl`” above, but only STIBP is controlled per thread. IBPB is issued always when switching between different user space processes.

**seccomp** Same as “`prctl`” above, but all `seccomp` threads will enable the mitigation unless they explicitly opt out.

**seccomp,ibpb** Like “`seccomp`” above, but only STIBP is controlled per thread. IBPB is issued always when switching between different user space processes.

**auto** Kernel selects the mitigation depending on the available CPU features and vulnerability.

Default mitigation: If `CONFIG_SECCOMP=y` then “`seccomp`”, otherwise “`prctl`”

Not specifying this option is equivalent to `spectre_v2_user=auto`.

In general the kernel by default selects reasonable mitigations for the current CPU. To disable Spectre variant 2 mitigations, boot with `spectre_v2=off`. Spectre variant 1 mitigations cannot be disabled.

### 5.1.10 Mitigation selection guide

#### 1. Trusted userspace

If all userspace applications are from trusted sources and do not execute externally supplied untrusted code, then the mitigations can be disabled.

#### 2. Protect sensitive programs

For security-sensitive programs that have secrets (e.g. crypto keys), protection against Spectre variant 2 can be put in place by disabling indirect branch speculation when the program is running (See [Documentation/userspace-api/spec\\_ctrl.rst](#)).

#### 3. Sandbox untrusted programs

Untrusted programs that could be a source of attacks can be cordoned off by disabling their indirect branch speculation when they are run (See [Documentation/userspace-api/spec\\_ctrl.rst](#)). This prevents untrusted programs from polluting the branch target buffer. All programs running in SECCOMP sandboxes have indirect branch speculation restricted by default. This behavior can be changed via the kernel command line and `sysfs` control files. See Mitigation control on the kernel command line.

### 3. High security mode

All Spectre variant 2 mitigations can be forced on at boot time for all programs (See the “on” option in Mitigation control on the kernel command line). This will add overhead as indirect branch speculations for all programs will be restricted.

On x86, branch target buffer will be flushed with IBPB when switching to a new program. STIBP is left on all the time to protect programs against variant 2 attacks originating from programs running on sibling threads.

Alternatively, STIBP can be used only when running programs whose indirect branch speculation is explicitly disabled, while IBPB is still used all the time when switching to a new program to clear the branch target buffer (See “ibpb” option in Mitigation control on the kernel command line). This “ibpb” option has less performance cost than the “on” option, which leaves STIBP on all the time.

#### 5.1.11 References on Spectre

Intel white papers:

- [1] [Intel analysis of speculative execution side channels.](#)
- [2] [Bounds check bypass.](#)
- [3] [Deep dive: Retpoline: A branch target injection mitigation.](#)
- [4] [Deep Dive: Single Thread Indirect Branch Predictors.](#)

AMD white papers:

- [5] [AMD64 technology indirect branch control extension.](#)
- [6] [Software techniques for managing speculation on AMD processors.](#)

ARM white papers:

- [7] [Cache speculation side-channels.](#)
- [8] [Cache speculation issues update.](#)

Google white paper:

- [9] [Retpoline: a software construct for preventing branch-target-injection.](#)

MIPS white paper:

- [10] [MIPS: response on speculative execution and side channel vulnerabilities.](#)

Academic papers:

- [11] [Spectre Attacks: Exploiting Speculative Execution.](#)
- [12] [NetSpectre: Read Arbitrary Memory over Network.](#)
- [13] [Spectre Returns! Speculation Attacks using the Return Stack Buffer.](#)

## 5.2 L1TF - L1 Terminal Fault

L1 Terminal Fault is a hardware vulnerability which allows unprivileged speculative access to data which is available in the Level 1 Data Cache when the page table entry controlling the virtual address, which is used for the access, has the Present bit cleared or other reserved bits set.

### 5.2.1 Affected processors

This vulnerability affects a wide range of Intel processors. The vulnerability is not present on:

- Processors from AMD, Centaur and other non Intel vendors
- Older processor models, where the CPU family is < 6
- A range of Intel ATOM processors (Cedarview, Cloverview, Lincroft, Penwell, Pineview, Silvermont, Airmont, Merrifield)
- The Intel XEON PHI family
- Intel processors which have the ARCH\_CAP\_RDCL\_NO bit set in the IA32\_ARCH\_CAPABILITIES MSR. If the bit is set the CPU is not affected by the Meltdown vulnerability either. These CPUs should become available by end of 2018.

Whether a processor is affected or not can be read out from the L1TF vulnerability file in sysfs. See L1TF system information.

### 5.2.2 Related CVEs

The following CVE entries are related to the L1TF vulnerability:

CVE-2018-3615	L1 Terminal Fault	SGX related aspects
CVE-2018-3620	L1 Terminal Fault	OS, SMM related aspects
CVE-2018-3646	L1 Terminal Fault	Virtualization related aspects

### 5.2.3 Problem

If an instruction accesses a virtual address for which the relevant page table entry (PTE) has the Present bit cleared or other reserved bits set, then speculative execution ignores the invalid PTE and loads the referenced data if it is present in the Level 1 Data Cache, as if the page referenced by the address bits in the PTE was still present and accessible.

While this is a purely speculative mechanism and the instruction will raise a page fault when it is retired eventually, the pure act of loading the data and making it available to other speculative instructions opens up the opportunity for side channel attacks to unprivileged malicious code, similar to the Meltdown attack.

While Meltdown breaks the user space to kernel space protection, L1TF allows to attack any physical memory address in the system and the attack works across

all protection domains. It allows an attack of SGX and also works from inside virtual machines because the speculation bypasses the extended page table (EPT) protection mechanism.

## **5.2.4 Attack scenarios**

### **1. Malicious user space**

Operating Systems store arbitrary information in the address bits of a PTE which is marked non present. This allows a malicious user space application to attack the physical memory to which these PTEs resolve. In some cases user-space can maliciously influence the information encoded in the address bits of the PTE, thus making attacks more deterministic and more practical.

The Linux kernel contains a mitigation for this attack vector, PTE inversion, which is permanently enabled and has no performance impact. The kernel ensures that the address bits of PTEs, which are not marked present, never point to cacheable physical memory space.

A system with an up to date kernel is protected against attacks from malicious user space applications.

### **2. Malicious guest in a virtual machine**

The fact that L1TF breaks all domain protections allows malicious guest OSes, which can control the PTEs directly, and malicious guest user space applications, which run on an unprotected guest kernel lacking the PTE inversion mitigation for L1TF, to attack physical host memory.

A special aspect of L1TF in the context of virtualization is symmetric multi threading (SMT). The Intel implementation of SMT is called HyperThreading. The fact that Hyperthreads on the affected processors share the L1 Data Cache (L1D) is important for this. As the flaw allows only to attack data which is present in L1D, a malicious guest running on one Hyperthread can attack the data which is brought into the L1D by the context which runs on the sibling Hyperthread of the same physical core. This context can be host OS, host user space or a different guest.

If the processor does not support Extended Page Tables, the attack is only possible, when the hypervisor does not sanitize the content of the effective (shadow) page tables.

While solutions exist to mitigate these attack vectors fully, these mitigations are not enabled by default in the Linux kernel because they can affect performance significantly. The kernel provides several mechanisms which can be utilized to address the problem depending on the deployment scenario. The mitigations, their protection scope and impact are described in the next sections.

The default mitigations and the rationale for choosing them are explained at the end of this document. See Default mitigations.

### 5.2.5 L1TF system information

The Linux kernel provides a sysfs interface to enumerate the current L1TF status of the system: whether the system is vulnerable, and which mitigations are active. The relevant sysfs file is:

`/sys/devices/system/cpu/vulnerabilities/l1tf`

The possible values in this file are:

'Not affected'	The processor is not vulnerable
'Mitigation: PTE Inversion'	The host protection is active

If KVM/VMX is enabled and the processor is vulnerable then the following information is appended to the 'Mitigation: PTE Inversion' part:

- SMT status:

'VMX: SMT vulnerable'	SMT is enabled
'VMX: SMT disabled'	SMT is disabled

- L1D Flush mode:

'L1D vulnerable'	L1D flushing is disabled
'L1D conditional cache flushes'	L1D flush is conditionally enabled
'L1D cache flushes'	L1D flush is unconditionally enabled

The resulting grade of protection is discussed in the following sections.

### 5.2.6 Host mitigation mechanism

The kernel is unconditionally protected against L1TF attacks from malicious user space running on the host.

### 5.2.7 Guest mitigation mechanisms

#### 1. L1D flush on VMENTER

To make sure that a guest cannot attack data which is present in the L1D the hypervisor flushes the L1D before entering the guest.

Flushing the L1D evicts not only the data which should not be accessed by a potentially malicious guest, it also flushes the guest data. Flushing the L1D has a performance impact as the processor has to bring the flushed guest data back into the L1D. Depending on the frequency of VMEXIT/VMENTER and the type of computations in the guest performance degradation in the range of 1% to 50% has been observed. For scenarios where guest VMEXIT/VMENTER are rare the performance

impact is minimal. Virtio and mechanisms like posted interrupts are designed to confine the VMEXITs to a bare minimum, but specific configurations and application scenarios might still suffer from a high VMEXIT rate.

**The kernel provides two L1D flush modes:**

- conditional ( 'cond' )
- unconditional ( 'always' )

The conditional mode avoids L1D flushing after VMEXITs which execute only audited code paths before the corresponding VMENTER. These code paths have been verified that they cannot expose secrets or other interesting data to an attacker, but they can leak information about the address space layout of the hypervisor.

Unconditional mode flushes L1D on all VMENTER invocations and provides maximum protection. It has a higher overhead than the conditional mode. The overhead cannot be quantified correctly as it depends on the workload scenario and the resulting number of VMEXITs.

The general recommendation is to enable L1D flush on VMENTER. The kernel defaults to conditional mode on affected processors.

**Note**, that L1D flush does not prevent the SMT problem because the sibling thread will also bring back its data into the L1D which makes it attackable again.

L1D flush can be controlled by the administrator via the kernel command line and sysfs control files. See Mitigation control on the kernel command line and Mitigation control for KVM - module parameter.

## 2. Guest VCPU confinement to dedicated physical cores

To address the SMT problem, it is possible to make a guest or a group of guests affine to one or more physical cores. The proper mechanism for that is to utilize exclusive cpusets to ensure that no other guest or host tasks can run on these cores.

If only a single guest or related guests run on sibling SMT threads on the same physical core then they can only attack their own memory and restricted parts of the host memory.

Host memory is attackable, when one of the sibling SMT threads runs in host OS (hypervisor) context and the other in guest context. The amount of valuable information from the host OS context depends on the context which the host OS executes, i.e. interrupts, soft interrupts and kernel threads. The amount of valuable data from these contexts cannot be declared as non-interesting for an attacker without deep inspection of the code.

**Note**, that assigning guests to a fixed set of physical cores affects the ability of the scheduler to do load balancing and might have negative effects on CPU utilization depending on the hosting scenario. Disabling SMT might be a viable alternative for particular scenarios.

For further information about confining guests to a single or to a group of cores consult the cpusets documentation:

<https://www.kernel.org/doc/Documentation/admin-guide/cgroup-v1/cpusets.rst>

### 3. Interrupt affinity

Interrupts can be made affine to logical CPUs. This is not universally true because there are types of interrupts which are truly per CPU interrupts, e.g. the local timer interrupt. Aside of that multi queue devices affine their interrupts to single CPUs or groups of CPUs per queue without allowing the administrator to control the affinities.

Moving the interrupts, which can be affinity controlled, away from CPUs which run untrusted guests, reduces the attack vector space.

Whether the interrupts with are affine to CPUs, which run untrusted guests, provide interesting data for an attacker depends on the system configuration and the scenarios which run on the system. While for some of the interrupts it can be assumed that they won't expose interesting information beyond exposing hints about the host OS memory layout, there is no way to make general assumptions.

Interrupt affinity can be controlled by the administrator via the `/proc/irq/$NR/smp_affinity[_list]` files. Limited documentation is available at:

<https://www.kernel.org/doc/Documentation/core-api/irq/irq-affinity.rst>

### 4. SMT control

To prevent the SMT issues of L1TF it might be necessary to disable SMT completely. Disabling SMT can have a significant performance impact, but the impact depends on the hosting scenario and the type of workloads. The impact of disabling SMT needs also to be weighted against the impact of other mitigation solutions like confining guests to dedicated cores.

The kernel provides a sysfs interface to retrieve the status of SMT and to control it. It also provides a kernel command line interface to control SMT.

The kernel command line interface consists of the following options:

nosmt	Affects the bring up of the secondary CPUs during boot. The kernel tries to bring all present CPUs online during the boot process. “nosmt” makes sure that from each physical core only one - the so called primary (hyper) thread is activated. Due to a design flaw of Intel processors related to Machine Check Exceptions the non primary siblings have to be brought up at least partially and are then shut down again. “nosmt” can be undone via the sysfs interface.
nosmtforceoff	Has the same effect as “nosmt” but it does not allow to undo the SMT disable via the sysfs interface.

The sysfs interface provides two files:

- /sys/devices/system/cpu/smt/control
- /sys/devices/system/cpu/smt/active

/sys/devices/system/cpu/smt/control:

This file allows to read out the SMT control state and provides the ability to disable or (re)enable SMT. The possible states are:

on	SMT is supported by the CPU and enabled. All logical CPUs can be onlined and offlined without restrictions.
off	SMT is supported by the CPU and disabled. Only the so called primary SMT threads can be onlined and offlined without restrictions. An attempt to online a non-primary sibling is rejected
forceoff	Same as ‘off’ but the state cannot be controlled. Attempts to write to the control file are rejected.
not-supported	The processor does not support SMT. It’s therefore not affected by the SMT implications of L1TF. Attempts to write to the control file are rejected.

The possible states which can be written into this file to control SMT state are:

- on
- off
- forceoff

/sys/devices/system/cpu/smt/active:

This file reports whether SMT is enabled and active, i.e. if on any physical core two or more sibling threads are online.

SMT control is also possible at boot time via the `l1tf` kernel command line parameter in combination with L1D flush control. See Mitigation control on the kernel command line.

## 5. Disabling EPT

Disabling EPT for virtual machines provides full mitigation for L1TF even with SMT enabled, because the effective page tables for guests are managed and sanitized by the hypervisor. Though disabling EPT has a significant performance impact especially when the Meltdown mitigation KPTI is enabled.

EPT can be disabled in the hypervisor via the ‘kvm-intel.ept’ parameter.

There is ongoing research and development for new mitigation mechanisms to address the performance impact of disabling SMT or EPT.

### 5.2.8 Mitigation control on the kernel command line

The kernel command line allows to control the L1TF mitigations at boot time with the option “l1tf=” . The valid arguments for this option are:

full	Provides all available mitigations for the L1TF vulnerability. Disables SMT and enables all mitigations in the hypervisors, i.e. unconditional L1D flushing SMT control and L1D flush control via the sysfs interface is still possible after boot. Hypervisors will issue a warning when the first VM is started in a potentially insecure configuration, i.e. SMT enabled or L1D flush disabled.
full,force	Same as ‘full’ , but disables SMT and L1D flush runtime control. Implies the ‘nosmt=force’ command line option. (i.e. sysfs control of SMT is disabled.)
flush	Leaves SMT enabled and enables the default hypervisor mitigation, i.e. conditional L1D flushing SMT control and L1D flush control via the sysfs interface is still possible after boot. Hypervisors will issue a warning when the first VM is started in a potentially insecure configuration, i.e. SMT enabled or L1D flush disabled.
flush,disable	Disables SMT and enables the default hypervisor mitigation, i.e. conditional L1D flushing. SMT control and L1D flush control via the sysfs interface is still possible after boot. Hypervisors will issue a warning when the first VM is started in a potentially insecure configuration, i.e. SMT enabled or L1D flush disabled.
flush,nowarn	Same as ‘flush’ , but hypervisors will not warn when a VM is started in a potentially insecure configuration.
off	Disables hypervisor mitigations and doesn’ t emit any warnings. It also drops the swap size and available RAM limit restrictions on both hypervisor and bare metal.

The default is ‘flush’. For details about L1D flushing see 1. L1D flush on VMENTER.

### 5.2.9 Mitigation control for KVM - module parameter

The KVM hypervisor mitigation mechanism, flushing the L1D cache when entering a guest, can be controlled with a module parameter.

The option/parameter is “kvm-intel.vmentry\_l1d\_flush=” . It takes the following arguments:

al- ways	L1D cache flush on every VMENTER.
cond	Flush L1D on VMENTER only when the code between VMEXIT and VMENTER can leak host memory which is considered interesting for an attacker. This still can leak host memory which allows e.g. to determine the hosts address space layout.
never	Disables the mitigation

The parameter can be provided on the kernel command line, as a module parameter when loading the modules and at runtime modified via the sysfs file:

```
/sys/module/kvm_intel/parameters/vmentry_l1d_flush
```

The default is ‘cond’ . If ‘l1tf=full,force’ is given on the kernel command line, then ‘always’ is enforced and the kvm-intel.vmentry\_l1d\_flush module parameter is ignored and writes to the sysfs file are rejected.

### 5.2.10 Mitigation selection guide

#### 1. No virtualization in use

The system is protected by the kernel unconditionally and no further action is required.

#### 2. Virtualization with trusted guests

If the guest comes from a trusted source and the guest OS kernel is guaranteed to have the L1TF mitigations in place the system is fully protected against L1TF and no further action is required.

To avoid the overhead of the default L1D flushing on VMENTER the administrator can disable the flushing via the kernel command line and sysfs control files. See Mitigation control on the kernel command line and Mitigation control for KVM - module parameter.

### 3. Virtualization with untrusted guests

#### 3.1. SMT not supported or disabled

If SMT is not supported by the processor or disabled in the BIOS or by the kernel, it's only required to enforce L1D flushing on VMENTER.

Conditional L1D flushing is the default behaviour and can be tuned. See Mitigation control on the kernel command line and Mitigation control for KVM - module parameter.

#### 3.2. EPT not supported or disabled

If EPT is not supported by the processor or disabled in the hypervisor, the system is fully protected. SMT can stay enabled and L1D flushing on VMENTER is not required.

EPT can be disabled in the hypervisor via the 'kvm-intel.ept' parameter.

#### 3.3. SMT and EPT supported and active

If SMT and EPT are supported and active then various degrees of mitigations can be employed:

- L1D flushing on VMENTER:

L1D flushing on VMENTER is the minimal protection requirement, but it is only potent in combination with other mitigation methods.

Conditional L1D flushing is the default behaviour and can be tuned. See Mitigation control on the kernel command line and Mitigation control for KVM - module parameter.

- Guest confinement:

Confinement of guests to a single or a group of physical cores which are not running any other processes, can reduce the attack surface significantly, but interrupts, soft interrupts and kernel threads can still expose valuable data to a potential attacker. See 2. Guest VCPU confinement to dedicated physical cores.

- Interrupt isolation:

Isolating the guest CPUs from interrupts can reduce the attack surface further, but still allows a malicious guest to explore a limited amount of host physical memory. This can at least be used to gain knowledge about the host address space layout. The interrupts which have a fixed affinity to the CPUs which run the untrusted guests can depending on the scenario still trigger soft interrupts and schedule kernel threads which might expose valuable information. See 3. Interrupt affinity.

The above three mitigation methods combined can provide protection to a certain degree, but the risk of the remaining attack surface has to be carefully analyzed. For full protection the following methods are available:

- Disabling SMT:

Disabling SMT and enforcing the L1D flushing provides the maximum amount of protection. This mitigation is not depending on any of the above mitigation methods.

SMT control and L1D flushing can be tuned by the command line parameters ‘nosmt’ , ‘l1tf’ , ‘kvm-intel.vmentry\_l1d\_flush’ and at run time with the matching sysfs control files. See 4. SMT control, Mitigation control on the kernel command line and Mitigation control for KVM - module parameter.

- Disabling EPT:

Disabling EPT provides the maximum amount of protection as well. It is not depending on any of the above mitigation methods. SMT can stay enabled and L1D flushing is not required, but the performance impact is significant.

EPT can be disabled in the hypervisor via the ‘kvm-intel.ept’ parameter.

### 3.4. Nested virtual machines

When nested virtualization is in use, three operating systems are involved: the bare metal hypervisor, the nested hypervisor and the nested virtual machine. VMENTER operations from the nested hypervisor into the nested guest will always be processed by the bare metal hypervisor. If KVM is the bare metal hypervisor it will:

- Flush the L1D cache on every switch from the nested hypervisor to the nested virtual machine, so that the nested hypervisor’s secrets are not exposed to the nested virtual machine;
- Flush the L1D cache on every switch from the nested virtual machine to the nested hypervisor; this is a complex operation, and flushing the L1D cache avoids that the bare metal hypervisor’s secrets are exposed to the nested virtual machine;
- Instruct the nested hypervisor to not perform any L1D cache flush. This is an optimization to avoid double L1D flushing.

#### 5.2.11 Default mitigations

The kernel default mitigations for vulnerable processors are:

- PTE inversion to protect against malicious user space. This is done unconditionally and cannot be controlled. The swap storage is limited to ~16TB.
- L1D conditional flushing on VMENTER when EPT is enabled for a guest.

The kernel does not by default enforce the disabling of SMT, which leaves SMT systems vulnerable when running untrusted guests with EPT enabled.

The rationale for this choice is:

- Force disabling SMT can break existing setups, especially with unattended updates.
- If regular users run untrusted guests on their machine, then L1TF is just an add on to other malware which might be embedded in an untrusted guest, e.g. spam-bots or attacks on the local network.

There is no technical way to prevent a user from running untrusted code on their machines blindly.

- It's technically extremely unlikely and from today's knowledge even impossible that L1TF can be exploited via the most popular attack mechanisms like JavaScript because these mechanisms have no way to control PTEs. If this would be possible and not other mitigation would be possible, then the default might be different.
- The administrators of cloud and hosting setups have to carefully analyze the risk for their scenarios and make the appropriate mitigation choices, which might even vary across their deployed machines and also result in other changes of their overall setup. There is no way for the kernel to provide a sensible default for this kind of scenarios.

## 5.3 MDS - Microarchitectural Data Sampling

Microarchitectural Data Sampling is a hardware vulnerability which allows unprivileged speculative access to data which is available in various CPU internal buffers.

### 5.3.1 Affected processors

This vulnerability affects a wide range of Intel processors. The vulnerability is not present on:

- Processors from AMD, Centaur and other non Intel vendors
- Older processor models, where the CPU family is < 6
- Some Atoms (Bonnell, Saltwell, Goldmont, GoldmontPlus)
- Intel processors which have the ARCH\_CAP\_MDS\_NO bit set in the IA32\_ARCH\_CAPABILITIES MSR.

Whether a processor is affected or not can be read out from the MDS vulnerability file in sysfs. See MDS system information.

Not all processors are affected by all variants of MDS, but the mitigation is identical for all of them so the kernel treats them as a single vulnerability.

### 5.3.2 Related CVEs

The following CVE entries are related to the MDS vulnerability:

CVE-2018-12126	MS-BDS	Microarchitectural Store Buffer Data Sampling
CVE-2018-12130	MF-BDS	Microarchitectural Fill Buffer Data Sampling
CVE-2018-12127	MLPDS	Microarchitectural Load Port Data Sampling
CVE-2019-11091	MD-SUM	Microarchitectural Data Sampling Uncacheable Memory

### 5.3.3 Problem

When performing store, load, L1 refill operations, processors write data into temporary microarchitectural structures (buffers). The data in the buffer can be forwarded to load operations as an optimization.

Under certain conditions, usually a fault/assist caused by a load operation, data unrelated to the load memory address can be speculatively forwarded from the buffers. Because the load operation causes a fault or assist and its result will be discarded, the forwarded data will not cause incorrect program execution or state changes. But a malicious operation may be able to forward this speculative data to a disclosure gadget which allows in turn to infer the value via a cache side channel attack.

Because the buffers are potentially shared between Hyper-Threads cross Hyper-Thread attacks are possible.

Deeper technical information is available in the MDS specific x86 architecture section: [Documentation/x86/mds.rst](#).

### 5.3.4 Attack scenarios

Attacks against the MDS vulnerabilities can be mounted from malicious non privileged user space applications running on hosts or guest. Malicious guest OSes can obviously mount attacks as well.

Contrary to other speculation based vulnerabilities the MDS vulnerability does not allow the attacker to control the memory target address. As a consequence the attacks are purely sampling based, but as demonstrated with the TLBleed attack samples can be postprocessed successfully.

### Web-Browsers

It's unclear whether attacks through Web-Browsers are possible at all. The exploitation through Java-Script is considered very unlikely, but other widely used web technologies like Webassembly could possibly be abused.

#### 5.3.5 MDS system information

The Linux kernel provides a sysfs interface to enumerate the current MDS status of the system: whether the system is vulnerable, and which mitigations are active. The relevant sysfs file is:

```
/sys/devices/system/cpu/vulnerabilities/mds
```

The possible values in this file are:

'Not affected'	The processor is not vulnerable
'Vulnerable'	The processor is vulnerable, but no mitigation enabled
'Vulnerable: Clear CPU buffers attempted, no microcode'	The processor is vulnerable but microcode is not updated. The mitigation is enabled on a best effort basis. See Best effort mitigation mode
'Mitigation: Clear CPU buffers'	The processor is vulnerable and the CPU buffer clearing mitigation is enabled.

If the processor is vulnerable then the following information is appended to the above information:

'SMT vulnerable'	SMT is enabled
'SMT mitigated'	SMT is enabled and mitigated
'SMT disabled'	SMT is disabled
'SMT Host state unknown'	Kernel runs in a VM, Host SMT state unknown

#### Best effort mitigation mode

If the processor is vulnerable, but the availability of the microcode based mitigation mechanism is not advertised via CPUID the kernel selects a best effort mitigation mode. This mode invokes the mitigation instructions without a guarantee that they clear the CPU buffers.

This is done to address virtualization scenarios where the host has the microcode update applied, but the hypervisor is not yet updated to expose the CPUID to the guest. If the host has updated microcode the protection takes effect otherwise a few cpu cycles are wasted pointlessly.

The state in the mds sysfs file reflects this situation accordingly.

### 5.3.6 Mitigation mechanism

The kernel detects the affected CPUs and the presence of the microcode which is required.

If a CPU is affected and the microcode is available, then the kernel enables the mitigation by default. The mitigation can be controlled at boot time via a kernel command line option. See Mitigation control on the kernel command line.

#### CPU buffer clearing

The mitigation for MDS clears the affected CPU buffers on return to user space and when entering a guest.

If SMT is enabled it also clears the buffers on idle entry when the CPU is only affected by MSBDS and not any other MDS variant, because the other variants cannot be protected against cross Hyper-Thread attacks.

For CPUs which are only affected by MSBDS the user space, guest and idle transition mitigations are sufficient and SMT is not affected.

#### Virtualization mitigation

The protection for host to guest transition depends on the L1TF vulnerability of the CPU:

- CPU is affected by L1TF:

If the L1D flush mitigation is enabled and up to date microcode is available, the L1D flush mitigation is automatically protecting the guest transition.

If the L1D flush mitigation is disabled then the MDS mitigation is invoked explicit when the host MDS mitigation is enabled.

For details on L1TF and virtualization see: Documentation/admin-guide/hw-vuln/l1tf.rst.

- CPU is not affected by L1TF:

CPU buffers are flushed before entering the guest when the host MDS mitigation is enabled.

The resulting MDS protection matrix for the host to guest transition:

L1TF	MDS	VMX-L1FLUSH	Host MDS	MDS-State
Don' t care	No	Don' t care	N/A	Not affected
Yes	Yes	Disabled	Off	Vulnerable
Yes	Yes	Disabled	Full	Mitigated
Yes	Yes	Enabled	Don' t care	Mitigated
No	Yes	N/A	Off	Vulnerable
No	Yes	N/A	Full	Mitigated

This only covers the host to guest transition, i.e. prevents leakage from host to guest, but does not protect the guest internally. Guests need to have their own protections.

### XEON PHI specific considerations

The XEON PHI processor family is affected by MSBDS which can be exploited cross Hyper-Threads when entering idle states. Some XEON PHI variants allow to use MWAIT in user space (Ring 3) which opens an potential attack vector for malicious user space. The exposure can be disabled on the kernel command line with the ‘ring3mwait=disable’ command line option.

XEON PHI is not affected by the other MDS variants and MSBDS is mitigated before the CPU enters a idle state. As XEON PHI is not affected by L1TF either disabling SMT is not required for full protection.

### SMT control

All MDS variants except MSBDS can be attacked cross Hyper-Threads. That means on CPUs which are affected by MFBDS or MLPDS it is necessary to disable SMT for full protection. These are most of the affected CPUs; the exception is XEON PHI, see XEON PHI specific considerations.

Disabling SMT can have a significant performance impact, but the impact depends on the type of workloads.

See the relevant chapter in the L1TF mitigation documentation for details: Documentation/admin-guide/hw-vuln/l1tf.rst.

### 5.3.7 Mitigation control on the kernel command line

The kernel command line allows to control the MDS mitigations at boot time with the option “mds=”. The valid arguments for this option are:

full	If the CPU is vulnerable, enable all available mitigations for the MDS vulnerability, CPU buffer clearing on exit to userspace and when entering a VM. Idle transitions are protected as well if SMT is enabled. It does not automatically disable SMT.
full,no_smt	The same as mds=full, with SMT disabled on vulnerable CPUs. This is the complete mitigation.
off	Disables MDS mitigations completely.

Not specifying this option is equivalent to “mds=full”. For processors that are affected by both TAA (TSX Asynchronous Abort) and MDS, specifying just “mds=off” without an accompanying “tsx\_async\_abort=off” will have no effect as the same mitigation is used for both vulnerabilities.

## 5.3.8 Mitigation selection guide

### 1. Trusted userspace

If all userspace applications are from a trusted source and do not execute untrusted code which is supplied externally, then the mitigation can be disabled.

### 2. Virtualization with trusted guests

The same considerations as above versus trusted user space apply.

### 3. Virtualization with untrusted guests

The protection depends on the state of the L1TF mitigations. See Virtualization mitigation.

If the MDS mitigation is enabled and SMT is disabled, guest to host and guest to guest attacks are prevented.

## 5.3.9 Default mitigations

The kernel default mitigations for vulnerable processors are:

- Enable CPU buffer clearing

The kernel does not by default enforce the disabling of SMT, which leaves SMT systems vulnerable when running untrusted code. The same rationale as for L1TF applies. See [Documentation/admin-guide/hw-vuln/l1tf.rst](#).

## 5.4 TAA - TSX Asynchronous Abort

TAA is a hardware vulnerability that allows unprivileged speculative access to data which is available in various CPU internal buffers by using asynchronous aborts within an Intel TSX transactional region.

### 5.4.1 Affected processors

This vulnerability only affects Intel processors that support Intel Transactional Synchronization Extensions (TSX) when the TAA\_NO bit (bit 8) is 0 in the IA32\_ARCH\_CAPABILITIES MSR. On processors where the MDS\_NO bit (bit 5) is 0 in the IA32\_ARCH\_CAPABILITIES MSR, the existing MDS mitigations also mitigate against TAA.

Whether a processor is affected or not can be read out from the TAA vulnerability file in sysfs. See TAA system information.

### 5.4.2 Related CVEs

The following CVE entry is related to this TAA issue:

CVE-2019-11135	TAA	TSX Asynchronous Abort (TAA) condition on some micro-processors utilizing speculative execution may allow an authenticated user to potentially enable information disclosure via a side channel with local access.
----------------	-----	--

### 5.4.3 Problem

When performing store, load or L1 refill operations, processors write data into temporary microarchitectural structures (buffers). The data in those buffers can be forwarded to load operations as an optimization.

Intel TSX is an extension to the x86 instruction set architecture that adds hardware transactional memory support to improve performance of multi-threaded software. TSX lets the processor expose and exploit concurrency hidden in an application due to dynamically avoiding unnecessary synchronization.

TSX supports atomic memory transactions that are either committed (success) or aborted. During an abort, operations that happened within the transactional region are rolled back. An asynchronous abort takes place, among other options, when a different thread accesses a cache line that is also used within the transactional region when that access might lead to a data race.

Immediately after an uncompleted asynchronous abort, certain speculatively executed loads may read data from those internal buffers and pass it to dependent operations. This can be then used to infer the value via a cache side channel attack.

Because the buffers are potentially shared between Hyper-Threads cross Hyper-Thread attacks are possible.

The victim of a malicious actor does not need to make use of TSX. Only the attacker needs to begin a TSX transaction and raise an asynchronous abort which in turn potentially leaks data stored in the buffers.

More detailed technical information is available in the TAA specific x86 architecture section: [Documentation/x86/tsx\\_async\\_abort.rst](#).

### 5.4.4 Attack scenarios

Attacks against the TAA vulnerability can be implemented from unprivileged applications running on hosts or guests.

As for MDS, the attacker has no control over the memory addresses that can be leaked. Only the victim is responsible for bringing data to the CPU. As a result, the malicious actor has to sample as much data as possible and then postprocess it to try to infer any useful information from it.

A potential attacker only has read access to the data. Also, there is no direct privilege escalation by using this technique.

### 5.4.5 TAA system information

The Linux kernel provides a sysfs interface to enumerate the current TAA status of mitigated systems. The relevant sysfs file is:

```
/sys/devices/system/cpu/vulnerabilities/tsx_async_abort
```

The possible values in this file are:

'Vulnerable'	The CPU is affected by this vulnerability and the microcode and kernel mitigation are not applied.
'Vulnerable: Clear CPU buffers attempted, no microcode'	The system tries to clear the buffers but the microcode might not support the operation.
'Mitigation: Clear CPU buffers'	The microcode has been updated to clear the buffers. TSX is still enabled.
'Mitigation: TSX disabled'	TSX is disabled.
'Not affected'	The CPU is not affected by this issue.

### Best effort mitigation mode

If the processor is vulnerable, but the availability of the microcode-based mitigation mechanism is not advertised via CPUID the kernel selects a best effort mitigation mode. This mode invokes the mitigation instructions without a guarantee that they clear the CPU buffers.

This is done to address virtualization scenarios where the host has the microcode update applied, but the hypervisor is not yet updated to expose the CPUID to the guest. If the host has updated microcode the protection takes effect; otherwise a few CPU cycles are wasted pointlessly.

The state in the `tsx_async_abort` sysfs file reflects this situation accordingly.

### 5.4.6 Mitigation mechanism

The kernel detects the affected CPUs and the presence of the microcode which is required. If a CPU is affected and the microcode is available, then the kernel enables the mitigation by default.

The mitigation can be controlled at boot time via a kernel command line option. See Mitigation control on the kernel command line.

## Virtualization mitigation

Affected systems where the host has TAA microcode and TAA is mitigated by having disabled TSX previously, are not vulnerable regardless of the status of the VMs.

In all other cases, if the host either does not have the TAA microcode or the kernel is not mitigated, the system might be vulnerable.

### 5.4.7 Mitigation control on the kernel command line

The kernel command line allows to control the TAA mitigations at boot time with the option “`tsx_async_abort=`”. The valid arguments for this option are:

off	This option disables the TAA mitigation on affected platforms. If the system has TSX enabled (see next parameter) and the CPU is affected, the system is vulnerable.
full	TAA mitigation is enabled. If TSX is enabled, on an affected system it will clear CPU buffers on ring transitions. On systems which are MDS-affected and deploy MDS mitigation, TAA is also mitigated. Specifying this option on those systems will have no effect.
full, no_smt	The same as <code>tsx_async_abort=full</code> , with SMT disabled on vulnerable CPUs that have TSX enabled. This is the complete mitigation. When TSX is disabled, SMT is not disabled because CPU is not vulnerable to cross-thread TAA attacks.

Not specifying this option is equivalent to “`tsx_async_abort=full`”. For processors that are affected by both TAA and MDS, specifying just “`tsx_async_abort=off`” without an accompanying “`mds=off`” will have no effect as the same mitigation is used for both vulnerabilities.

The kernel command line also allows to control the TSX feature using the parameter “`tsx=`” on CPUs which support TSX control. `MSR_IA32_TSX_CTRL` is used to control the TSX feature and the enumeration of the TSX feature bits (RTM and HLE) in `CPUID`.

The valid options are:

off	Disables TSX on the system. Note that this option takes effect only on newer CPUs which are not vulnerable to MDS, i.e., have <code>MSR_IA32_ARCH_CAPABILITIES.MDS_NO=1</code> and which get the new <code>IA32_TSX_CTRL</code> MSR through a microcode update. This new MSR allows for the reliable deactivation of the TSX functionality.
on	Enables TSX. Although there are mitigations for all known security vulnerabilities, TSX has been known to be an accelerator for several previous speculation-related CVEs, and so there may be unknown security risks associated with leaving it enabled.
auto	Disables TSX if <code>X86_BUG_TAA</code> is present, otherwise enables TSX on the system.

Not specifying this option is equivalent to “tsx=off” .

The following combinations of the “tsx\_async\_abort” and “tsx” are possible. For affected platforms tsx=auto is equivalent to tsx=off and the result will be:

tsx=on	tsx_async_abort=full	The system will use VERW to clear CPU buffers. Cross-thread attacks are still possible on SMT machines.
tsx=on	tsx_async_abort=auto	As full, no cross-thread attacks on SMT mitigated.
tsx=off	tsx_async_abort=full	The system is vulnerable.
tsx=off	tsx_async_abort=auto	TSX might be disabled if microcode provides a TSX control MSR. If so, system is not vulnerable.
tsx=off	tsx_async_abort=off	Diff, nosmt
tsx=off	tsx_async_abort=off	Diff

For unaffected platforms “tsx=on” and “tsx\_async\_abort=full” does not clear CPU buffers. For platforms without TSX control (MSR\_IA32\_ARCH\_CAPABILITIES.MDS\_NO=0) “tsx” command line argument has no effect.

For the affected platforms below table indicates the mitigation status for the combinations of CPUID bit MD\_CLEAR and IA32\_ARCH\_CAPABILITIES MSR bits MDS\_NO and TSX\_CTRL\_MSR.

MDS_NO	MD_CLEAR	TSX_CTRL_MSR	Status
0	0	0	Vulnerable (needs microcode)
0	1	0	MDS and TAA mitigated via VERW
1	1	0	MDS fixed, TAA vulnerable if TSX enabled because MD_CLEAR has no meaning and VERW is not guaranteed to clear buffers
1	X	1	MDS fixed, TAA can be mitigated by VERW or TSX_CTRL_MSR

## 5.4.8 Mitigation selection guide

### 1. Trusted userspace and guests

If all user space applications are from a trusted source and do not execute untrusted code which is supplied externally, then the mitigation can be disabled. The same applies to virtualized environments with trusted guests.

## 2. Untrusted userspace and guests

If there are untrusted applications or guests on the system, enabling TSX might allow a malicious actor to leak data from the host or from other processes running on the same physical core.

If the microcode is available and the TSX is disabled on the host, attacks are prevented in a virtualized environment as well, even if the VMs do not explicitly enable the mitigation.

### 5.4.9 Default mitigations

The kernel's default action for vulnerable processors is:

- Deploy TSX disable mitigation (`tsx_async_abort=full tsx=off`).

## 5.5 iTLB multihit

iTLB multihit is an erratum where some processors may incur a machine check error, possibly resulting in an unrecoverable CPU lockup, when an instruction fetch hits multiple entries in the instruction TLB. This can occur when the page size is changed along with either the physical address or cache type. A malicious guest running on a virtualized system can exploit this erratum to perform a denial of service attack.

### 5.5.1 Affected processors

Variations of this erratum are present on most Intel Core and Xeon processor models. The erratum is not present on:

- non-Intel processors
- Some Atoms (Airmont, Bonnell, Goldmont, GoldmontPlus, Saltwell, Silvermont)
- Intel processors that have the `PSCHANGE_MC_NO` bit set in the `IA32_ARCH_CAPABILITIES` MSR.

### 5.5.2 Related CVEs

The following CVE entry is related to this issue:

CVE-2018-12207	Machine Check Error Avoidance on Page Size Change
----------------	---

### 5.5.3 Problem

Privileged software, including OS and virtual machine managers (VMM), are in charge of memory management. A key component in memory management is the control of the page tables. Modern processors use virtual memory, a technique that creates the illusion of a very large memory for processors. This virtual space is split into pages of a given size. Page tables translate virtual addresses to physical addresses.

To reduce latency when performing a virtual to physical address translation, processors include a structure, called TLB, that caches recent translations. There are separate TLBs for instruction (iTLB) and data (dTLB).

Under this errata, instructions are fetched from a linear address translated using a 4 KB translation cached in the iTLB. Privileged software modifies the paging structure so that the same linear address using large page size (2 MB, 4 MB, 1 GB) with a different physical address or memory type. After the page structure modification but before the software invalidates any iTLB entries for the linear address, a code fetch that happens on the same linear address may cause a machine-check error which can result in a system hang or shutdown.

### 5.5.4 Attack scenarios

Attacks against the iTLB multihit erratum can be mounted from malicious guests in a virtualized system.

### 5.5.5 iTLB multihit system information

The Linux kernel provides a sysfs interface to enumerate the current iTLB multihit status of the system: whether the system is vulnerable and which mitigations are active. The relevant sysfs file is:

`/sys/devices/system/cpu/vulnerabilities/itlb_multihit`

The possible values in this file are:

Not affected	The processor is not vulnerable.
KVM: Mitigation: Split huge pages	Software changes mitigate this issue.
KVM: Vulnerable	The processor is vulnerable, but no mitigation enabled

### 5.5.6 Enumeration of the erratum

A new bit has been allocated in the IA32\_ARCH\_CAPABILITIES (PSCHANGE\_MC\_NO) msr and will be set on CPU' s which are mitigated against this issue.

IA32_ARCH_CAPABILITIES MSR	Not present	Possibly vulnerable,check model
IA32_ARCH_CAPABILITIES[PSCHANGE_MC_NO]	Not present	Possibly vulnerable,check model
IA32_ARCH_CAPABILITIES[PSCHANGE_MC_NO]	Not present	Not vulnerable

### 5.5.7 Mitigation mechanism

This erratum can be mitigated by restricting the use of large page sizes to non-executable pages. This forces all iTLB entries to be 4K, and removes the possibility of multiple hits.

In order to mitigate the vulnerability, KVM initially marks all huge pages as non-executable. If the guest attempts to execute in one of those pages, the page is broken down into 4K pages, which are then marked executable.

If EPT is disabled or not available on the host, KVM is in control of TLB flushes and the problematic situation cannot happen. However, the shadow EPT paging mechanism used by nested virtualization is vulnerable, because the nested guest can trigger multiple iTLB hits by modifying its own (non-nested) page tables. For simplicity, KVM will make large pages non-executable in all shadow paging modes.

### 5.5.8 Mitigation control on the kernel command line and KVM - module parameter

The KVM hypervisor mitigation mechanism for marking huge pages as non-executable can be controlled with a module parameter “nx\_huge\_pages=”. The kernel command line allows to control the iTLB multihit mitigations at boot time with the option “kvm.nx\_huge\_pages=” .

The valid arguments for these options are:

force	Mitigation is enabled. In this case, the mitigation implements non-executable huge pages in Linux kernel KVM module. All huge pages in the EPT are marked as non-executable. If a guest attempts to execute in one of those pages, the page is broken down into 4K pages, which are then marked executable.
off	Mitigation is disabled.
auto	Enable mitigation only if the platform is affected and the kernel was not booted with the “mitigations=off” command line parameter. This is the default option.

## 5.5.9 Mitigation selection guide

### 1. No virtualization in use

The system is protected by the kernel unconditionally and no further action is required.

### 2. Virtualization with trusted guests

If the guest comes from a trusted source, you may assume that the guest will not attempt to maliciously exploit these errata and no further action is required.

### 3. Virtualization with untrusted guests

If the guest comes from an untrusted source, the guest host kernel will need to apply iTLB multihit mitigation via the kernel command line or kvm module parameter.

## 5.6 SRBDS - Special Register Buffer Data Sampling

SRBDS is a hardware vulnerability that allows MDS - Microarchitectural Data Sampling techniques to infer values returned from special register accesses. Special register accesses are accesses to off core registers. According to Intel's evaluation, the special register reads that have a security expectation of privacy are RDRAND, RDSEED and SGX EGETKEY.

When RDRAND, RDSEED and EGETKEY instructions are used, the data is moved to the core through the special register mechanism that is susceptible to MDS attacks.

### 5.6.1 Affected processors

Core models (desktop, mobile, Xeon-E3) that implement RDRAND and/or RDSEED may be affected.

A processor is affected by SRBDS if its Family\_Model and stepping is in the following list, with the exception of the listed processors exporting MDS\_NO while Intel TSX is available yet not enabled. The latter class of processors are only affected when Intel TSX is enabled by software using TSX\_CTRL\_MSR otherwise they are not affected.

common name	Family_Model	Stepping
IvyBridge	06_3AH	All
Haswell	06_3CH	All
Haswell_L	06_45H	All
Haswell_G	06_46H	All
Broadwell_G	06_47H	All
Broadwell	06_3DH	All
Skylake_L	06_4EH	All
Skylake	06_5EH	All
Kabylake_L	06_8EH	<= 0xC
Kabylake	06_9EH	<= 0xD

### 5.6.2 Related CVEs

The following CVE entry is related to this SRBDS issue:

CVE-2020-0543	SRBDS	Special Register Buffer Data Sampling
---------------	-------	---------------------------------------

### 5.6.3 Attack scenarios

An unprivileged user can extract values returned from RDRAND and RDSEED executed on another core or sibling thread using MDS techniques.

### 5.6.4 Mitigation mechanism

Intel will release microcode updates that modify the RDRAND, RDSEED, and EGETKEY instructions to overwrite secret special register data in the shared staging buffer before the secret data can be accessed by another logical processor.

During execution of the RDRAND, RDSEED, or EGETKEY instructions, off-core accesses from other logical processors will be delayed until the special register read is complete and the secret data in the shared staging buffer is overwritten.

This has three effects on performance:

1. RDRAND, RDSEED, or EGETKEY instructions have higher latency.
2. Executing RDRAND at the same time on multiple logical processors will be serialized, resulting in an overall reduction in the maximum RDRAND bandwidth.
3. Executing RDRAND, RDSEED or EGETKEY will delay memory accesses from other logical processors that miss their core caches, with an impact similar to legacy locked cache-line-split accesses.

The microcode updates provide an opt-out mechanism (RNGDS\_MITG\_DIS) to disable the mitigation for RDRAND and RDSEED instructions executed outside of Intel Software Guard Extensions (Intel SGX) enclaves. On logical processors that disable the mitigation using this opt-out mechanism, RDRAND and RDSEED do not take longer to execute and do not impact performance of sibling logical processors

memory accesses. The opt-out mechanism does not affect Intel SGX enclaves (including execution of RDRAND or RDSEED inside an enclave, as well as EGETKEY execution).

### 5.6.5 IA32\_MCU\_OPT\_CTRL MSR Definition

Along with the mitigation for this issue, Intel added a new thread-scope IA32\_MCU\_OPT\_CTRL MSR, (address 0x123). The presence of this MSR and RNGDS\_MITG\_DIS (bit 0) is enumerated by CPUID.(EAX=07H,ECX=0).EDX[SRBDS\_CTRL = 9]=1. This MSR is introduced through the microcode update.

Setting IA32\_MCU\_OPT\_CTRL[0] (RNGDS\_MITG\_DIS) to 1 for a logical processor disables the mitigation for RDRAND and RDSEED executed outside of an Intel SGX enclave on that logical processor. Opting out of the mitigation for a particular logical processor does not affect the RDRAND and RDSEED mitigations for other logical processors.

Note that inside of an Intel SGX enclave, the mitigation is applied regardless of the value of RNGDS\_MITG\_DS.

### 5.6.6 Mitigation control on the kernel command line

The kernel command line allows control over the SRBDS mitigation at boot time with the option “`srbds=`”. The option for this is:

off	This option disables SRBDS mitigation for RDRAND and RDSEED on affected platforms.
-----	--

### 5.6.7 SRBDS System Information

The Linux kernel provides vulnerability status information through sysfs. For SRBDS this can be accessed by the following sysfs file: `/sys/devices/system/cpu/vulnerabilities/srbds`

The possible values contained in this file are:

Not affected	Processor not vulnerable
Vulnerable	Processor vulnerable and mitigation disabled
Vulnerable: No microcode	Processor vulnerable and microcode is missing mitigation
Mitigation: Microcode	Processor is vulnerable and mitigation is in effect.
Mitigation: TSX disabled	Processor is only vulnerable when TSX is enabled while this system was booted with TSX disabled.
Unknown: Dependent on	
hypervisor status	Running on virtual guest processor that is affected but with no way to know if host processor is mitigated or vulnerable.

### 5.6.8 SRBDS Default mitigation

This new microcode serializes processor access during execution of RDRAND, RDSEED ensures that the shared buffer is overwritten before it is released for reuse. Use the “srbds=off” kernel command line to disable the mitigation for RDRAND and RDSEED.

Here is a set of documents aimed at users who are trying to track down problems and bugs in particular.

## REPORTING BUGS

### 6.1 Background

The upstream Linux kernel maintainers only fix bugs for specific kernel versions. Those versions include the current “release candidate” (or -rc) kernel, any “stable” kernel versions, and any “long term” kernels.

Please see <https://www.kernel.org/> for a list of supported kernels. Any kernel marked with [EOL] is “end of life” and will not have any fixes backported to it.

If you’ve found a bug on a kernel version that isn’t listed on kernel.org, contact your Linux distribution or embedded vendor for support. Alternatively, you can attempt to run one of the supported stable or -rc kernels, and see if you can reproduce the bug on that. It’s preferable to reproduce the bug on the latest -rc kernel.

### 6.2 How to report Linux kernel bugs

#### 6.2.1 Identify the problematic subsystem

Identifying which part of the Linux kernel might be causing your issue increases your chances of getting your bug fixed. Simply posting to the generic linux-kernel mailing list (LKML) may cause your bug report to be lost in the noise of a mailing list that gets 1000+ emails a day.

Instead, try to figure out which kernel subsystem is causing the issue, and email that subsystem’s maintainer and mailing list. If the subsystem maintainer doesn’t answer, then expand your scope to mailing lists like LKML.

#### 6.2.2 Identify who to notify

Once you know the subsystem that is causing the issue, you should send a bug report. Some maintainers prefer bugs to be reported via bugzilla (<https://bugzilla.kernel.org>), while others prefer that bugs be reported via the subsystem mailing list.

To find out where to send an emailed bug report, find your subsystem or device driver in the MAINTAINERS file. Search in the file for relevant entries, and send your bug report to the person(s) listed in the “M:” lines, making sure to Cc the

mailing list(s) in the “L:” lines. When the maintainer replies to you, make sure to ‘Reply-all’ in order to keep the public mailing list(s) in the email thread.

If you know which driver is causing issues, you can pass one of the driver files to the `get_maintainer.pl` script:

```
perl scripts/get_maintainer.pl -f <filename>
```

If it is a security bug, please copy the Security Contact listed in the MAINTAINERS file. They can help coordinate bugfix and disclosure. See `Documentation/admin-guide/security-bugs.rst` for more information.

If you can't figure out which subsystem caused the issue, you should file a bug in kernel.org bugzilla and send email to [linux-kernel@vger.kernel.org](mailto:linux-kernel@vger.kernel.org), referencing the bugzilla URL. (For more information on the linux-kernel mailing list see <http://vger.kernel.org/lkml/>).

### 6.2.3 Tips for reporting bugs

If you haven't reported a bug before, please read:

<https://www.chiark.greenend.org.uk/~sgtatham/bugs.html>

<http://www.catb.org/esr/faqs/smart-questions.html>

It's REALLY important to report bugs that seem unrelated as separate email threads or separate bugzilla entries. If you report several unrelated bugs at once, it's difficult for maintainers to tease apart the relevant data.

### 6.2.4 Gather information

The most important information in a bug report is how to reproduce the bug. This includes system information, and (most importantly) step-by-step instructions for how a user can trigger the bug.

If the failure includes an “OOPS:”, take a picture of the screen, capture a net-console trace, or type the message from your screen into the bug report. Please read “`Documentation/admin-guide/bug-hunting.rst`” before posting your bug report. This explains what you should do with the “Oops” information to make it useful to the recipient.

This is a suggested format for a bug report sent via email or bugzilla. Having a standardized bug report form makes it easier for you not to overlook things, and easier for the developers to find the pieces of information they're really interested in. If some information is not relevant to your bug, feel free to exclude it.

First run the `ver_linux` script included as `scripts/ver_linux`, which reports the version of some important subsystems. Run this script with the command `awk -f scripts/ver_linux`.

Use that information to fill in all fields of the bug report form, and post it to the mailing list with a subject of “PROBLEM: <one line summary from [1.]>” for easy identification by the developers:

```
[1.] One line summary of the problem:
[2.] Full description of the problem/report:
[3.] Keywords (i.e., modules, networking, kernel):
[4.] Kernel information
[4.1.] Kernel version (from /proc/version):
[4.2.] Kernel .config file:
[5.] Most recent kernel version which did not have the bug:
[6.] Output of Oops.. message (if applicable) with symbolic information
    resolved (see Documentation/admin-guide/bug-hunting.rst)
[7.] A small shell script or example program which triggers the
    problem (if possible)
[8.] Environment
[8.1.] Software (add the output of the ver_linux script here)
[8.2.] Processor information (from /proc/cpuinfo):
[8.3.] Module information (from /proc/modules):
[8.4.] Loaded driver and hardware information (/proc/ioproports, /proc/iomem)
[8.5.] PCI information ('lspci -vvv' as root)
[8.6.] SCSI information (from /proc/scsi/scsi)
[8.7.] Other information that might be relevant to the problem
    (please look in /proc and include all information that you
    think to be relevant):
[X.] Other notes, patches, fixes, workarounds:
```

## 6.3 Follow up

### 6.3.1 Expectations for bug reporters

Linux kernel maintainers expect bug reporters to be able to follow up on bug reports. That may include running new tests, applying patches, recompiling your kernel, and/or re-triggering your bug. The most frustrating thing for maintainers is for someone to report a bug, and then never follow up on a request to try out a fix.

That said, it's still useful for a kernel maintainer to know a bug exists on a supported kernel, even if you can't follow up with retests. Follow up reports, such as replying to the email thread with "I tried the latest kernel and I can't reproduce my bug anymore" are also helpful, because maintainers have to assume silence means things are still broken.

### 6.3.2 Expectations for kernel maintainers

Linux kernel maintainers are busy, overworked human beings. Some times they may not be able to address your bug in a day, a week, or two weeks. If they don't answer your email, they may be on vacation, or at a Linux conference. Check the conference schedule at <https://LWN.net> for more info:

<https://lwn.net/Calendar/>

In general, kernel maintainers take 1 to 5 business days to respond to bugs. The majority of kernel maintainers are employed to work on the kernel, and they may not work on the weekends. Maintainers are scattered around the world, and they may not work in your time zone. Unless you have a high priority bug, please wait

at least a week after the first bug report before sending the maintainer a reminder email.

The exceptions to this rule are regressions, kernel crashes, security holes, or userspace breakage caused by new kernel behavior. Those bugs should be addressed by the maintainers ASAP. If you suspect a maintainer is not responding to these types of bugs in a timely manner (especially during a merge window), escalate the bug to LKML and Linus Torvalds.

Thank you!

[Some of this is taken from Frohwalt Egerer' s original linux-kernel FAQ]

## **SECURITY BUGS**

Linux kernel developers take security very seriously. As such, we'd like to know when a security bug is found so that it can be fixed and disclosed as quickly as possible. Please report security bugs to the Linux kernel security team.

### **7.1 Contact**

The Linux kernel security team can be contacted by email at [<security@kernel.org>](mailto:security@kernel.org). This is a private list of security officers who will help verify the bug report and develop and release a fix. If you already have a fix, please include it with your report, as that can speed up the process considerably. It is possible that the security team will bring in extra help from area maintainers to understand and fix the security vulnerability.

As it is with any bug, the more information provided the easier it will be to diagnose and fix. Please review the procedure outlined in `admin-guide/reporting-bugs.rst` if you are unclear about what information is helpful. Any exploit code is very helpful and will not be released without consent from the reporter unless it has already been made public.

### **7.2 Disclosure and embargoed information**

The security list is not a disclosure channel. For that, see Coordination below.

Once a robust fix has been developed, the release process starts. Fixes for publicly known bugs are released immediately.

Although our preference is to release fixes for publicly undisclosed bugs as soon as they become available, this may be postponed at the request of the reporter or an affected party for up to 7 calendar days from the start of the release process, with an exceptional extension to 14 calendar days if it is agreed that the criticality of the bug requires more time. The only valid reason for deferring the publication of a fix is to accommodate the logistics of QA and large scale rollouts which require release coordination.

While embargoed information may be shared with trusted individuals in order to develop a fix, such information will not be published alongside the fix or on any other disclosure channel without the permission of the reporter. This includes but is not limited to the original bug report and followup discussions (if any), exploits, CVE information or the identity of the reporter.

In other words our only interest is in getting bugs fixed. All other information submitted to the security list and any followup discussions of the report are treated confidentially even after the embargo has been lifted, in perpetuity.

### 7.3 Coordination

Fixes for sensitive bugs, such as those that might lead to privilege escalations, may need to be coordinated with the private <[linux-distros@vs.openwall.org](mailto:linux-distros@vs.openwall.org)> mailing list so that distribution vendors are well prepared to issue a fixed kernel upon public disclosure of the upstream fix. Distros will need some time to test the proposed patch and will generally request at least a few days of embargo, and vendor update publication prefers to happen Tuesday through Thursday. When appropriate, the security team can assist with this coordination, or the reporter can include linux-distros from the start. In this case, remember to prefix the email Subject line with “[vs]” as described in the linux-distros wiki: <<http://oss-security.openwall.org/wiki/mailling-lists/distros#how-to-use-the-lists>>

### 7.4 CVE assignment

The security team does not normally assign CVEs, nor do we require them for reports or fixes, as this can needlessly complicate the process and may delay the bug handling. If a reporter wishes to have a CVE identifier assigned ahead of public disclosure, they will need to contact the private linux-distros list, described above. When such a CVE identifier is known before a patch is provided, it is desirable to mention it in the commit message if the reporter agrees.

### 7.5 Non-disclosure agreements

The Linux kernel security team is not a formal body and therefore unable to enter any non-disclosure agreements.

## BUG HUNTING

Kernel bug reports often come with a stack dump like the one below:

```
-----[ cut here ]-----
WARNING: CPU: 1 PID: 28102 at kernel/module.c:1108 module_put+0x57/0x70
Modules linked in: dvb_usb_gp8psk(-) dvb_usb dvb_core nvidia_drm(P0)
↳nvidia_modeset(P0) snd_hda_codec_hdmi snd_hda_intel snd_hda_codec snd_
↳hwdep snd_hda_core snd_pcm snd_timer snd soundcore nvidia(P0) [last
↳unloaded: rc_core]
CPU: 1 PID: 28102 Comm: rmmmod Tainted: P          WC 0 4.8.4-build.1 #1
Hardware name: MSI MS-7309/MS-7309, BIOS V1.12 02/23/2009
00000000 c12ba080 00000000 00000000 c103ed6a c1616014 00000001 00006dc6
c1615862 00000454 c109e8a7 c109e8a7 00000009 ffffffff 00000000 f13f6a10
f5f5a600 c103ee33 00000009 00000000 00000000 c109e8a7 f80ca4d0 c109f617
Call Trace:
[<c12ba080>] ? dump_stack+0x44/0x64
[<c103ed6a>] ? __warn+0xfa/0x120
[<c109e8a7>] ? module_put+0x57/0x70
[<c109e8a7>] ? module_put+0x57/0x70
[<c103ee33>] ? warn_slowpath_null+0x23/0x30
[<c109e8a7>] ? module_put+0x57/0x70
[<f80ca4d0>] ? gp8psk_fe_set_frontend+0x460/0x460 [dvb_usb_gp8psk]
[<c109f617>] ? symbol_put_addr+0x27/0x50
[<f80bc9ca>] ? dvb_usb_adapter_frontend_exit+0x3a/0x70 [dvb_usb]
[<f80bb3bf>] ? dvb_usb_exit+0x2f/0xd0 [dvb_usb]
[<c13d03bc>] ? usb_disable_endpoint+0x7c/0xb0
[<f80bb48a>] ? dvb_usb_device_exit+0x2a/0x50 [dvb_usb]
[<c13d2882>] ? usb_unbind_interface+0x62/0x250
[<c136b514>] ? __pm_runtime_idle+0x44/0x70
[<c13620d8>] ? __device_release_driver+0x78/0x120
[<c1362907>] ? driver_detach+0x87/0x90
[<c1361c48>] ? bus_remove_driver+0x38/0x90
[<c13d1c18>] ? usb_deregister+0x58/0xb0
[<c109fbb0>] ? Sys_delete_module+0x130/0x1f0
[<c1055654>] ? task_work_run+0x64/0x80
[<c1000fa5>] ? exit_to_usermode_loop+0x85/0x90
[<c10013f0>] ? do_fast_syscall_32+0x80/0x130
[<c1549f43>] ? sysenter_past_esp+0x40/0x6a
---[ end trace 6ebc60ef3981792f ]---
```

Such stack traces provide enough information to identify the line inside the Kernel's source code where the bug happened. Depending on the severity of the issue, it may also contain the word **Oops**, as on this one:

```
BUG: unable to handle kernel NULL pointer dereference at (null)
IP: [
```

Despite being an **Oops** or some other sort of stack trace, the offended line is usually required to identify and handle the bug. Along this chapter, we’ ll refer to “Oops” for all kinds of stack traces that need to be analyzed.

If the kernel is compiled with `CONFIG_DEBUG_INFO`, you can enhance the quality of the stack trace by using `file:scripts/decode_stacktrace.sh`.

## 8.1 Modules linked in

Modules that are tainted or are being loaded or unloaded are marked with “(…)” , where the taint flags are described in `file:Documentation/admin-guide/tainted-kernels.rst`, “being loaded” is annotated with “+” , and “being unloaded” is annotated with “-” .

## 8.2 Where is the Oops message is located?

Normally the Oops text is read from the kernel buffers by `klogd` and handed to `syslogd` which writes it to a `syslog` file, typically `/var/log/messages` (depends on `/etc/syslog.conf`). On systems with `systemd`, it may also be stored by the `journald` daemon, and accessed by running `journalctl` command.

Sometimes `klogd` dies, in which case you can run `dmesg > file` to read the data from the kernel buffers and save it. Or you can `cat /proc/kmsg > file`, however you have to break in to stop the transfer, since `kmsg` is a “never ending file” .

If the machine has crashed so badly that you cannot enter commands or the disk is not available then you have three options:

- (1) Hand copy the text from the screen and type it in after the machine has restarted. Messy but it is the only option if you have not planned for a crash. Alternatively, you can take a picture of the screen with a digital camera - not nice, but better than nothing. If the messages scroll off the top of the console, you may find that booting with a higher resolution (e.g., `vga=791`) will allow you to read more of the text. (Caveat: This needs `vesafb`, so won’ t help for ‘early’ oopses.)
- (2) Boot with a serial console (see `Documentation/admin-guide/serial-console.rst`), run a null modem to a second machine and capture the output there using your favourite communication program. `Minicom` works well.
- (3) Use `Kdump` (see `Documentation/admin-guide/kdump/kdump.rst`), extract the kernel ring buffer from old memory with using `dmesg gdbmacro` in `Documentation/admin-guide/kdump/gdbmacros.txt`.

## 8.3 Finding the bug' s location

Reporting a bug works best if you point the location of the bug at the Kernel source file. There are two methods for doing that. Usually, using `gdb` is easier, but the Kernel should be pre-compiled with debug info.

### 8.3.1 `gdb`

The GNU debugger (`gdb`) is the best way to figure out the exact file and line number of the OOPS from the `vmlinux` file.

The usage of `gdb` works best on a kernel compiled with `CONFIG_DEBUG_INFO`. This can be set by running:

```
$ ./scripts/config -d COMPILE_TEST -e DEBUG_KERNEL -e DEBUG_INFO
```

On a kernel compiled with `CONFIG_DEBUG_INFO`, you can simply copy the EIP value from the OOPS:

```
EIP: 0060:[<c021e50e>] Not tainted VLI
```

And use GDB to translate that to human-readable form:

```
$ gdb vmlinux
(gdb) l *0xc021e50e
```

If you don' t have `CONFIG_DEBUG_INFO` enabled, you use the function offset from the OOPS:

```
EIP is at vt_ioctl+0xda8/0x1482
```

And recompile the kernel with `CONFIG_DEBUG_INFO` enabled:

```
$ ./scripts/config -d COMPILE_TEST -e DEBUG_KERNEL -e DEBUG_INFO
$ make vmlinux
$ gdb vmlinux
(gdb) l *vt_ioctl+0xda8
0x1888 is in vt_ioctl (drivers/tty/vt/vt_ioctl.c:293).
288  {
289      struct vc_data *vc = NULL;
290      int ret = 0;
291
292      console_lock();
293      if (VT_BUSY(vc_num))
294          ret = -EBUSY;
295      else if (vc_num)
296          vc = vc_deallocate(vc_num);
297      console_unlock();
```

or, if you want to be more verbose:

```
(gdb) p vt_ioctl
$1 = {int (struct tty_struct *, unsigned int, unsigned long)} 0xae0 <vt_
->ioctl>
(gdb) l *0xae0+0xda8
```

You could, instead, use the object file:

```
$ make drivers/tty/  
$ gdb drivers/tty/vt/vt_ioctl.o  
(gdb) l *vt_ioctl+0xda8
```

If you have a call trace, such as:

```
Call Trace:  
[<ffffffff8802c8e9>] :jbd:log_wait_commit+0xa3/0xf5  
[<ffffffff810482d9>] autoremove_wake_function+0x0/0x2e  
[<ffffffff8802770b>] :jbd:journal_stop+0x1be/0x1ee  
...
```

this shows the problem likely is in the `:jbd:` module. You can load that module in `gdb` and list the relevant code:

```
$ gdb fs/jbd/jbd.ko  
(gdb) l *log_wait_commit+0xa3
```

---

**Note:** You can also do the same for any function call at the stack trace, like this one:

```
[<f80bc9ca>] ? dvb_usb_adapter_frontend_exit+0x3a/0x70 [dvb_usb]
```

The position where the above call happened can be seen with:

```
$ gdb drivers/media/usb/dvb-usb/dvb-usb.o  
(gdb) l *dvb_usb_adapter_frontend_exit+0x3a
```

---

### 8.3.2 objdump

To debug a kernel, use `objdump` and look for the hex offset from the crash output to find the valid line of code/ assembler. Without debug symbols, you will see the assembler code for the routine shown, but if your kernel has debug symbols the C code will also be available. (Debug symbols can be enabled in the kernel hacking menu of the menu configuration.) For example:

```
$ objdump -r -S -l --disassemble net/dccp/ipv4.o
```

---

**Note:** You need to be at the top level of the kernel tree for this to pick up your C files.

If you don't have access to the source code you can still debug some crash dumps using the following method (example crash dump output as shown by Dave Miller):

```
EIP is at +0x14/0x4c0  
...  
Code: 44 24 04 e8 6f 05 00 00 e9 e8 fe ff ff 8d 76 00 8d bc 27 00 00
```

(continues on next page)

(continued from previous page)

```
00 00 55 57 56 53 81 ec bc 00 00 00 8b ac 24 d0 00 00 00 8b 5d 08
<8b> 83 3c 01 00 00 89 44 24 14 8b 45 28 85 c0 89 44 24 18 0f 85
```

Put the bytes into a "foo.s" file like this:

```
.text
.globl foo
foo:
.byte .... /* bytes from Code: part of OOPS dump */
```

Compile it with "gcc -c -o foo.o foo.s" then look at the output of "objdump --disassemble foo.o".

Output:

```
ip_queue_xmit:
push    %ebp
push    %edi
push    %esi
push    %ebx
sub     $0xbc, %esp
mov     0xd0(%esp), %ebp      ! %ebp = arg0 (skb)
mov     0x8(%ebp), %ebx      ! %ebx = skb->sk
mov     0x13c(%ebx), %eax    ! %eax = inet_sk(skb)->opt
```

file:scripts/decodecode can be used to automate most of this, depending on what CPU architecture is being debugged.

## 8.4 Reporting the bug

Once you find where the bug happened, by inspecting its location, you could either try to fix it yourself or report it upstream.

In order to report it upstream, you should identify the mailing list used for the development of the affected code. This can be done by using the `get_maintainer.pl` script.

For example, if you find a bug at the `gspca`'s `sonixj.c` file, you can get its maintainers with:

```
$ ./scripts/get_maintainer.pl -f drivers/media/usb/gspca/sonixj.c
Hans Verkuil <hverkuil@xs4all.nl> (odd fixer:GSPCA USB WEBCAM DRIVER,
↳commit_signer:1/1=100%)
Mauro Carvalho Chehab <mchehab@kernel.org> (maintainer:MEDIA INPUT,
↳INFRASTRUCTURE (V4L/DVB),commit_signer:1/1=100%)
Tejun Heo <tj@kernel.org> (commit_signer:1/1=100%)
Bhaktipriya Shridhar <bhaktipriya96@gmail.com> (commit_signer:1/1=100%,
↳authored:1/1=100%,added_lines:4/4=100%,removed_lines:9/9=100%)
linux-media@vger.kernel.org (open list:GSPCA USB WEBCAM DRIVER)
linux-kernel@vger.kernel.org (open list)
```

Please notice that it will point to:

- The last developers that touched the source code (if this is done inside a git

tree). On the above example, Tejun and Bhaktipriya (in this specific case, none really involved on the development of this file);

- The driver maintainer (Hans Verkuil);
- The subsystem maintainer (Mauro Carvalho Chehab);
- The driver and/or subsystem mailing list ([linux-media@vger.kernel.org](mailto:linux-media@vger.kernel.org));
- the Linux Kernel mailing list ([linux-kernel@vger.kernel.org](mailto:linux-kernel@vger.kernel.org)).

Usually, the fastest way to have your bug fixed is to report it to mailing list used for the development of the code (linux-media ML) copying the driver maintainer (Hans).

If you are totally stumped as to whom to send the report, and `get_maintainer.pl` didn't provide you anything useful, send it to [linux-kernel@vger.kernel.org](mailto:linux-kernel@vger.kernel.org).

Thanks for your help in making Linux as stable as humanly possible.

## 8.5 Fixing the bug

If you know programming, you could help us by not only reporting the bug, but also providing us with a solution. After all, open source is about sharing what you do and don't you want to be recognised for your genius?

If you decide to take this way, once you have worked out a fix please submit it upstream.

Please do read Documentation/process/submitting-patches.rst though to help your code get accepted.

---

## 8.6 Notes on Oops tracing with klogd

In order to help Linus and the other kernel developers there has been substantial support incorporated into klogd for processing protection faults. In order to have full support for address resolution at least version 1.3-pl3 of the sysklogd package should be used.

When a protection fault occurs the klogd daemon automatically translates important addresses in the kernel log messages to their symbolic equivalents. This translated kernel message is then forwarded through whatever reporting mechanism klogd is using. The protection fault message can be simply cut out of the message files and forwarded to the kernel developers.

Two types of address resolution are performed by klogd. The first is static translation and the second is dynamic translation. Static translation uses the System.map file. In order to do static translation the klogd daemon must be able to find a system map file at daemon initialization time. See the klogd man page for information on how klogd searches for map files.

Dynamic address translation is important when kernel loadable modules are being used. Since memory for kernel modules is allocated from the kernel's dynamic

memory pools there are no fixed locations for either the start of the module or for functions and symbols in the module.

The kernel supports system calls which allow a program to determine which modules are loaded and their location in memory. Using these system calls the klogd daemon builds a symbol table which can be used to debug a protection fault which occurs in a loadable kernel module.

At the very minimum klogd will provide the name of the module which generated the protection fault. There may be additional symbolic information available if the developer of the loadable module chose to export symbol information from the module.

Since the kernel module environment can be dynamic there must be a mechanism for notifying the klogd daemon when a change in module environment occurs. There are command line options available which allow klogd to signal the currently executing daemon that symbol information should be refreshed. See the klogd manual page for more information.

A patch is included with the sysklogd distribution which modifies the modules-2.0.0 package to automatically signal klogd whenever a module is loaded or unloaded. Applying this patch provides essentially seamless support for debugging protection faults which occur with kernel loadable modules.

The following is an example of a protection fault in a loadable module processed by klogd:

```
Aug 29 09:51:01 blizard kernel: Unable to handle kernel paging request at
↳virtual address f15e97cc
Aug 29 09:51:01 blizard kernel: current->tss.cr3 = 0062d000, %cr3 =
↳0062d000
Aug 29 09:51:01 blizard kernel: *pde = 00000000
Aug 29 09:51:01 blizard kernel: Oops: 0002
Aug 29 09:51:01 blizard kernel: CPU: 0
Aug 29 09:51:01 blizard kernel: EIP: 0010:[oops:_oops+16/3868]
Aug 29 09:51:01 blizard kernel: EFLAGS: 00010212
Aug 29 09:51:01 blizard kernel: eax: 315e97cc ebx: 003a6f80 ecx:
↳001be77b edx: 00237c0c
Aug 29 09:51:01 blizard kernel: esi: 00000000 edi: bffffdb3 ebp:
↳00589f90 esp: 00589f8c
Aug 29 09:51:01 blizard kernel: ds: 0018 es: 0018 fs: 002b gs: 002b
↳
↳ss: 0018
Aug 29 09:51:01 blizard kernel: Process oops_test (pid: 3374, process nr:
↳21, stackpage=00589000)
Aug 29 09:51:01 blizard kernel: Stack: 315e97cc 00589f98 0100b0b4 bffffed4
↳0012e38e 00240c64 003a6f80 00000001
Aug 29 09:51:01 blizard kernel: 00000000 00237810 bffffff0 0010a7fa
↳00000003 00000001 00000000 bffffff0
Aug 29 09:51:01 blizard kernel: bffffdb3 bffffed4 fffffffda 0000002b
↳0007002b 0000002b 0000002b 00000036
Aug 29 09:51:01 blizard kernel: Call Trace: [oops:_oops_ioctl+48/80] [_sys_
↳ioctl+254/272] [_system_call+82/128]
Aug 29 09:51:01 blizard kernel: Code: c7 00 05 00 00 00 eb 08 90 90 90 90
↳90 90 90 90 89 ec 5d c3
```

Dr. G.W. Wettstein Roger Maris Cancer Center 820 4th St. N. Fargo, ND 58122 Phone: 701-234-7556	Oncology Research Div. Computing Facility INTERNET: greg@wind.rmcc.com
---	---

## BISECTING A BUG

Last updated: 28 October 2016

### 9.1 Introduction

Always try the latest kernel from kernel.org and build from source. If you are not confident in doing that please report the bug to your distribution vendor instead of to a kernel developer.

Finding bugs is not always easy. Have a go though. If you can't find it don't give up. Report as much as you have found to the relevant maintainer. See MAINTAINERS for who that is for the subsystem you have worked on.

Before you submit a bug report read Documentation/admin-guide/reporting-bugs.rst.

### 9.2 Devices not appearing

Often this is caused by udev/systemd. Check that first before blaming it on the kernel.

### 9.3 Finding patch that caused a bug

Using the provided tools with git makes finding bugs easy provided the bug is reproducible.

Steps to do it:

- build the Kernel from its git source
- start bisect with<sup>1</sup>:

```
$ git bisect start
```

- mark the broken changeset with:

---

<sup>1</sup> You can, optionally, provide both good and bad arguments at git start with `git bisect start [BAD] [GOOD]`

```
$ git bisect bad [commit]
```

- mark a changeset where the code is known to work with:

```
$ git bisect good [commit]
```

- rebuild the Kernel and test
- interact with git bisect by using either:

```
$ git bisect good
```

or:

```
$ git bisect bad
```

depending if the bug happened on the changeset you' re testing

- After some interactions, git bisect will give you the changeset that likely caused the bug.
- For example, if you know that the current version is bad, and version 4.8 is good, you could do:

```
$ git bisect start
$ git bisect bad           # Current version is bad
$ git bisect good v4.8
```

For further references, please read:

- [The man page for git-bisect](#)
- [Fighting regressions with git bisect](#)
- [Fully automated bisecting with “git bisect run”](#)
- [Using Git bisect to figure out when brokenness was introduced](#)

## TAINTED KERNELS

The kernel will mark itself as ‘tainted’ when something occurs that might be relevant later when investigating problems. Don’ t worry too much about this, most of the time it’ s not a problem to run a tainted kernel; the information is mainly of interest once someone wants to investigate some problem, as its real cause might be the event that got the kernel tainted. That’ s why bug reports from tainted kernels will often be ignored by developers, hence try to reproduce problems with an untainted kernel.

Note the kernel will remain tainted even after you undo what caused the taint (i.e. unload a proprietary kernel module), to indicate the kernel remains not trustworthy. That’ s also why the kernel will print the tainted state when it notices an internal problem (a ‘kernel bug’ ), a recoverable error ( ‘kernel oops’ ) or a non-recoverable error ( ‘kernel panic’ ) and writes debug information about this to the logs dmesg outputs. It’ s also possible to check the tainted state at runtime through a file in /proc/.

### 10.1 Tainted flag in bugs, oops or panics messages

You find the tainted state near the top in a line starting with ‘CPU:’ ; if or why the kernel was tainted is shown after the Process ID ( ‘PID:’ ) and a shortened name of the command ( ‘Comm:’ ) that triggered the event:

```
BUG: unable to handle kernel NULL pointer dereference at 0000000000000000
Oops: 0002 [#1] SMP PTI
CPU: 0 PID: 4424 Comm: insmod Tainted: P          W 0          4.20.0-0.rc6.
→fc30 #1
Hardware name: Red Hat KVM, BIOS 0.5.1 01/01/2011
RIP: 0010:my_oops_init+0x13/0x1000 [kpanic]
[...]
```

You’ ll find a ‘Not tainted:’ there if the kernel was not tainted at the time of the event; if it was, then it will print ‘Tainted:’ and characters either letters or blanks. In above example it looks like this:

```
Tainted: P          W 0
```

The meaning of those characters is explained in the table below. In tis case the kernel got tainted earlier because a proprietary Module (P) was loaded, a warning occurred (W), and an externally-built module was loaded (0). To decode other letters use the table below.

## 10.2 Decoding tainted state at runtime

At runtime, you can query the tainted state by reading `cat /proc/sys/kernel/tainted`. If that returns `0`, the kernel is not tainted; any other number indicates the reasons why it is. The easiest way to decode that number is the script `tools/debugging/kernel-chktaint`, which your distribution might ship as part of a package called `linux-tools` or `kernel-tools`; if it doesn't you can download the script from [git.kernel.org](https://git.kernel.org) and execute it with `sh kernel-chktaint`, which would print something like this on the machine that had the statements in the logs that were quoted earlier:

```
Kernel is Tainted for following reasons:
* Proprietary module was loaded (#0)
* Kernel issued warning (#9)
* Externally-built ('out-of-tree') module was loaded (#12)
See Documentation/admin-guide/tainted-kernels.rst in the the Linux kernel
↳or
https://www.kernel.org/doc/html/latest/admin-guide/tainted-kernels.html
↳for
a more details explanation of the various taint flags.
Raw taint value as int/string: 4609/'P          W 0          '
```

You can try to decode the number yourself. That's easy if there was only one reason that got your kernel tainted, as in this case you can find the number with the table below. If there were multiple reasons you need to decode the number, as it is a bitfield, where each bit indicates the absence or presence of a particular type of taint. It's best to leave that to the aforementioned script, but if you need something quick you can use this shell command to check which bits are set:

```
$ for i in $(seq 18); do echo $((($i-1)) $($ (cat /proc/sys/kernel/tainted)>
↳>($i-1)&1));done
```

## 10.3 Table for decoding tainted state

Bit	Log	Number	Reason that got the kernel tainted
0	G/P	1	proprietary module was loaded
1	_/F	2	module was force loaded
2	_/S	4	SMP kernel oops on an officially SMP incapable processor
3	_/R	8	module was force unloaded
4	_/M	16	processor reported a Machine Check Exception (MCE)
5	_/B	32	bad page referenced or some unexpected page flags
6	_/U	64	taint requested by userspace application
7	_/D	128	kernel died recently, i.e. there was an OOPS or BUG
8	_/A	256	ACPI table overridden by user
9	_/W	512	kernel issued warning
10	_/C	1024	staging driver was loaded
11	_/I	2048	workaround for bug in platform firmware applied
12	_/O	4096	externally-built ( "out-of-tree" ) module was loaded
13	_/E	8192	unsigned module was loaded
14	_/L	16384	soft lockup occurred
15	_/K	32768	kernel has been live patched
16	_/X	65536	auxiliary taint, defined for and used by distros
17	_/T	131072	kernel was built with the struct randomization plugin

Note: The character \_ is representing a blank in this table to make reading easier.

## 10.4 More detailed explanation for tainting

- 0) G if all modules loaded have a GPL or compatible license, P if any proprietary module has been loaded. Modules without a `MODULE_LICENSE` or with a `MODULE_LICENSE` that is not recognised by `insmod` as GPL compatible are assumed to be proprietary.
- 1) F if any module was force loaded by `insmod -f`, ' ' if all modules were loaded normally.
- 2) S if the oops occurred on an SMP kernel running on hardware that hasn't been certified as safe to run multiprocessor. Currently this occurs only on various Athlons that are not SMP capable.
- 3) R if a module was force unloaded by `rmmod -f`, ' ' if all modules were unloaded normally.
- 4) M if any processor has reported a Machine Check Exception, ' ' if no Machine Check Exceptions have occurred.
- 5) B If a page-release function has found a bad page reference or some unexpected page flags. This indicates a hardware problem or a kernel bug; there should be other information in the log indicating why this tainting occurred.
- 6) U if a user or user application specifically requested that the Tainted flag be set, ' ' otherwise.

- 7) D if the kernel has died recently, i.e. there was an OOPS or BUG.
- 8) A if an ACPI table has been overridden.
- 9) W if a warning has previously been issued by the kernel. (Though some warnings may set more specific taint flags.)
- 10) C if a staging driver has been loaded.
- 11) I if the kernel is working around a severe bug in the platform firmware (BIOS or similar).
- 12) O if an externally-built ( “out-of-tree” ) module has been loaded.
- 13) E if an unsigned module has been loaded in a kernel supporting module signature.
- 14) L if a soft lockup has previously occurred on the system.
- 15) K if the kernel has been live patched.
- 16) X Auxiliary taint, defined for and used by Linux distributors.
- 17) T Kernel was build with the randstruct plugin, which can intentionally produce extremely unusual kernel structure layouts (even performance pathological ones), which is important to know when debugging. Set at build time.

## RAMOOPS OOPS/PANIC LOGGER

Sergiu Iordache <sergiu@chromium.org>

Updated: 17 November 2011

### 11.1 Introduction

Ramoops is an oops/panic logger that writes its logs to RAM before the system crashes. It works by logging oopses and panics in a circular buffer. Ramoops needs a system with persistent RAM so that the content of that area can survive after a restart.

### 11.2 Ramoops concepts

Ramoops uses a predefined memory area to store the dump. The start and size and type of the memory area are set using three variables:

- `mem_address` for the start
- `mem_size` for the size. The memory size will be rounded down to a power of two.
- `mem_type` to specify if the memory type (default is `pgprot_writecombine`).

Typically the default value of `mem_type=0` should be used as that sets the `pstore` mapping to `pgprot_writecombine`. Setting `mem_type=1` attempts to use `pgprot_noncached`, which only works on some platforms. This is because `pstore` depends on atomic operations. At least on ARM, `pgprot_noncached` causes the memory to be mapped strongly ordered, and atomic operations on strongly ordered memory are implementation defined, and won't work on many ARMs such as omap.

The memory area is divided into `record_size` chunks (also rounded down to power of two) and each `kmsg` dump writes a `record_size` chunk of information.

Limiting which kinds of `kmsg` dumps are stored can be controlled via the `max_reason` value, as defined in `include/linux/kmsg_dump.h`'s enum `kmsg_dump_reason`. For example, to store both Oopses and Panics, `max_reason` should be set to 2 (`KMSG_DUMP_OOPS`), to store only Panics `max_reason` should be set to 1 (`KMSG_DUMP_PANIC`). Setting this to 0 (`KMSG_DUMP_UNDEF`),

means the reason filtering will be controlled by the `printk.always_kmsg_dump` boot param: if unset, it'll be `KMSG_DUMP_OOPS`, otherwise `KMSG_DUMP_MAX`.

The module uses a counter to record multiple dumps but the counter gets reset on restart (i.e. new dumps after the restart will overwrite old ones).

Ramoops also supports software ECC protection of persistent memory regions. This might be useful when a hardware reset was used to bring the machine back to life (i.e. a watchdog triggered). In such cases, RAM may be somewhat corrupt, but usually it is restorable.

### 11.3 Setting the parameters

Setting the ramoops parameters can be done in several different manners:

A. Use the module parameters (which have the names of the variables described as before). For quick debugging, you can also reserve parts of memory during boot and then use the reserved memory for ramoops. For example, assuming a machine with > 128 MB of memory, the following kernel command line will tell the kernel to use only the first 128 MB of memory, and place ECC-protected ramoops region at 128 MB boundary:

```
mem=128M ramoops.mem_address=0x8000000 ramoops.ecc=1
```

B. Use Device Tree bindings, as described in `Documentation/devicetree/bindings/reserved-memory/ramoops.txt`. For example:

```
reserved-memory {
    #address-cells = <2>;
    #size-cells = <2>;
    ranges;

    ramoops@8f000000 {
        compatible = "ramoops";
        reg = <0 0x8f000000 0 0x100000>;
        record-size = <0x4000>;
        console-size = <0x4000>;
    };
};
```

C. Use a platform device and set the platform data. The parameters can then be set through that platform data. An example of doing that is:

```
#include <linux/pstore_ram.h>
[...]
```

```
static struct ramoops_platform_data ramoops_data = {
    .mem_size           = <...>,
    .mem_address       = <...>,
    .mem_type          = <...>,
    .record_size       = <...>,
    .max_reason        = <...>,
    .ecc               = <...>,
};
```

(continues on next page)

(continued from previous page)

```

static struct platform_device ramoops_dev = {
    .name = "ramoops",
    .dev = {
        .platform_data = &ramoops_data,
    },
};

[... inside a function ...]
int ret;

ret = platform_device_register(&ramoops_dev);
if (ret) {
    printk(KERN_ERR "unable to register platform device\n");
    return ret;
}

```

You can specify either RAM memory or peripheral devices' memory. However, when specifying RAM, be sure to reserve the memory by issuing `memblock_reserve()` very early in the architecture code, e.g.:

```

#include <linux/memblock.h>

memblock_reserve(ramoops_data.mem_address, ramoops_data.mem_size);

```

## 11.4 Dump format

The data dump begins with a header, currently defined as `====` followed by a timestamp and a new line. The dump then continues with the actual data.

## 11.5 Reading the data

The dump data can be read from the `pstore` filesystem. The format for these files is `dmesg - ramoops -N`, where `N` is the record number in memory. To delete a stored record from RAM, simply unlink the respective `pstore` file.

## 11.6 Persistent function tracing

Persistent function tracing might be useful for debugging software or hardware related hangs. The functions call chain log is stored in a `ftrace-ramoops` file. Here is an example of usage:

```

# mount -t debugfs debugfs /sys/kernel/debug/
# echo 1 > /sys/kernel/debug/pstore/record_ftrace
# reboot -f
[...]
# mount -t pstore pstore /mnt/
# tail /mnt/ftrace-ramoops

```

(continues on next page)

(continued from previous page)

```
0 ffffffff8101ea64 ffffffff8101bcda native_apic_mem_read <- disconnect_
↳bsp_APIC+0x6a/0xc0
0 ffffffff8101ea44 ffffffff8101bcf6 native_apic_mem_write <- disconnect_
↳bsp_APIC+0x86/0xc0
0 ffffffff81020084 ffffffff8101a4b5 hpet_disable <- native_machine_
↳shutdown+0x75/0x90
0 ffffffff81005f94 ffffffff8101a4bb iommu_shutdown_noop <- native_
↳machine_shutdown+0x7b/0x90
0 ffffffff8101a6a1 ffffffff8101a437 native_machine_emergency_restart <-
↳native_machine_restart+0x37/0x40
0 ffffffff811f9876 ffffffff8101a73a acpi_reboot <- native_machine_
↳emergency_restart+0xaa/0x1e0
0 ffffffff8101a514 ffffffff8101a772 mach_reboot_fixups <- native_machine_
↳emergency_restart+0xe2/0x1e0
0 ffffffff811d9c54 ffffffff8101a7a0 __const_udelay <- native_machine_
↳emergency_restart+0x110/0x1e0
0 ffffffff811d9c34 ffffffff811d9c80 __delay <- __const_udelay+0x30/0x40
0 ffffffff811d9d14 ffffffff811d9c3f delay_tsc <- __delay+0xf/0x20
```

## DYNAMIC DEBUG

### 12.1 Introduction

This document describes how to use the dynamic debug (dyndbg) feature.

Dynamic debug is designed to allow you to dynamically enable/disable kernel code to obtain additional kernel information. Currently, if `CONFIG_DYNAMIC_DEBUG` is set, then all `pr_debug()/dev_dbg()` and `print_hex_dump_debug()/print_hex_dump_bytes()` calls can be dynamically enabled per-callsite.

If you do not want to enable dynamic debug globally (i.e. in some embedded system), you may set `CONFIG_DYNAMIC_DEBUG_CORE` as basic support of dynamic debug and add `ccflags := -DDYNAMIC_DEBUG_MODULE` into the Makefile of any modules which you'd like to dynamically debug later.

If `CONFIG_DYNAMIC_DEBUG` is not set, `print_hex_dump_debug()` is just shortcut for `print_hex_dump(KERN_DEBUG)`.

For `print_hex_dump_debug()/print_hex_dump_bytes()`, format string is its `prefix_str` argument, if it is constant string; or hexdump in case `prefix_str` is built dynamically.

Dynamic debug has even more useful features:

- Simple query language allows turning on and off debugging statements by matching any combination of 0 or 1 of:
  - source filename
  - function name
  - line number (including ranges of line numbers)
  - module name
  - format string
- Provides a debugfs control file: `<debugfs>/dynamic_debug/control` which can be read to display the complete list of known debug statements, to help guide you

## 12.2 Controlling dynamic debug Behaviour

The behaviour of `pr_debug()`/`dev_dbg()` are controlled via writing to a control file in the 'debugfs' filesystem. Thus, you must first mount the debugfs filesystem, in order to make use of this feature. Subsequently, we refer to the control file as: `<debugfs>/dynamic_debug/control`. For example, if you want to enable printing from source file `svcsock.c`, line 1603 you simply do:

```
nullarbor:~ # echo 'file svcsock.c line 1603 +p' >
               <debugfs>/dynamic_debug/control
```

If you make a mistake with the syntax, the write will fail thus:

```
nullarbor:~ # echo 'file svcsock.c wtf 1 +p' >
               <debugfs>/dynamic_debug/control
-bash: echo: write error: Invalid argument
```

Note, for systems without 'debugfs' enabled, the control file can be found in `/proc/dynamic_debug/control`.

## 12.3 Viewing Dynamic Debug Behaviour

You can view the currently configured behaviour of all the debug statements via:

```
nullarbor:~ # cat <debugfs>/dynamic_debug/control
# filename:lineno [module]function flags format
/usr/src/packages/BUILD/sgi-enhancednfs-1.4/default/net/sunrpc/svc_rdma.
↪c:323 [svcxprt_rdma]svc_rdma_cleanup =_ "SVCRDMA Module Removed,
↪deregister RPC RDMA transport\012"
/usr/src/packages/BUILD/sgi-enhancednfs-1.4/default/net/sunrpc/svc_rdma.
↪c:341 [svcxprt_rdma]svc_rdma_init =_ "\011max_inline      : %d\012"
/usr/src/packages/BUILD/sgi-enhancednfs-1.4/default/net/sunrpc/svc_rdma.
↪c:340 [svcxprt_rdma]svc_rdma_init =_ "\011sq_depth       : %d\012"
/usr/src/packages/BUILD/sgi-enhancednfs-1.4/default/net/sunrpc/svc_rdma.
↪c:338 [svcxprt_rdma]svc_rdma_init =_ "\011max_requests   : %d\012"
...
```

You can also apply standard Unix text manipulation filters to this data, e.g.:

```
nullarbor:~ # grep -i rdma <debugfs>/dynamic_debug/control | wc -l
62

nullarbor:~ # grep -i tcp <debugfs>/dynamic_debug/control | wc -l
42
```

The third column shows the currently enabled flags for each debug statement call-site (see below for definitions of the flags). The default value, with no flags enabled, is `=_`. So you can view all the debug statement call-sites with any non-default flags:

```
nullarbor:~ # awk '$3 != "=" <debugfs>/dynamic_debug/control
# filename:lineno [module]function flags format
/usr/src/packages/BUILD/sgi-enhancednfs-1.4/default/net/sunrpc/svcsock.
↪c:1603 [sunrpc]svc_send p "svc_process: st_sendto returned %d\012"
```

## 12.4 Command Language Reference

At the lexical level, a command comprises a sequence of words separated by spaces or tabs. So these are all equivalent:

```
nullarbor:~ # echo -n 'file svcsock.c line 1603 +p' >
                <debugfs>/dynamic_debug/control
nullarbor:~ # echo -n ' file   svcsock.c   line 1603 +p ' >
                <debugfs>/dynamic_debug/control
nullarbor:~ # echo -n 'file svcsock.c line 1603 +p' >
                <debugfs>/dynamic_debug/control
```

Command submissions are bounded by a write() system call. Multiple commands can be written together, separated by ; or \n:

```
~# echo "func pnpacpi_get_resources +p; func pnp_assign_mem +p" \
  > <debugfs>/dynamic_debug/control
```

If your query set is big, you can batch them too:

```
~# cat query-batch-file > <debugfs>/dynamic_debug/control
```

Another way is to use wildcards. The match rule supports \* (matches zero or more characters) and ? (matches exactly one character). For example, you can match all usb drivers:

```
~# echo "file drivers/usb/* +p" > <debugfs>/dynamic_debug/control
```

At the syntactical level, a command comprises a sequence of match specifications, followed by a flags change specification:

```
command ::= match-spec* flags-spec
```

The match-spec's are used to choose a subset of the known pr\_debug() callsites to which to apply the flags-spec. Think of them as a query with implicit ANDs between each pair. Note that an empty list of match-specs will select all debug statement callsites.

A match specification comprises a keyword, which controls the attribute of the callsite to be compared, and a value to compare against. Possible keywords are::

```
match-spec ::= 'func' string |
               'file' string |
               'module' string |
               'format' string |
               'line' line-range

line-range ::= lineno |
             '-'lineno |
             lineno'-' |
             lineno'-'lineno

lineno ::= unsigned-int
```

**Note:** line-range cannot contain space, e.g. “1-30” is valid range but “1 - 30” is not.

---

The meanings of each keyword are:

**func** The given string is compared against the function name of each callsite.  
Example:

```
func svc_tcp_accept
```

**file** The given string is compared against either the full pathname, the src-root relative pathname, or the basename of the source file of each callsite. Examples:

```
file svcsock.c
file kernel/freezer.c
file /usr/src/packages/BUILD/sgi-enhancednfs-1.4/default/net/sunrpc/
↪svcsock.c
```

**module** The given string is compared against the module name of each callsite. The module name is the string as seen in `lsmod`, i.e. without the directory or the `.ko` suffix and with `-` changed to `_`. Examples:

```
module sunrpc
module nfsd
```

**format** The given string is searched for in the dynamic debug format string. Note that the string does not need to match the entire format, only some part. Whitespace and other special characters can be escaped using C octal character escape `\000` notation, e.g. the space character is `\040`. Alternatively, the string can be enclosed in double quote characters (`"`) or single quote characters (`'`). Examples:

```
format svcrdma:          // many of the NFS/RDMA server pr_debugs
format readahead        // some pr_debugs in the readahead cache
format nfsd:\040SETATTR // one way to match a format with whitespace
format "nfsd: SETATTR"  // a neater way to match a format with ↪
↪whitespace
format 'nfsd: SETATTR'  // yet another way to match a format with ↪
↪whitespace
```

**line** The given line number or range of line numbers is compared against the line number of each `pr_debug()` callsite. A single line number matches the callsite line number exactly. A range of line numbers matches any callsite between the first and last line number inclusive. An empty first number means the first line in the file, an empty last line number means the last line number in the file. Examples:

```
line 1603                // exactly line 1603
line 1600-1605          // the six lines from line 1600 to line 1605
line -1605              // the 1605 lines from line 1 to line 1605
line 1600-              // all lines from line 1600 to the end of the file
```

The flags specification comprises a change operation followed by one or more flag characters. The change operation is one of the characters:

-	remove the given flags
+	add the given flags
=	set the flags to the given flags

The flags are:

p	enables the <code>pr_debug()</code> callsite.
f	Include the function name in the printed message
l	Include line number in the printed message
m	Include module name in the printed message
t	Include thread ID in messages not generated from interrupt context
_	No flags are set. (Or'd with others on input)

For `print_hex_dump_debug()` and `print_hex_dump_bytes()`, only `p` flag have meaning, other flags ignored.

For display, the flags are preceded by `=` (mnemonic: what the flags are currently equal to).

Note the regexp `^[ -+]=[flmpt_]+$` matches a flags specification. To clear all flags at once, use `=_` or `-flmpt`.

## 12.5 Debug messages during Boot Process

To activate debug messages for core code and built-in modules during the boot process, even before userspace and `debugfs` exists, use `dyndbg="QUERY"`, `module.dyndbg="QUERY"`, or `ddebug_query="QUERY"` (`ddebug_query` is obsoleted by `dyndbg`, and deprecated). `QUERY` follows the syntax described above, but must not exceed 1023 characters. Your bootloader may impose lower limits.

These `dyndbg` params are processed just after the `ddebug` tables are processed, as part of the `arch_initcall`. Thus you can enable debug messages in all code run after this `arch_initcall` via this boot parameter.

On an x86 system for example ACPI enablement is a `subsys_initcall` and:

<code>dyndbg="file ec.c +p"</code>
------------------------------------

will show early Embedded Controller transactions during ACPI setup if your machine (typically a laptop) has an Embedded Controller. PCI (or other devices) initialization also is a hot candidate for using this boot parameter for debugging purposes.

If `foo` module is not built-in, `foo.dyndbg` will still be processed at boot time, without effect, but will be reprocessed when module is loaded later. `ddebug_query=` and bare `dyndbg=` are only processed at boot.

## 12.6 Debug Messages at Module Initialization Time

When `modprobe foo` is called, `modprobe` scans `/proc/cmdline` for `foo.params`, strips `foo.`, and passes them to the kernel along with params given in `modprobe args` or `/etc/modprobe.d/*.conf` files, in the following order:

1. parameters given via `/etc/modprobe.d/*.conf`:

```
options foo dyndbg=+pt
options foo dyndbg # defaults to +p
```

2. `foo.dyndbg` as given in boot args, `foo.` is stripped and passed:

```
foo.dyndbg=" func bar +p; func buz +mp"
```

3. args to `modprobe`:

```
modprobe foo dyndbg==pmf # override previous settings
```

These `dyndbg` queries are applied in order, with last having final say. This allows boot args to override or modify those from `/etc/modprobe.d` (sensible, since 1 is system wide, 2 is kernel or boot specific), and `modprobe args` to override both.

In the `foo.dyndbg="QUERY"` form, the query must exclude module `foo`. `foo` is extracted from the param-name, and applied to each query in `QUERY`, and only 1 match-spec of each type is allowed.

The `dyndbg` option is a “fake” module parameter, which means:

- modules do not need to define it explicitly
- every module gets it tacitly, whether they use `pr_debug` or not
- it doesn't appear in `/sys/module/$module/parameters/` To see it, `grep` the control file, or inspect `/proc/cmdline`.

For `CONFIG_DYNAMIC_DEBUG` kernels, any settings given at boot-time (or enabled by `-DDEBUG` flag during compilation) can be disabled later via the `debugfs` interface if the debug messages are no longer needed:

```
echo "module module_name -p" > <debugfs>/dynamic_debug/control
```

## 12.7 Examples

```
// enable the message at line 1603 of file svcsock.c
nullarbor:~ # echo -n 'file svcsock.c line 1603 +p' >
                <debugfs>/dynamic_debug/control

// enable all the messages in file svcsock.c
nullarbor:~ # echo -n 'file svcsock.c +p' >
                <debugfs>/dynamic_debug/control

// enable all the messages in the NFS server module
nullarbor:~ # echo -n 'module nfsd +p' >
```

(continues on next page)

(continued from previous page)

```
<debugfs>/dynamic_debug/control

// enable all 12 messages in the function svc_process()
nullarbor:~ # echo -n 'func svc_process +p' >
               <debugfs>/dynamic_debug/control

// disable all 12 messages in the function svc_process()
nullarbor:~ # echo -n 'func svc_process -p' >
               <debugfs>/dynamic_debug/control

// enable messages for NFS calls READ, READLINK, REaddir and REaddir+.
nullarbor:~ # echo -n 'format "nfsd: READ" +p' >
               <debugfs>/dynamic_debug/control

// enable messages in files of which the paths include string "usb"
nullarbor:~ # echo -n '*usb* +p' > <debugfs>/dynamic_debug/control

// enable all messages
nullarbor:~ # echo -n '+p' > <debugfs>/dynamic_debug/control

// add module, function to all enabled messages
nullarbor:~ # echo -n '+mf' > <debugfs>/dynamic_debug/control

// boot-args example, with newlines and comments for readability
Kernel command line: ...
// see whats going on in dyndbg=value processing
dynamic_debug.verbose=1
// enable pr_debugs in 2 builtins, #cmt is stripped
dyndbg="module params +p #cmt ; module sys +p"
// enable pr_debugs in 2 functions in a module loaded later
pc87360.dyndbg="func pc87360_init_device +p; func pc87360_find +p"
```



## **EXPLAINING THE “NO WORKING INIT FOUND.” BOOT HANG MESSAGE**

**Authors** Andreas Mohr <andi at lisas period de> Cristian Souza  
<cristianmsbr at gmail period com>

This document provides some high-level reasons for failure (listed roughly in order of execution) to load the init binary.

- 1) **Unable to mount root FS:** Set “debug” kernel parameter (in bootloader config file or CONFIG\_CMDLINE) to get more detailed kernel messages.
- 2) **init binary doesn't exist on rootfs:** Make sure you have the correct root FS type (and root= kernel parameter points to the correct partition), required drivers such as storage hardware (such as SCSI or USB!) and filesystem (ext3, jffs2, etc.) are builtin (alternatively as modules, to be pre-loaded by an initrd).
- 3) **Broken console device:** Possibly a conflict in console= setup -> initial console unavailable. E.g. some serial consoles are unreliable due to serial IRQ issues (e.g. missing interrupt-based configuration). Try using a different console= device or e.g. netconsole=.
- 4) **Binary exists but dependencies not available:** E.g. required library dependencies of the init binary such as /lib/ld-linux.so.2 missing or broken. Use `readelf -d <INIT>|grep NEEDED` to find out which libraries are required.
- 5) **Binary cannot be loaded:** Make sure the binary's architecture matches your hardware. E.g. i386 vs. x86\_64 mismatch, or trying to load x86 on ARM hardware. In case you tried loading a non-binary file here (shell script?), you should make sure that the script specifies an interpreter in its shebang header line (`#!/...`) that is fully working (including its library dependencies). And before tackling scripts, better first test a simple non-script binary such as /bin/sh and confirm its successful execution. To find out more, add code to `init/main.c` to display `kernel_execve()`s return values.

Please extend this explanation whenever you find new failure causes (after all loading the init binary is a CRITICAL and hard transition step which needs to be made as painless as possible), then submit a patch to LKML. Further TODOs:

- Implement the various `run_init_process()` invocations via a struct array which can then store the `kernel_execve()` result value and on failure log it all by iterating over **all** results (very important usability fix).

- Try to make the implementation itself more helpful in general, e.g. by providing additional error messages at affected places.

## **DOCUMENTATION FOR KDUMP - THE KEXEC-BASED CRASH DUMPING SOLUTION**

This document includes overview, setup and installation, and analysis information.

### **14.1 Documentation for Kdump - The kexec-based Crash Dumping Solution**

This document includes overview, setup and installation, and analysis information.

#### **14.1.1 Overview**

Kdump uses kexec to quickly boot to a dump-capture kernel whenever a dump of the system kernel's memory needs to be taken (for example, when the system panics). The system kernel's memory image is preserved across the reboot and is accessible to the dump-capture kernel.

You can use common commands, such as `cp` and `scp`, to copy the memory image to a dump file on the local disk, or across the network to a remote system.

Kdump and kexec are currently supported on the x86, x86\_64, ppc64, ia64, s390x, arm and arm64 architectures.

When the system kernel boots, it reserves a small section of memory for the dump-capture kernel. This ensures that ongoing Direct Memory Access (DMA) from the system kernel does not corrupt the dump-capture kernel. The `kexec -p` command loads the dump-capture kernel into this reserved memory.

On x86 machines, the first 640 KB of physical memory is needed to boot, regardless of where the kernel loads. Therefore, kexec backs up this region just before rebooting into the dump-capture kernel.

Similarly on PPC64 machines first 32KB of physical memory is needed for booting regardless of where the kernel is loaded and to support 64K page size kexec backs up the first 64KB memory.

For s390x, when kdump is triggered, the crashkernel region is exchanged with the region [0, crashkernel region size] and then the kdump kernel runs in [0, crashkernel region size]. Therefore no relocatable kernel is needed for s390x.

All of the necessary information about the system kernel's core image is encoded in the ELF format, and stored in a reserved area of memory before a crash. The

physical address of the start of the ELF header is passed to the dump-capture kernel through the `elfcorehdr= boot` parameter. Optionally the size of the ELF header can also be passed when using the `elfcorehdr=[size[KMG]]@[offset[KMG]]` syntax.

With the dump-capture kernel, you can access the memory image through `/proc/vmcore`. This exports the dump as an ELF-format file that you can write out using file copy commands such as `cp` or `scp`. Further, you can use analysis tools such as the GNU Debugger (GDB) and the Crash tool to debug the dump file. This method ensures that the dump pages are correctly ordered.

### 14.1.2 Setup and Installation

#### Install kexec-tools

- 1) Login as the root user.
- 2) Download the kexec-tools user-space package from the following URL:

<http://kernel.org/pub/linux/utils/kernel/kexec/kexec-tools.tar.gz>

This is a symlink to the latest version.

The latest kexec-tools git tree is available at:

- [git://git.kernel.org/pub/scm/utils/kernel/kexec/kexec-tools.git](http://git.kernel.org/pub/scm/utils/kernel/kexec/kexec-tools.git)
- <http://www.kernel.org/pub/scm/utils/kernel/kexec/kexec-tools.git>

There is also a gitweb interface available at <http://www.kernel.org/git/?p=utils/kernel/kexec/kexec-tools.git>

More information about kexec-tools can be found at <http://horms.net/projects/kexec/>

- 3) Unpack the tarball with the tar command, as follows:

```
tar xvpzf kexec-tools.tar.gz
```

- 4) Change to the kexec-tools directory, as follows:

```
cd kexec-tools-VERSION
```

- 5) Configure the package, as follows:

```
./configure
```

- 6) Compile the package, as follows:

```
make
```

- 7) Install the package, as follows:

```
make install
```

## Build the system and dump-capture kernels

There are two possible methods of using Kdump.

- 1) Build a separate custom dump-capture kernel for capturing the kernel core dump.
- 2) Or use the system kernel binary itself as dump-capture kernel and there is no need to build a separate dump-capture kernel. This is possible only with the architectures which support a relocatable kernel. As of today, i386, x86\_64, ppc64, ia64, arm and arm64 architectures support relocatable kernel.

Building a relocatable kernel is advantageous from the point of view that one does not have to build a second kernel for capturing the dump. But at the same time one might want to build a custom dump capture kernel suitable to his needs.

Following are the configuration setting required for system and dump-capture kernels for enabling kdump support.

### System kernel config options

- 1) Enable “kexec system call” in “Processor type and features.” :

```
CONFIG_KEXEC=y
```

- 2) Enable “sysfs file system support” in “Filesystem” -> “Pseudo filesystems.” This is usually enabled by default:

```
CONFIG_SYSFS=y
```

Note that “sysfs file system support” might not appear in the “Pseudo filesystems” menu if “Configure standard kernel features (for small systems)” is not enabled in “General Setup.” In this case, check the .config file itself to ensure that sysfs is turned on, as follows:

```
grep 'CONFIG_SYSFS' .config
```

- 3) Enable “Compile the kernel with debug info” in “Kernel hacking.” :

```
CONFIG_DEBUG_INFO=Y
```

This causes the kernel to be built with debug symbols. The dump analysis tools require a vmlinux with debug symbols in order to read and analyze a dump file.

### Dump-capture kernel config options (Arch Independent)

- 1) Enable “kernel crash dumps” support under “Processor type and features” :

```
CONFIG_CRASH_DUMP=y
```

- 2) Enable “/proc/vmcore support” under “Filesystems” -> “Pseudo filesystems” :

```
CONFIG_PROC_VMCORE=y
```

(CONFIG\_PROC\_VMCORE is set by default when CONFIG\_CRASH\_DUMP is selected.)

### Dump-capture kernel config options (Arch Dependent, i386 and x86\_64)

- 1) On i386, enable high memory support under “Processor type and features” :

```
CONFIG_HIGHMEM64G=y
```

or:

```
CONFIG_HIGHMEM4G
```

- 2) On i386 and x86\_64, disable symmetric multi-processing support under “Processor type and features” :

```
CONFIG_SMP=n
```

(If CONFIG\_SMP=y, then specify maxcpus=1 on the kernel command line when loading the dump-capture kernel, see section “Load the Dump-capture Kernel” .)

- 3) If one wants to build and use a relocatable kernel, Enable “Build a relocatable kernel” support under “Processor type and features” :

```
CONFIG_RELOCATABLE=y
```

- 4) Use a suitable value for “Physical address where the kernel is loaded” (under “Processor type and features”). This only appears when “kernel crash dumps” is enabled. A suitable value depends upon whether kernel is relocatable or not.

If you are using a relocatable kernel use CONFIG\_PHYSICAL\_START=0x100000 This will compile the kernel for physical address 1MB, but given the fact kernel is relocatable, it can be run from any physical address hence kexec boot loader will load it in memory region reserved for dump-capture kernel.

Otherwise it should be the start of memory region reserved for second kernel using boot parameter “crashkernel=Y@X” . Here X is start of memory region reserved for dump-capture kernel. Generally X is 16MB (0x1000000). So you can set CONFIG\_PHYSICAL\_START=0x1000000

- 5) Make and install the kernel and its modules. DO NOT add this kernel to the boot loader configuration files.

### Dump-capture kernel config options (Arch Dependent, ppc64)

- 1) Enable “Build a kdump crash kernel” support under “Kernel” options:

```
CONFIG_CRASH_DUMP=y
```

- 2) Enable “Build a relocatable kernel” support:

```
CONFIG_RELOCATABLE=y
```

Make and install the kernel and its modules.

### Dump-capture kernel config options (Arch Dependent, ia64)

- No specific options are required to create a dump-capture kernel for ia64, other than those specified in the arch independent section above. This means that it is possible to use the system kernel as a dump-capture kernel if desired.

The crashkernel region can be automatically placed by the system kernel at run time. This is done by specifying the base address as 0, or omitting it all together:

```
crashkernel=256M@0
```

or:

```
crashkernel=256M
```

If the start address is specified, note that the start address of the kernel will be aligned to 64Mb, so if the start address is not then any space below the alignment point will be wasted.

### Dump-capture kernel config options (Arch Dependent, arm)

- To use a relocatable kernel, Enable “AUTO\_ZRELADDR” support under “Boot” options:

```
AUTO_ZRELADDR=y
```

### Dump-capture kernel config options (Arch Dependent, arm64)

- Please note that kvm of the dump-capture kernel will not be enabled on non-VHE systems even if it is configured. This is because the CPU will not be reset to EL2 on panic.

#### 14.1.3 Extended crashkernel syntax

While the “`crashkernel=size[@offset]`” syntax is sufficient for most configurations, sometimes it’s handy to have the reserved memory dependent on the value of System RAM – that’s mostly for distributors that pre-setup the kernel command line to avoid a unbootable system after some memory has been removed from the machine.

The syntax is:

```
crashkernel=<range1>:<size1>[,<range2>:<size2>, ...][@offset]
range=start-[end]
```

For example:

```
crashkernel=512M-2G:64M,2G-:128M
```

This would mean:

- 1) if the RAM is smaller than 512M, then don’t reserve anything (this is the “rescue” case)
- 2) if the RAM size is between 512M and 2G (exclusive), then reserve 64M
- 3) if the RAM size is larger than 2G, then reserve 128M

#### 14.1.4 Boot into System Kernel

- 1) Update the boot loader (such as grub, yaboot, or lilo) configuration files as necessary.
- 2) Boot the system kernel with the boot parameter “`crashkernel=Y@X`”, where Y specifies how much memory to reserve for the dump-capture kernel and X specifies the beginning of this reserved memory. For example, “`crashkernel=64M@16M`” tells the system kernel to reserve 64 MB of memory starting at physical address 0x01000000 (16MB) for the dump-capture kernel.

On x86 and x86\_64, use “`crashkernel=64M@16M`” .

On ppc64, use “`crashkernel=128M@32M`” .

On ia64, `256M@256M` is a generous value that typically works. The region may be automatically placed on ia64, see the dump-capture kernel config option notes above. If use sparse memory, the size should be rounded to GRANULE boundaries.

On s390x, typically use “`crashkernel=xxM`” . The value of xx is dependent on the memory consumption of the kdump system. In general this is not dependent on the memory size of the production system.

On arm, the use of “crashkernel=Y@X” is no longer necessary; the kernel will automatically locate the crash kernel image within the first 512MB of RAM if X is not given.

On arm64, use “crashkernel=Y[@X]” . Note that the start address of the kernel, X if explicitly specified, must be aligned to 2MiB (0x200000).

### 14.1.5 Load the Dump-capture Kernel

After booting to the system kernel, dump-capture kernel needs to be loaded.

Based on the architecture and type of image (relocatable or not), one can choose to load the uncompressed vmlinux or compressed bzImage/vmlinuz of dump-capture kernel. Following is the summary.

For i386 and x86\_64:

- Use vmlinux if kernel is not relocatable.
- Use bzImage/vmlinuz if kernel is relocatable.

For ppc64:

- Use vmlinux

For ia64:

- Use vmlinux or vmlinuz.gz

For s390x:

- Use image or bzImage

For arm:

- Use zImage

For arm64:

- Use vmlinux or Image

If you are using an uncompressed vmlinux image then use following command to load dump-capture kernel:

```
kexec -p <dump-capture-kernel-vmlinux-image> \  
--initrd=<initrd-for-dump-capture-kernel> --args-linux \  
--append="root=<root-dev> <arch-specific-options>"
```

If you are using a compressed bzImage/vmlinuz, then use following command to load dump-capture kernel:

```
kexec -p <dump-capture-kernel-bzImage> \  
--initrd=<initrd-for-dump-capture-kernel> \  
--append="root=<root-dev> <arch-specific-options>"
```

If you are using a compressed zImage, then use following command to load dump-capture kernel:

```
kexec --type zImage -p <dump-capture-kernel-bzImage> \  
--initrd=<initrd-for-dump-capture-kernel> \  
--dtb=<dtb-for-dump-capture-kernel> \  
--append="root=<root-dev> <arch-specific-options>"
```

If you are using an uncompressed Image, then use following command to load dump-capture kernel:

```
kexec -p <dump-capture-kernel-Image> \  
--initrd=<initrd-for-dump-capture-kernel> \  
--append="root=<root-dev> <arch-specific-options>"
```

Please note, that `-args-linux` does not need to be specified for ia64. It is planned to make this a no-op on that architecture, but for now it should be omitted

Following are the arch specific command line options to be used while loading dump-capture kernel.

For i386, x86\_64 and ia64:

```
"1 irqpoll maxcpus=1 reset_devices"
```

For ppc64:

```
"1 maxcpus=1 noirqdistrib reset_devices"
```

For s390x:

```
"1 maxcpus=1 cgroup_disable=memory"
```

For arm:

```
"1 maxcpus=1 reset_devices"
```

For arm64:

```
"1 maxcpus=1 reset_devices"
```

Notes on loading the dump-capture kernel:

- By default, the ELF headers are stored in ELF64 format to support systems with more than 4GB memory. On i386, kexec automatically checks if the physical RAM size exceeds the 4 GB limit and if not, uses ELF32. So, on non-PAE systems, ELF32 is always used.

The `-elf32-core-headers` option can be used to force the generation of ELF32 headers. This is necessary because GDB currently cannot open vmcore files with ELF64 headers on 32-bit systems.

- The `"irqpoll"` boot parameter reduces driver initialization failures due to shared interrupts in the dump-capture kernel.
- You must specify `<root-dev>` in the format corresponding to the root device name in the output of mount command.
- Boot parameter `"1"` boots the dump-capture kernel into single-user mode without networking. If you want networking, use `"3"`.
- We generally don't have to bring up a SMP kernel just to capture the dump. Hence generally it is useful either to build a UP dump-capture kernel or specify `maxcpus=1` option while loading dump-capture kernel. Note, though max-

cpus always works, you had better replace it with `nr_cpus` to save memory if supported by the current ARCH, such as x86.

- You should enable multi-cpu support in dump-capture kernel if you intend to use multi-thread programs with it, such as parallel dump feature of `make-dumpfile`. Otherwise, the multi-thread program may have a great performance degradation. To enable multi-cpu support, you should bring up an SMP dump-capture kernel and specify `maxcpus/nr_cpus`, `disable_cpu_apicid=[X]` options while loading it.
- For s390x there are two `kdump` modes: If a ELF header is specified with the `elfcorehdr=` kernel parameter, it is used by the `kdump` kernel as it is done on all other architectures. If no `elfcorehdr=` kernel parameter is specified, the s390x `kdump` kernel dynamically creates the header. The second mode has the advantage that for CPU and memory hotplug, `kdump` has not to be reloaded with `kexec_load()`.
- For s390x systems with many attached devices the “`cio_ignore`” kernel parameter should be used for the `kdump` kernel in order to prevent allocation of kernel memory for devices that are not relevant for `kdump`. The same applies to systems that use SCSI/FCP devices. In that case the “`allow_lun_scan`” `zfcplib` module parameter should be set to zero before setting FCP devices online.

### 14.1.6 Kernel Panic

After successfully loading the dump-capture kernel as previously described, the system will reboot into the dump-capture kernel if a system crash is triggered. Trigger points are located in `panic()`, `die()`, `die_nmi()` and in the `sysrq` handler (ALT-SysRq-c).

The following conditions will execute a crash trigger point:

If a hard lockup is detected and “NMI watchdog” is configured, the system will boot into the dump-capture kernel ( `die_nmi()` ).

If `die()` is called, and it happens to be a thread with `pid 0` or `1`, or `die()` is called inside interrupt context or `die()` is called and `panic_on_oops` is set, the system will boot into the dump-capture kernel.

On powerpc systems when a soft-reset is generated, `die()` is called by all cpus and the system will boot into the dump-capture kernel.

For testing purposes, you can trigger a crash by using “ALT-SysRq-c” , “`echo c > /proc/sysrq-trigger`” or write a module to force the panic.

### 14.1.7 Write Out the Dump File

After the dump-capture kernel is booted, write out the dump file with the following command:

```
cp /proc/vmcore <dump-file>
```

### 14.1.8 Analysis

Before analyzing the dump image, you should reboot into a stable kernel.

You can do limited analysis using GDB on the dump file copied out of /proc/vmcore. Use the debug vmlinux built with -g and run the following command:

```
gdb vmlinux <dump-file>
```

Stack trace for the task on processor 0, register display, and memory display work fine.

Note: GDB cannot analyze core files generated in ELF64 format for x86. On systems with a maximum of 4GB of memory, you can generate ELF32-format headers using the `-elf32-core-headers` kernel option on the dump kernel.

You can also use the Crash utility to analyze dump files in Kdump format. Crash is available on Dave Anderson's site at the following URL:

<http://people.redhat.com/~anderson/>

### 14.1.9 Trigger Kdump on WARN()

The kernel parameter, `panic_on_warn`, calls `panic()` in all `WARN()` paths. This will cause a kdump to occur at the `panic()` call. In cases where a user wants to specify this during runtime, `/proc/sys/kernel/panic_on_warn` can be set to 1 to achieve the same behaviour.

### 14.1.10 Trigger Kdump on add\_taint()

The kernel parameter `panic_on_taint` facilitates a conditional call to `panic()` from within `add_taint()` whenever the value set in this bitmask matches with the bit flag being set by `add_taint()`. This will cause a kdump to occur at the `add_taint()->panic()` call.

### 14.1.11 Contact

- Vivek Goyal (vgoyal@redhat.com)
- Maneesh Soni (maneesh@in.ibm.com)

### 14.1.12 GDB macros

```
#
# This file contains a few gdb macros (user defined commands) to
↳extract
# useful information from kernel crashdump (kdump) like stack
↳traces of
# all the processes or a particular process and trapinfo.
#
# These macros can be used by copying this file in .gdbinit (put in
↳home
# directory or current directory) or by invoking gdb command with
# --command=<command-file-name> option
#
# Credits:
# Alexander Nyberg <alexn@telia.com>
# V Srivatsa <vatsa@in.ibm.com>
# Maneesh Soni <maneesh@in.ibm.com>
#

define bttnobp
    set $tasks_off=((size_t)&((struct task_struct *)0)->tasks)
    set $pid_off=((size_t)&((struct task_struct *)0)->thread_
↳group.next)
    set $init_t=&init_task
    set $next_t=(((char *)($init_t->tasks).next) - $tasks_off)
    set var $stacksize = sizeof(union thread_union)
    while ($next_t != $init_t)
        set $next_t=(struct task_struct *)$next_t
        printf "\npid %d; comm %s:\n", $next_t.pid, $next_t.
↳comm

        printf "=====\n"
        set var $stackp = $next_t.thread.sp
        set var $stack_top = ($stackp & ~($stacksize - 1))
↳+ $stacksize

        while ($stackp < $stack_top)
            if (*($stackp) > _stext && *($stackp) < _
↳sinittext)

                info symbol *($stackp)
            end
            set $stackp += 4
        end
        set $next_th=(((char *)$next_t->thread_group.next)
↳- $pid_off)
end
```

```
        while ($next_th != $next_t)
            set $next_th=(struct task_struct *)$next_th
            printf "\npid %d; comm %s:\n", $next_t.pid,
→$next_t.comm
                printf "=====\n"
                set var $stackp = $next_t.thread.sp
                set var $stack_top = ($stackp & ~(
→$stacksize - 1)) + stacksize

                    while ($stackp < $stack_top)
                        if (*( $stackp) > _stext && *(
→$stackp) < _sinittext)
                            info symbol *( $stackp)
                            end
                            set $stackp += 4
                        end
                        set $next_th=(((char *)$next_th->thread_
→group.next) - $pid_off)
                    end
                    set $next_t=(char *)($next_t->tasks.next) - $tasks_
→off
                end
            end
        end
document bttnobp
    dump all thread stack traces on a kernel compiled with !
→CONFIG_FRAME_POINTER
end

define btthreadstack
    set var $pid_task = $arg0

    printf "\npid %d; comm %s:\n", $pid_task.pid, $pid_task.comm
    printf "task struct: "
    print $pid_task
    printf "=====\n"
    set var $stackp = $pid_task.thread.sp
    set var $stacksize = sizeof(union thread_union)
    set var $stack_top = ($stackp & ~($stacksize - 1)) +
→$stacksize
    set var $stack_bot = ($stackp & ~($stacksize - 1))

    set $stackp = *((unsigned long *) $stackp)
    while (($stackp < $stack_top) && ($stackp > $stack_bot))
        set var $addr = *((unsigned long *) $stackp) + 1
        info symbol $addr
        set $stackp = *((unsigned long *) $stackp)
    end
end
document btthreadstack
    dump a thread stack using the given task structure pointer
end
```

```

define btt
    set $tasks_off=((size_t)&((struct task_struct *)0)->tasks)
    set $pid_off=((size_t)&((struct task_struct *)0)->thread_
↳group.next)
    set $init_t=&init_task
    set $next_t=(((char *)($init_t->tasks).next) - $tasks_off)
    while ($next_t != $init_t)
        set $next_t=(struct task_struct *)$next_t
        btthreadstack $next_t

        set $next_th=(((char *)$next_t->thread_group.next)
↳- $pid_off)
        while ($next_th != $next_t)
            set $next_th=(struct task_struct *)$next_th
            btthreadstack $next_th
            set $next_th=(((char *)$next_th->thread_
↳group.next) - $pid_off)
        end
        set $next_t=(char *)($next_t->tasks.next) - $tasks_
↳off
    end
end
document btt
    dump all thread stack traces on a kernel compiled with
↳CONFIG_FRAME_POINTER
end

define btpid
    set var $pid = $arg0
    set $tasks_off=((size_t)&((struct task_struct *)0)->tasks)
    set $pid_off=((size_t)&((struct task_struct *)0)->thread_
↳group.next)
    set $init_t=&init_task
    set $next_t=(((char *)($init_t->tasks).next) - $tasks_off)
    set var $pid_task = 0

    while ($next_t != $init_t)
        set $next_t=(struct task_struct *)$next_t

        if ($next_t.pid == $pid)
            set $pid_task = $next_t
        end

        set $next_th=(((char *)$next_t->thread_group.next)
↳- $pid_off)
        while ($next_th != $next_t)
            set $next_th=(struct task_struct *)$next_th
            if ($next_th.pid == $pid)
                set $pid_task = $next_th
            end
        end
    end
end

```

```

                end
                set $next_th=(((char *)$next_th->thread_
↪group.next) - $pid_off)
                end
                set $next_t=(char *)($next_t->tasks.next) - $tasks_
↪off
            end

            btthreadstack $pid_task
        end
    document btpid
        backtrace of pid
    end

define trapinfo
    set var $pid = $arg0
    set $tasks_off=((size_t)&((struct task_struct *)0)->tasks)
    set $pid_off=((size_t)&((struct task_struct *)0)->thread_
↪group.next)
    set $init_t=&init_task
    set $next_t=(((char *)($init_t->tasks).next) - $tasks_off)
    set var $pid_task = 0

    while ($next_t != $init_t)
        set $next_t=(struct task_struct *)$next_t

        if ($next_t.pid == $pid)
            set $pid_task = $next_t
        end

        set $next_th=(((char *)$next_t->thread_group.next)
↪- $pid_off)
        while ($next_th != $next_t)
            set $next_th=(struct task_struct *)$next_th
            if ($next_th.pid == $pid)
                set $pid_task = $next_th
            end
            set $next_th=(((char *)$next_th->thread_
↪group.next) - $pid_off)
        end
        set $next_t=(char *)($next_t->tasks.next) - $tasks_
↪off
    end

    printf "Trapno %ld, cr2 0x%lx, error_code %ld\n", $pid_task.
↪thread.trap_no, \
                $pid_task.thread.cr2, $pid_task.
↪thread.error_code

end
```

```

document trapinfo
    Run info threads and lookup pid of thread #1
    'trapinfo <pid>' will tell you by which trap & possibly
    address the kernel panicked.
end

define dump_log_idx
    set $idx = $arg0
    if ($argc > 1)
        set $prev_flags = $arg1
    else
        set $prev_flags = 0
    end
    set $msg = ((struct printk_log *) (log_buf + $idx))
    set $prefix = 1
    set $newline = 1
    set $log = log_buf + $idx + sizeof(*$msg)

    # prev & LOG_CONT && !(msg->flags & LOG_PREIX)
    if (($prev_flags & 8) && !(msg->flags & 4))
        set $prefix = 0
    end

    # msg->flags & LOG_CONT
    if ($msg->flags & 8)
        # (prev & LOG_CONT && !(prev & LOG_NEWLINE))
        if (($prev_flags & 8) && !(prev_flags & 2))
            set $prefix = 0
        end
        # (!(msg->flags & LOG_NEWLINE))
        if (!(msg->flags & 2))
            set $newline = 0
        end
    end

    if ($prefix)
        printf "[%5lu.%06lu] ", $msg->ts_nsec / 1000000000,
→$msg->ts_nsec % 1000000000
    end
    if ($msg->text_len != 0)
        eval "printf \"%%%d.%ds\", $log", $msg->text_len,
→$msg->text_len
    end
    if ($newline)
        printf "\n"
    end
    if ($msg->dict_len > 0)
        set $dict = $log + $msg->text_len
        set $idx = 0
        set $line = 1
        while ($idx < $msg->dict_len)

```

```
        if ($line)
            printf " "
            set $line = 0
        end
        set $c = $dict[$idx]
        if ($c == '\0')
            printf "\n"
            set $line = 1
        else
            if ($c < ' ' || $c >= 127 || $c == ↵
↵ '\\\')
                printf "\\x%02x", $c
            else
                printf "%c", $c
            end
        end
        set $idx = $idx + 1
    end
    printf "\n"
end

end
document dump_log_idx
    Dump a single log given its index in the log buffer. The ↵
↵ first
↵ and
    parameter is the index into log_buf, the second is optional ↵
    specified the previous log buffer's flags, used for properly
    formatting continued lines.
end

define dmesg
    set $i = log_first_idx
    set $end_idx = log_first_idx
    set $prev_flags = 0

    while (1)
        set $msg = ((struct printk_log *) (log_buf + $i))
        if ($msg->len == 0)
            set $i = 0
        else
            dump_log_idx $i $prev_flags
            set $i = $i + $msg->len
            set $prev_flags = $msg->flags
        end
        if ($i == $end_idx)
            loop_break
        end
    end
end

end
document dmesg
    print the kernel ring buffer
```

end

## 14.2 VMCOREINFO

### 14.2.1 What is it?

VMCOREINFO is a special ELF note section. It contains various information from the kernel like structure size, page size, symbol values, field offsets, etc. These data are packed into an ELF note section and used by user-space tools like crash and makedumpfile to analyze a kernel's memory layout.

### 14.2.2 Common variables

#### **init\_uts\_ns.name.release**

The version of the Linux kernel. Used to find the corresponding source code from which the kernel has been built. For example, crash uses it to find the corresponding vmlinux in order to process vmcore.

#### **PAGE\_SIZE**

The size of a page. It is the smallest unit of data used by the memory management facilities. It is usually 4096 bytes of size and a page is aligned on 4096 bytes. Used for computing page addresses.

#### **init\_uts\_ns**

The UTS namespace which is used to isolate two specific elements of the system that relate to the `uname(2)` system call. It is named after the data structure used to store information returned by the `uname(2)` system call.

User-space tools can get the kernel name, host name, kernel release number, kernel version, architecture name and OS type from it.

#### **node\_online\_map**

An array `node_states[N_ONLINE]` which represents the set of online nodes in a system, one bit position per node number. Used to keep track of which nodes are in the system and online.

### **swapper\_pg\_dir**

The global page directory pointer of the kernel. Used to translate virtual to physical addresses.

### **\_stext**

Defines the beginning of the text section. In general, `_stext` indicates the kernel start address. Used to convert a virtual address from the direct kernel map to a physical address.

### **vmap\_area\_list**

Stores the virtual area list. `makedumpfile` gets the `vmalloc` start value from this variable and its value is necessary for `vmalloc` translation.

### **mem\_map**

Physical addresses are translated to struct pages by treating them as an index into the `mem_map` array. Right-shifting a physical address `PAGE_SHIFT` bits converts it into a page frame number which is an index into that `mem_map` array.

Used to map an address to the corresponding struct page.

### **contig\_page\_data**

`Makedumpfile` gets the `pghost_data` structure from this symbol, which is used to describe the memory layout.

User-space tools use this to exclude free pages when dumping memory.

### **mem\_section|(mem\_section, NR\_SECTION\_ROOTS)|(mem\_section, section\_mem\_map)**

The address of the `mem_section` array, its length, structure size, and the `section_mem_map` offset.

It exists in the sparse memory mapping model, and it is also somewhat similar to the `mem_map` variable, both of them are used to translate an address.

**page**

The size of a page structure. struct page is an important data structure and it is widely used to compute contiguous memory.

**pglist\_data**

The size of a pglist\_data structure. This value is used to check if the pglist\_data structure is valid. It is also used for checking the memory type.

**zone**

The size of a zone structure. This value is used to check if the zone structure has been found. It is also used for excluding free pages.

**free\_area**

The size of a free\_area structure. It indicates whether the free\_area structure is valid or not. Useful when excluding free pages.

**list\_head**

The size of a list\_head structure. Used when iterating lists in a post-mortem analysis session.

**nodemask\_t**

The size of a nodemask\_t type. Used to compute the number of online nodes.

**(page, flags|\_refcount|mapping|lru|\_mapcount|private|compound\_dtor|compound\_ord**

User-space tools compute their values based on the offset of these variables. The variables are used when excluding unnecessary pages.

**(pglist\_data, node\_zones|nr\_zones|node\_mem\_map|node\_start\_pfn|node\_spanned\_pag**

On NUMA machines, each NUMA node has a pg\_data\_t to describe its memory layout. On UMA machines there is a single pglist\_data which describes the whole memory.

These values are used to check the memory type and to compute the virtual address for memory map.

### **(zone, free\_area|vm\_stat|spanned\_pages)**

Each node is divided into a number of blocks called zones which represent ranges within memory. A zone is described by a structure zone.

User-space tools compute required values based on the offset of these variables.

### **(free\_area, free\_list)**

Offset of the free\_list' s member. This value is used to compute the number of free pages.

Each zone has a free\_area structure array called free\_area[MAX\_ORDER]. The free\_list represents a linked list of free page blocks.

### **(list\_head, next|prev)**

Offsets of the list\_head' s members. list\_head is used to define a circular linked list. User-space tools need these in order to traverse lists.

### **(vmap\_area, va\_start|list)**

Offsets of the vmap\_area' s members. They carry vmalloc-specific information. Makedumpfile gets the start address of the vmalloc region from this.

### **(zone.free\_area, MAX\_ORDER)**

Free areas descriptor. User-space tools use this value to iterate the free\_area ranges. MAX\_ORDER is used by the zone buddy allocator.

### **log\_first\_idx**

Index of the first record stored in the buffer log\_buf. Used by user-space tools to read the strings in the log\_buf.

### **log\_buf**

Console output is written to the ring buffer log\_buf at index log\_first\_idx. Used to get the kernel log.

**log\_buf\_len**

log\_buf's length.

**clear\_idx**

The index that the next printk() record to read after the last clear command. It indicates the first record after the last SYSLOG\_ACTION\_CLEAR, like issued by 'dmesg -c'. Used by user-space tools to dump the dmesg log.

**log\_next\_idx**

The index of the next record to store in the buffer log\_buf. Used to compute the index of the current buffer position.

**printk\_log**

The size of a structure printk\_log. Used to compute the size of messages, and extract dmesg log. It encapsulates header information for log\_buf, such as timestamp, syslog level, etc.

**(printk\_log, ts\_nsec|len|text\_len|dict\_len)**

It represents field offsets in struct printk\_log. User space tools parse it and check whether the values of printk\_log's members have been changed.

**(free\_area.free\_list, MIGRATE\_TYPES)**

The number of migrate types for pages. The free\_list is described by the array. Used by tools to compute the number of free pages.

**NR\_FREE\_PAGES**

On linux-2.6.21 or later, the number of free pages is in vm\_stat[NR\_FREE\_PAGES]. Used to get the number of free pages.

**PG\_lru|PG\_private|PG\_swapcache|PG\_swapbacked|PG\_slab|PG\_hwpoison|PG\_head\_ma**

Page attributes. These flags are used to filter various unnecessary for dumping pages.

### **PAGE\_BUDDY\_MAPCOUNT\_VALUE(~PG\_buddy)|PAGE\_OFFLINE\_MAPCOUNT\_VALUE(~PG**

More page attributes. These flags are used to filter various unnecessary for dumping pages.

### **HUGETLB\_PAGE\_DTOR**

The HUGETLB\_PAGE\_DTOR flag denotes hugetlbfs pages. Makedumpfile excludes these pages.

## **14.2.3 x86\_64**

### **phys\_base**

Used to convert the virtual address of an exported kernel symbol to its corresponding physical address.

### **init\_top\_pgt**

Used to walk through the whole page table and convert virtual addresses to physical addresses. The init\_top\_pgt is somewhat similar to swapper\_pg\_dir, but it is only used in x86\_64.

### **pgtable\_l5\_enabled**

User-space tools need to know whether the crash kernel was in 5-level paging mode.

### **node\_data**

This is a struct pglst\_data array and stores all NUMA nodes information. Makedumpfile gets the pglst\_data structure from it.

### **(node\_data, MAX\_NUMNODES)**

The maximum number of nodes in system.

### **KERNELOFFSET**

The kernel randomization offset. Used to compute the page offset. If KASLR is disabled, this value is zero.

## KERNEL\_IMAGE\_SIZE

Currently unused by Makedumpfile. Used to compute the module virtual address by Crash.

## sme\_mask

AMD-specific with SME support: it indicates the secure memory encryption mask. Makedumpfile tools need to know whether the crash kernel was encrypted. If SME is enabled in the first kernel, the crash kernel's page table entries (pgd/pud/pmd/pte) contain the memory encryption mask. This is used to remove the SME mask and obtain the true physical address.

Currently, sme\_mask stores the value of the C-bit position. If needed, additional SME-relevant info can be placed in that variable.

For example:

```
[ misc          ][ enc bit  ][ other misc SME info      ]
0000_0000_0000_0000_1000_0000_0000_0000_0000_0000_..._0000
63  59  55  51  47  43  39  35  31  27  ... 3
```

## 14.2.4 x86\_32

### X86\_PAE

Denotes whether physical address extensions are enabled. It has the cost of a higher page table lookup overhead, and also consumes more page table space per process. Used to check whether PAE was enabled in the crash kernel when converting virtual addresses to physical addresses.

## 14.2.5 ia64

### pgdat\_list|(pgdat\_list, MAX\_NUMNODES)

pg\_data\_t array storing all NUMA nodes information. MAX\_NUMNODES indicates the number of the nodes.

### node\_memblk|(node\_memblk, NR\_NODE\_MEMBLKS)

List of node memory chunks. Filled when parsing the SRAT table to obtain information about memory nodes. NR\_NODE\_MEMBLKS indicates the number of node memory chunks.

These values are used to compute the number of nodes the crashed kernel used.

### **node\_memblk\_s|(node\_memblk\_s, start\_paddr)|(node\_memblk\_s, size)**

The size of a struct `node_memblk_s` and the offsets of the `node_memblk_s`'s members. Used to compute the number of nodes.

### **PGTABLE\_3|PGTABLE\_4**

User-space tools need to know whether the crash kernel was in 3-level or 4-level paging mode. Used to distinguish the page table.

## **14.2.6 ARM64**

### **VA\_BITS**

The maximum number of bits for virtual addresses. Used to compute the virtual memory ranges.

### **kimage\_voffset**

The offset between the kernel virtual and physical mappings. Used to translate virtual to physical addresses.

### **PHYS\_OFFSET**

Indicates the physical address of the start of memory. Similar to `kimage_voffset`, which is used to translate virtual to physical addresses.

### **KERNELOFFSET**

The kernel randomization offset. Used to compute the page offset. If KASLR is disabled, this value is zero.

### **KERNELPACMASK**

The mask to extract the Pointer Authentication Code from a kernel virtual address.

## **14.2.7 arm**

### **ARM\_LPAE**

It indicates whether the crash kernel supports large physical address extensions. Used to translate virtual to physical addresses.

## **14.2.8 s390**

### **lowcore\_ptr**

An array with a pointer to the lowcore of every CPU. Used to print the psw and all registers information.

### **high\_memory**

Used to get the vmalloc\_start address from the high\_memory symbol.

### **(lowcore\_ptr, NR\_CPUS)**

The maximum number of CPUs.

## **14.2.9 powerpc**

### **node\_data|(node\_data, MAX\_NUMNODES)**

See above.

### **contig\_page\_data**

See above.

### **vmemmap\_list**

The vmemmap\_list maintains the entire vmemmap physical mapping. Used to get vmemmap list count and populated vmemmap regions info. If the vmemmap address translation information is stored in the crash kernel, it is used to translate vmemmap kernel virtual addresses.

### **mmu\_vmemmap\_psize**

The size of a page. Used to translate virtual to physical addresses.

### **mmu\_psize\_defs**

Page size definitions, i.e. 4k, 64k, or 16M.

Used to make vtop translations.

**vmemmap\_backing|(vmemmap\_backing, list)|(vmemmap\_backing, phys)|(vmemmap\_backing, virt\_addr)**

The vmemmap virtual address space management does not have a traditional page table to track which virtual struct pages are backed by a physical mapping. The virtual to physical mappings are tracked in a simple linked list format.

User-space tools need to know the offset of list, phys and virt\_addr when computing the count of vmemmap regions.

**mmu\_psize\_def|(mmu\_psize\_def, shift)**

The size of a struct mmu\_psize\_def and the offset of mmu\_psize\_def's member. Used in vtop translations.

### 14.2.10 sh

**node\_data|(node\_data, MAX\_NUMNODES)**

See above.

### X2TLB

Indicates whether the crashed kernel enabled SH extended mode.

## **PERFORMANCE MONITOR SUPPORT**

### **15.1 HiSilicon SoC uncore Performance Monitoring Unit (PMU)**

The HiSilicon SoC chip includes various independent system device PMUs such as L3 cache (L3C), Hydra Home Agent (HHA) and DDRC. These PMUs are independent and have hardware logic to gather statistics and performance information.

The HiSilicon SoC encapsulates multiple CPU and IO dies. Each CPU cluster (CCL) is made up of 4 cpu cores sharing one L3 cache; each CPU die is called Super CPU cluster (SCCL) and is made up of 6 CCLs. Each SCCL has two HHAs (0 - 1) and four DDRCs (0 - 3), respectively.

#### **15.1.1 HiSilicon SoC uncore PMU driver**

Each device PMU has separate registers for event counting, control and interrupt, and the PMU driver shall register perf PMU drivers like L3C, HHA and DDRC etc. The available events and configuration options shall be described in the sysfs, see:

```
/sys/devices/hisi_sccl{X}_<l3c{Y}/hha{Y}/ddrc{Y}>/, or  
/sys/bus/event_source/devices/hisi_sccl{X}_<l3c{Y}/hha{Y}/ddrc{Y}>. The  
“perf list” command shall list the available events from sysfs.
```

Each L3C, HHA and DDRC is registered as a separate PMU with perf. The PMU name will appear in event listing as hisi\_sccl<sccl-id>\_module<index-id>. where “sccl-id” is the identifier of the SCCL and “index-id” is the index of module.

e.g. hisi\_sccl3\_l3c0/rd\_hit\_cpipe is READ\_HIT\_CPIPE event of L3C index #0 in SCCL ID #3.

e.g. hisi\_sccl1\_hha0/rx\_operations is RX\_OPERATIONS event of HHA index #0 in SCCL ID #1.

The driver also provides a “cpumask” sysfs attribute, which shows the CPU core ID used to count the uncore PMU event.

Example usage of perf:

```
## perf list  
hisi_sccl3_l3c0/rd_hit_cpipe/ [kernel PMU event]  
-----  
hisi_sccl3_l3c0/wr_hit_cpipe/ [kernel PMU event]
```

(continues on next page)

(continued from previous page)

```

-----
hisi_sccl1_l3c0/rd_hit_cpipe/ [kernel PMU event]
-----
hisi_sccl1_l3c0/wr_hit_cpipe/ [kernel PMU event]
-----

$# perf stat -a -e hisi_sccl3_l3c0/rd_hit_cpipe/ sleep 5
$# perf stat -a -e hisi_sccl3_l3c0/config=0x02/ sleep 5

```

The current driver does not support sampling. So “perf record” is unsupported. Also attach to a task is unsupported as the events are all uncore.

Note: Please contact the maintainer for a complete list of events supported for the PMU devices in the SoC and its information if needed.

## 15.2 Freescale i.MX8 DDR Performance Monitoring Unit (PMU)

There are no performance counters inside the DRAM controller, so performance signals are brought out to the edge of the controller where a set of 4 x 32 bit counters is implemented. This is controlled by the CSV modes programmed in counter control register which causes a large number of PERF signals to be generated.

Selection of the value for each counter is done via the config registers. There is one register for each counter. Counter 0 is special in that it always counts “time” and when expired causes a lock on itself and the other counters and an interrupt is raised. If any other counter overflows, it continues counting, and no interrupt is raised.

The “format” directory describes format of the config (event ID) and config1 (AXI filtering) fields of the perf\_event\_attr structure, see /sys/bus/event\_source/devices/imx8\_ddr0/format/. The “events” directory describes the events types hardware supported that can be used with perf tool, see /sys/bus/event\_source/devices/imx8\_ddr0/events/. The “caps” directory describes filter features implemented in DDR PMU, see /sys/bus/event\_source/devices/imx8\_ddr0/caps/.

```

perf stat -a -e imx8_ddr0/cycles/ cmd
perf stat -a -e imx8_ddr0/read/,imx8_ddr0/write/ cmd

```

AXI filtering is only used by CSV modes 0x41 (axid-read) and 0x42 (axid-write) to count reading or writing matches filter setting. Filter setting is various from different DRAM controller implementations, which is distinguished by quirks in the driver. You also can dump info from userspace, filter in “caps” directory indicates whether PMU supports AXI ID filter or not; enhanced\_filter indicates whether PMU supports enhanced AXI ID filter or not. Value 0 for un-supported, and value 1 for supported.

- With DDR\_CAP\_AXI\_ID\_FILTER quirk(filter: 1, enhanced\_filter: 0). Filter is defined with two configuration parts: -AXI\_ID defines AxID matching value. -AXI\_MASKING defines which bits of AxID are meaningful for the matching.
  - 0: corresponding bit is masked.

- 1: corresponding bit is not masked, i.e. used to do the matching.

AXI\_ID and AXI\_MASKING are mapped on DPCR1 register in performance counter. When non-masked bits are matching corresponding AXI\_ID bits then counter is incremented. Perf counter is incremented if:

```
AXID && AXI_MASKING == AXI_ID && AXI_MASKING
```

This filter doesn't support filter different AXI ID for axid-read and axid-write event at the same time as this filter is shared between counters.

```
perf stat -a -e imx8_ddr0/axid-read,axi_mask=0xMMMM,axi_id=0xDDDD/ cmd
perf stat -a -e imx8_ddr0/axid-write,axi_mask=0xMMMM,axi_id=0xDDDD/
↪cmd
```

**Note:** axi\_mask is inverted in userspace(i.e. set bits are bits to mask), and it will be reverted in driver automatically. so that the user can just specify axi\_id to monitor a specific id, rather than having to specify axi\_mask.

```
perf stat -a -e imx8_ddr0/axid-read,axi_id=0x12/ cmd, which will
↪monitor ARID=0x12
```

- With DDR\_CAP\_AXI\_ID\_FILTER\_ENHANCED quirk(filter: 1, enhanced\_filter: 1). This is an extension to the DDR\_CAP\_AXI\_ID\_FILTER quirk which permits counting the number of bytes (as opposed to the number of bursts) from DDR read and write transactions concurrently with another set of data counters.

## 15.3 Qualcomm Technologies Level-2 Cache Performance Monitoring Unit (PMU)

This driver supports the L2 cache clusters found in Qualcomm Technologies Centriq SoCs. There are multiple physical L2 cache clusters, each with their own PMU. Each cluster has one or more CPUs associated with it.

There is one logical L2 PMU exposed, which aggregates the results from the physical PMUs.

The driver provides a description of its available events and configuration options in sysfs, see `/sys/devices/l2cache_0`.

The “format” directory describes the format of the events.

Events can be envisioned as a 2-dimensional array. Each column represents a group of events. There are 8 groups. Only one entry from each group can be in use at a time. If multiple events from the same group are specified, the conflicting events cannot be counted at the same time.

Events are specified as `0xCCG`, where `CC` is 2 hex digits specifying the code (array row) and `G` specifies the group (column) 0-7.

In addition there is a cycle counter event specified by the value `0xFE` which is outside the above scheme.

The driver provides a “cpumask” sysfs attribute which contains a mask consisting of one CPU per cluster which will be used to handle all the PMU events on that cluster.

Examples for use with perf:

```
perf stat -e l2cache_0/config=0x001/,l2cache_0/config=0x042/ -a sleep 1
perf stat -e l2cache_0/config=0xfe/ -C 2 sleep 1
```

The driver does not support sampling, therefore “perf record” will not work. Per-task perf sessions are not supported.

### 15.4 Qualcomm Datacenter Technologies L3 Cache Performance Monitoring Unit (PMU)

This driver supports the L3 cache PMUs found in Qualcomm Datacenter Technologies Centriq SoCs. The L3 cache on these SOCs is composed of multiple slices, shared by all cores within a socket. Each slice is exposed as a separate uncore perf PMU with device name `l3cache_<socket>_<instance>`. User space is responsible for aggregating across slices.

The driver provides a description of its available events and configuration options in sysfs, see `/sys/devices/l3cache*`. Given that these are uncore PMUs the driver also exposes a “cpumask” sysfs attribute which contains a mask consisting of one CPU per socket which will be used to handle all the PMU events on that socket.

The hardware implements 32bit event counters and has a flat 8bit event space exposed via the “event” format attribute. In addition to the 32bit physical counters the driver supports virtual 64bit hardware counters by using hardware counter chaining. This feature is exposed via the “lc” (long counter) format flag. E.g.:

```
perf stat -e l3cache_0_0/read-miss,lc/
```

Given that these are uncore PMUs the driver does not support sampling, therefore “perf record” will not work. Per-task perf sessions are not supported.

### 15.5 ARM Cache Coherent Network

CCN-504 is a ring-bus interconnect consisting of 11 crosspoints (XPs), with each crosspoint supporting up to two device ports, so nodes (devices) 0 and 1 are connected to crosspoint 0, nodes 2 and 3 to crosspoint 1 etc.

### 15.5.1 PMU (perf) driver

The CCN driver registers a perf PMU driver, which provides description of available events and configuration options in sysfs, see `/sys/bus/event_source/devices/ccn*`.

The “format” directory describes format of the config, config1 and config2 fields of the `perf_event_attr` structure. The “events” directory provides configuration templates for all documented events, that can be used with perf tool. For example “`xp_valid_flit`” is an equivalent of “`type=0x8,event=0x4`”. Other parameters must be explicitly specified.

For events originating from device, “node” defines its index.

Crosspoint PMU events require “xp” (index), “bus” (bus number) and “vc” (virtual channel ID).

Crosspoint watchpoint-based events (special “event” value 0xfe) require “xp” and “vc” as above plus “port” (device port index), “dir” (transmit/receive direction), comparator values ( “`cmp_l`” and “`cmp_h`” ) and “mask” , being index of the comparator mask.

Masks are defined separately from the event description (due to limited number of the config values) in the “`cmp_mask`” directory, with first 8 configurable by user and additional 4 hardcoded for the most frequent use cases.

Cycle counter is described by a “type” value 0xff and does not require any other settings.

The driver also provides a “`cpumask`” sysfs attribute, which contains a single CPU ID, of the processor which will be used to handle all the CCN PMU events. It is recommended that the user space tools request the events on this processor (if not, the `perf_event->cpu` value will be overwritten anyway). In case of this processor being offlined, the events are migrated to another one and the attribute is updated.

Example of perf tool use:

```
/ # perf list | grep ccn
  ccn/cycles/                                [Kernel PMU event]
<...>
  ccn/xp_valid_flit,xp=?,port=?,vc=?,dir=?/  [Kernel PMU event]
<...>

/ # perf stat -a -e ccn/cycles/,ccn/xp_valid_flit,xp=1,port=0,vc=1,dir=1/ \
↳sleep 1
```

The driver does not support sampling, therefore “perf record” will not work. Per-task (without “-a” ) perf sessions are not supported.

## 15.6 APM X-Gene SoC Performance Monitoring Unit (PMU)

X-Gene SoC PMU consists of various independent system device PMUs such as L3 cache(s), I/O bridge(s), memory controller bridge(s) and memory controller(s). These PMU devices are loosely architected to follow the same model as the PMU for ARM cores. The PMUs share the same top level interrupt and status CSR region.

### 15.6.1 PMU (perf) driver

The xgene-pmu driver registers several perf PMU drivers. Each of the perf driver provides description of its available events and configuration options in sysfs, see `/sys/devices/<l3cX/iobX/mcbX/mcX>/`.

The “format” directory describes format of the config (event ID), config1 (agent ID) fields of the `perf_event_attr` structure. The “events” directory provides configuration templates for all supported event types that can be used with perf tool. For example, “`l3c0/bank-fifo-full/`” is an equivalent of “`l3c0/config=0x0b/`” .

Most of the SoC PMU has a specific list of agent ID used for monitoring performance of a specific datapath. For example, agents of a L3 cache can be a specific CPU or an I/O bridge. Each PMU has a set of 2 registers capable of masking the agents from which the request come from. If the bit with the bit number corresponding to the agent is set, the event is counted only if it is caused by a request from that agent. Each agent ID bit is inversely mapped to a corresponding bit in “config1” field. By default, the event will be counted for all agent requests (config1 = 0x0). For all the supported agents of each PMU, please refer to APM X-Gene User Manual.

Each perf driver also provides a “cpumask” sysfs attribute, which contains a single CPU ID of the processor which will be used to handle all the PMU events.

Example for perf tool use:

```
/ # perf list | grep -e l3c -e iob -e mcb -e mc
  l3c0/ackq-full/                               [Kernel PMU event]
<...>
  mcb1/mcb-csw-stall/                           [Kernel PMU event]

/ # perf stat -a -e l3c0/read-miss/,mcb1/csw-write-request/ sleep 1

/ # perf stat -a -e l3c0/read-miss,config1=0xfffffffffffffffe/ sleep 1
```

The driver does not support sampling, therefore “perf record” will not work. Per-task (without “-a” ) perf sessions are not supported.

## 15.7 ARM DynamIQ Shared Unit (DSU) PMU

ARM DynamIQ Shared Unit integrates one or more cores with an L3 memory system, control logic and external interfaces to form a multicore cluster. The PMU allows counting the various events related to the L3 cache, Snoop Control Unit etc, using 32bit independent counters. It also provides a 64bit cycle counter.

The PMU can only be accessed via CPU system registers and are common to the cores connected to the same DSU. Like most of the other uncore PMUs, DSU PMU doesn't support process specific events and cannot be used in sampling mode.

The DSU provides a bitmap for a subset of implemented events via hardware registers. There is no way for the driver to determine if the other events are available or not. Hence the driver exposes only those events advertised by the DSU, in "events" directory under:

```
/sys/bus/event_sources/devices/arm_dsu_<N>/
```

The user should refer to the TRM of the product to figure out the supported events and use the raw event code for the unlisted events.

The driver also exposes the CPUs connected to the DSU instance in "associated\_cpus" .

e.g usage:

```
perf stat -a -e arm_dsu_0/cycles/
```

## 15.8 Cavium ThunderX2 SoC Performance Monitoring Unit (PMU UNCORE)

The ThunderX2 SoC PMU consists of independent, system-wide, per-socket PMUs such as the Level 3 Cache (L3C), DDR4 Memory Controller (DMC) and Cavium Coherent Processor Interconnect (CCPI2).

The DMC has 8 interleaved channels and the L3C has 16 interleaved tiles. Events are counted for the default channel (i.e. channel 0) and prorated to the total number of channels/tiles.

The DMC and L3C support up to 4 counters, while the CCPI2 supports up to 8 counters. Counters are independently programmable to different events and can be started and stopped individually. None of the counters support an overflow interrupt. DMC and L3C counters are 32-bit and read every 2 seconds. The CCPI2 counters are 64-bit and assumed not to overflow in normal operation.

PMU UNCORE (perf) driver:

The thunderx2\_pmu driver registers per-socket perf PMUs for the DMC and L3C devices. Each PMU can be used to count up to 4 (DMC/L3C) or up to 8 (CCPI2) events simultaneously. The PMUs provide a description of their available events and configuration options under sysfs, see `/sys/devices/uncore_<l3c_S/dmc_S/ccpi2_S/>`; S is the socket id.

The driver does not support sampling, therefore “perf record” will not work. Per-task perf sessions are also not supported.

Examples:

```
# perf stat -a -e uncore_dmc_0/cnt_cycles/ sleep 1

# perf stat -a -e \
uncore_dmc_0/cnt_cycles/,\
uncore_dmc_0/data_transfers/,\
uncore_dmc_0/read_txns/,\
uncore_dmc_0/write_txns/ sleep 1

# perf stat -a -e \
uncore_l3c_0/read_request/,\
uncore_l3c_0/read_hit/,\
uncore_l3c_0/inv_request/,\
uncore_l3c_0/inv_hit/ sleep 1
```

This is the beginning of a section with information of interest to application developers. Documents covering various aspects of the kernel ABI will be found here.

## **RULES ON HOW TO ACCESS INFORMATION IN SYSFS**

The kernel-exported `sysfs` exports internal kernel implementation details and depends on internal kernel structures and layout. It is agreed upon by the kernel developers that the Linux kernel does not provide a stable internal API. Therefore, there are aspects of the `sysfs` interface that may not be stable across kernel releases.

To minimize the risk of breaking users of `sysfs`, which are in most cases low-level userspace applications, with a new kernel release, the users of `sysfs` must follow some rules to use an as-abstract-as-possible way to access this filesystem. The current `udev` and `HAL` programs already implement this and users are encouraged to plug, if possible, into the abstractions these programs provide instead of accessing `sysfs` directly.

But if you really do want or need to access `sysfs` directly, please follow the following rules and then your programs should work with future versions of the `sysfs` interface.

- **Do not use `libsfs`** It makes assumptions about `sysfs` which are not true. Its API does not offer any abstraction, it exposes all the kernel driver-core implementation details in its own API. Therefore it is not better than reading directories and opening the files yourself. Also, it is not actively maintained, in the sense of reflecting the current kernel development. The goal of providing a stable interface to `sysfs` has failed; it causes more problems than it solves. It violates many of the rules in this document.
- **`sysfs` is always at `/sys`** Parsing `/proc/mounts` is a waste of time. Other mount points are a system configuration bug you should not try to solve. For test cases, possibly support a `SYSFS_PATH` environment variable to overwrite the application's behavior, but never try to search for `sysfs`. Never try to mount it, if you are not an early boot script.
- **devices are only “devices”** There is no such thing like class-, bus-, physical devices, interfaces, and such that you can rely on in userspace. Everything is just simply a “device”. Class-, bus-, physical, ...types are just kernel implementation details which should not be expected by applications that look for devices in `sysfs`.

The properties of a device are:

- `devpath` (`/devices/pci0000:00/0000:00:1d.1/usb2/2-2/2-2:1.0`)
  - \* identical to the `DEVPATH` value in the event sent from the kernel at device creation and removal

- \* the unique key to the device at that point in time
- \* the kernel's path to the device directory without the leading /sys, and always starting with a slash
- \* all elements of a devpath must be real directories. Symlinks pointing to /sys/devices must always be resolved to their real target and the target path must be used to access the device. That way the devpath to the device matches the devpath of the kernel used at event time.
- \* using or exposing symlink values as elements in a devpath string is a bug in the application
- kernel name (sda, tty, 0000:00:1f.2, ...)
  - \* a directory name, identical to the last element of the devpath
  - \* applications need to handle spaces and characters like ! in the name
- subsystem (block, tty, pci, ...)
  - \* simple string, never a path or a link
  - \* retrieved by reading the "subsystem" -link and using only the last element of the target path
- driver (tg3, ata\_piix, uhci\_hcd)
  - \* a simple string, which may contain spaces, never a path or a link
  - \* it is retrieved by reading the "driver" -link and using only the last element of the target path
  - \* devices which do not have "driver" -link just do not have a driver; copying the driver value in a child device context is a bug in the application
- attributes
  - \* the files in the device directory or files below subdirectories of the same device directory
  - \* accessing attributes reached by a symlink pointing to another device, like the "device" -link, is a bug in the application

Everything else is just a kernel driver-core implementation detail that should not be assumed to be stable across kernel releases.

- **Properties of parent devices never belong into a child device.** Always look at the parent devices themselves for determining device context properties. If the device eth0 or sda does not have a "driver" -link, then this device does not have a driver. Its value is empty. Never copy any property of the parent-device into a child-device. Parent device properties may change dynamically without any notice to the child device.
- **Hierarchy in a single device tree** There is only one valid place in sysfs where hierarchy can be examined and this is below: /sys/devices. It

is planned that all device directories will end up in the tree below this directory.

- **Classification by subsystem** There are currently three places for classification of devices: `/sys/block`, `/sys/class` and `/sys/bus`. It is planned that these will not contain any device directories themselves, but only flat lists of symlinks pointing to the unified `/sys/devices` tree. All three places have completely different rules on how to access device information. It is planned to merge all three classification directories into one place at `/sys/subsystem`, following the layout of the bus directories. All buses and classes, including the converted block subsystem, will show up there. The devices belonging to a subsystem will create a symlink in the “devices” directory at `/sys/subsystem/<name>/devices`,

If `/sys/subsystem` exists, `/sys/bus`, `/sys/class` and `/sys/block` can be ignored. If it does not exist, you always have to scan all three places, as the kernel is free to move a subsystem from one place to the other, as long as the devices are still reachable by the same subsystem name.

Assuming `/sys/class/<subsystem>` and `/sys/bus/<subsystem>`, or `/sys/block` and `/sys/class/block` are not interchangeable is a bug in the application.

- **Block** The converted block subsystem at `/sys/class/block` or `/sys/subsystem/block` will contain the links for disks and partitions at the same level, never in a hierarchy. Assuming the block subsystem to contain only disks and not partition devices in the same flat list is a bug in the application.
- **“device” -link and <subsystem>:<kernel name>-links** Never depend on the “device” -link. The “device” -link is a workaround for the old layout, where class devices are not created in `/sys/devices/` like the bus devices. If the link-resolving of a device directory does not end in `/sys/devices/`, you can use the “device” -link to find the parent devices in `/sys/devices/`. That is the single valid use of the “device” -link; it must never appear in any path as an element. Assuming the existence of the “device” -link for a device in `/sys/devices/` is a bug in the application. Accessing `/sys/class/net/eth0/device` is a bug in the application.

Never depend on the class-specific links back to the `/sys/class` directory. These links are also a workaround for the design mistake that class devices are not created in `/sys/devices`. If a device directory does not contain directories for child devices, these links may be used to find the child devices in `/sys/class`. That is the single valid use of these links; they must never appear in any path as an element. Assuming the existence of these links for devices which are real child device directories in the `/sys/devices` tree is a bug in the application.

It is planned to remove all these links when all class device directories live in `/sys/devices`.

- **Position of devices along device chain can change.** Never depend on a specific parent device position in the devpath, or the chain of parent devices. The kernel is free to insert devices into the chain. You must always request the parent device you are looking for by its subsystem value. You

need to walk up the chain until you find the device that matches the expected subsystem. Depending on a specific position of a parent device or exposing relative paths using `../` to access the chain of parents is a bug in the application.

- **When reading and writing sysfs device attribute files, avoid dependency** on specific error codes wherever possible. This minimizes coupling to the error handling implementation within the kernel.

In general, failures to read or write sysfs device attributes shall propagate errors wherever possible. Common errors include, but are not limited to:

- EIO: The read or store operation is not supported, typically returned by the sysfs system itself if the read or store pointer is NULL.

- ENXIO: The read or store operation failed

Error codes will not be changed without good reason, and should a change to error codes result in user-space breakage, it will be fixed, or the the offending change will be reverted.

Userspace applications can, however, expect the format and contents of the attribute files to remain consistent in the absence of a version attribute change in the context of a given attribute.

The rest of this manual consists of various unordered guides on how to configure specific aspects of kernel behavior to your liking.

## **ACPI SUPPORT**

Here we document in detail how to interact with various mechanisms in the Linux ACPI support.

### **17.1 Upgrading ACPI tables via initrd**

#### **17.1.1 What is this about**

If the `ACPI_TABLE_UPGRADE` compile option is true, it is possible to upgrade the ACPI execution environment that is defined by the ACPI tables via upgrading the ACPI tables provided by the BIOS with an instrumented, modified, more recent version one, or installing brand new ACPI tables.

When building `initrd` with kernel in a single image, option `ACPI_TABLE_OVERRIDE_VIA_BUILTIN_INITRD` should also be true for this feature to work.

For a full list of ACPI tables that can be upgraded/installed, take a look at the char `*table_sigs[MAX_ACPI_SIGNATURE]`; definition in `drivers/acpi/tables.c`.

All ACPI tables `iasl` (Intel' s ACPI compiler and disassembler) knows should be overridable, except:

- `ACPI_SIG_RSDP` (has a signature of 6 bytes)
- `ACPI_SIG_FACS` (does not have an ordinary ACPI table header)

Both could get implemented as well.

#### **17.1.2 What is this for**

Complain to your platform/BIOS vendor if you find a bug which is so severe that a workaround is not accepted in the Linux kernel. And this facility allows you to upgrade the buggy tables before your platform/BIOS vendor releases an upgraded BIOS binary.

This facility can be used by platform/BIOS vendors to provide a Linux compatible environment without modifying the underlying platform firmware.

This facility also provides a powerful feature to easily debug and test ACPI BIOS table compatibility with the Linux kernel by modifying old platform provided ACPI tables or inserting new ACPI tables.

It can and should be enabled in any kernel because there is no functional change with not instrumented initrds.

### 17.1.3 How does it work

```
# Extract the machine's ACPI tables:
cd /tmp
acpidump >acpidump
acpixtract -a acpidump
# Disassemble, modify and recompile them:
iasl -d *.dat
# For example add this statement into a _PRT (PCI Routing Table) function
# of the DSDT:
Store("HELLO WORLD", debug)
# And increase the OEM Revision. For example, before modification:
DefinitionBlock ("DSDT.aml", "DSDT", 2, "INTEL ", "TEMPLATE", 0x00000000)
# After modification:
DefinitionBlock ("DSDT.aml", "DSDT", 2, "INTEL ", "TEMPLATE", 0x00000001)
iasl -sa dsdt.dsl
# Add the raw ACPI tables to an uncompressed cpio archive.
# They must be put into a /kernel/firmware/acpi directory inside the cpio
# archive. Note that if the table put here matches a platform table
# (similar Table Signature, and similar OEMID, and similar OEM Table ID)
# with a more recent OEM Revision, the platform table will be upgraded by
# this table. If the table put here doesn't match a platform table
# (dissimilar Table Signature, or dissimilar OEMID, or dissimilar OEM Table
# ID), this table will be appended.
mkdir -p kernel/firmware/acpi
cp dsdt.aml kernel/firmware/acpi
# A maximum of "NR_ACPI_INITRD_TABLES (64)" tables are currently allowed
# (see osl.c):
iasl -sa facp.dsl
iasl -sa ssdt1.dsl
cp facp.aml kernel/firmware/acpi
cp ssdt1.aml kernel/firmware/acpi
# The uncompressed cpio archive must be the first. Other, typically
# compressed cpio archives, must be concatenated on top of the uncompressed
# one. Following command creates the uncompressed cpio archive and
# concatenates the original initrd on top:
find kernel | cpio -H newc --create > /boot/instrumented_initrd
cat /boot/initrd >>/boot/instrumented_initrd
# reboot with increased acpi debug level, e.g. boot params:
acpi.debug_level=0x2 acpi.debug_layer=0xFFFFFFFF
# and check your syslog:
[ 1.268089] ACPI: PCI Interrupt Routing Table [_SB_.PCI0._PRT]
[ 1.272091] [ACPI Debug] String [0x0B] "HELLO WORLD"
```

iasl is able to disassemble and recompile quite a lot different, also static ACPI tables.

### 17.1.4 Where to retrieve userspace tools

iasl and acpextract are part of Intel' s ACPICA project: <https://acpica.org/> and should be packaged by distributions (for example in the acpica package on SUSE).

acpidump can be found in Len Browns pmtools: <ftp://kernel.org/pub/linux/kernel/people/lenb/acpi/utills/pmtools/acpidump>

This tool is also part of the acpica package on SUSE. Alternatively, used ACPI tables can be retrieved via sysfs in latest kernels: `/sys/firmware/acpi/tables`

## 17.2 Overriding DSDT

Linux supports a method of overriding the BIOS DSDT:

`CONFIG_ACPI_CUSTOM_DSDT` - builds the image into the kernel.

When to use this method is described in detail on the Linux/ACPI home page: <https://01.org/linux-acpi/documentation/overriding-dsdt>

## 17.3 SSDT Overlays

In order to support ACPI open-ended hardware configurations (e.g. development boards) we need a way to augment the ACPI configuration provided by the firmware image. A common example is connecting sensors on I2C / SPI buses on development boards.

Although this can be accomplished by creating a kernel platform driver or re-compiling the firmware image with updated ACPI tables, neither is practical: the former proliferates board specific kernel code while the latter requires access to firmware tools which are often not publicly available.

Because ACPI supports external references in AML code a more practical way to augment firmware ACPI configuration is by dynamically loading user defined SSDT tables that contain the board specific information.

For example, to enumerate a Bosch BMA222E accelerometer on the I2C bus of the Minnowboard MAX development board exposed via the LSE connector [1], the following ASL code can be used:

```
DefinitionBlock ("minnowmax.aml", "SSDT", 1, "Vendor", "Accel", 0x00000003)
{
    External (\_SB.I2C6, DeviceObj)

    Scope (\_SB.I2C6)
    {
        Device (STAC)
        {
            Name (_ADR, Zero)
            Name (_HID, "BMA222E")
        }
    }
}
```

(continues on next page)

(continued from previous page)

```

Method (_CRS, 0, Serialized)
{
    Name (RBUF, ResourceTemplate ()
    {
        I2cSerialBus (0x0018, ControllerInitiated, 0x00061A80,
            AddressingMode7Bit, "\\_SB.I2C6", 0x00,
            ResourceConsumer, ,)
        GpioInt (Edge, ActiveHigh, Exclusive, PullDown, 0x0000,
            "\\_SB.GP02", 0x00, ResourceConsumer, , )
        { // Pin list
            0
        }
    })
    Return (RBUF)
}
}
}
}
}

```

which can then be compiled to AML binary format:

```

$ iasl minnowmax.asl

Intel ACPI Component Architecture
ASL Optimizing Compiler version 20140214-64 [Mar 29 2014]
Copyright (c) 2000 - 2014 Intel Corporation

ASL Input:      minnowmax.asl - 30 lines, 614 bytes, 7 keywords
AML Output:     minnowmax.aml - 165 bytes, 6 named objects, 1 executable,
↳ opcodes

```

[1] [https://www.elinux.org/Minnowboard:MinnowMax#Low\\_Speed\\_Expansion\\_.28Top.29](https://www.elinux.org/Minnowboard:MinnowMax#Low_Speed_Expansion_.28Top.29)

The resulting AML code can then be loaded by the kernel using one of the methods below.

### 17.3.1 Loading ACPI SSDTs from initrd

This option allows loading of user defined SSDTs from initrd and it is useful when the system does not support EFI or when there is not enough EFI storage.

It works in a similar way with initrd based ACPI tables override/upgrade: SSDT aml code must be placed in the first, uncompressed, initrd under the “kernel/firmware/acpi” path. Multiple files can be used and this will translate in loading multiple tables. Only SSDT and OEM tables are allowed. See `initrd_table_override.txt` for more details.

Here is an example:

```

# Add the raw ACPI tables to an uncompressed cpio archive.
# They must be put into a /kernel/firmware/acpi directory inside the
# cpio archive.
# The uncompressed cpio archive must be the first.

```

(continues on next page)

(continued from previous page)

```
# Other, typically compressed cpio archives, must be
# concatenated on top of the uncompressed one.
mkdir -p kernel/firmware/acpi
cp ssdt.aml kernel/firmware/acpi

# Create the uncompressed cpio archive and concatenate the original initrd
# on top:
find kernel | cpio -H newc --create > /boot/instrumented_initrd
cat /boot/initrd >>/boot/instrumented_initrd
```

### 17.3.2 Loading ACPI SSDTs from EFI variables

This is the preferred method, when EFI is supported on the platform, because it allows a persistent, OS independent way of storing the user defined SSDTs. There is also work underway to implement EFI support for loading user defined SSDTs and using this method will make it easier to convert to the EFI loading mechanism when that will arrive.

In order to load SSDTs from an EFI variable the `efivar_ssdt` kernel command line parameter can be used. The argument for the option is the variable name to use. If there are multiple variables with the same name but with different vendor GUIDs, all of them will be loaded.

In order to store the AML code in an EFI variable the `efivarfs` filesystem can be used. It is enabled and mounted by default in `/sys/firmware/efi/efivars` in all recent distribution.

Creating a new file in `/sys/firmware/efi/efivars` will automatically create a new EFI variable. Updating a file in `/sys/firmware/efi/efivars` will update the EFI variable. Please note that the file name needs to be specially formatted as “Name-GUID” and that the first 4 bytes in the file (little-endian format) represent the attributes of the EFI variable (see `EFI_VARIABLE_MASK` in `include/linux/efi.h`). Writing to the file must also be done with one write operation.

For example, you can use the following bash script to create/update an EFI variable with the content from a given file:

```
#!/bin/sh -e

while ! [ -z "$1" ]; do
    case "$1" in
        "-f") filename="$2"; shift;;
        "-g") guid="$2"; shift;;
        *) name="$1";;
    esac
    shift
done

usage()
{
    echo "Syntax: ${0##*/} -f filename [ -g guid ] name"
    exit 1
}
```

(continues on next page)

(continued from previous page)

```
[ -n "$name" -a -f "$filename" ] || usage
EFIVARFS="/sys/firmware/efi/efivars"

[ -d "$EFIVARFS" ] || exit 2

if stat -tf $EFIVARFS | grep -q -v de5e81e4; then
    mount -t efivarfs none $EFIVARFS
fi

# try to pick up an existing GUID
[ -n "$guid" ] || guid=$(find "$EFIVARFS" -name "$name-*" | head -n1 | cut_
↪-f2- -d-)

# use a randomly generated GUID
[ -n "$guid" ] || guid="$(cat /proc/sys/kernel/random/uuid)"

# efivarfs expects all of the data in one write
tmp=$(mktemp)
/bin/echo -ne "\007\000\000\000" | cat - $filename > $tmp
dd if=$tmp of="$EFIVARFS/$name-$guid" bs=$(stat -c %s $tmp)
rm $tmp
```

### 17.3.3 Loading ACPI SSDTs from configs

This option allows loading of user defined SSDTs from userspace via the configs interface. The CONFIG\_ACPI\_CONFIGFS option must be select and configs must be mounted. In the following examples, we assume that configs has been mounted in /config.

New tables can be loading by creating new directories in /config/acpi/table/ and writing the SSDT aml code in the aml attribute:

```
cd /config/acpi/table
mkdir my_ssdt
cat ~/ssdt.aml > my_ssdt/aml
```

## 17.4 Collaborative Processor Performance Control (CPPC)

### 17.4.1 CPPC

CPPC defined in the ACPI spec describes a mechanism for the OS to manage the performance of a logical processor on a contiguous and abstract performance scale. CPPC exposes a set of registers to describe abstract performance scale, to request performance levels and to measure per-cpu delivered performance.

For more details on CPPC please refer to the ACPI specification at:

<http://uefi.org/specifications>

Some of the CPPC registers are exposed via sysfs under:

```
/sys/devices/system/cpu/cpuX/acpi_cppc/
```

for each cpu X:

```
$ ls -lR /sys/devices/system/cpu/cpu0/acpi_cppc/
/sys/devices/system/cpu/cpu0/acpi_cppc/:
total 0
-r--r--r-- 1 root root 65536 Mar  5 19:38 feedback_ctrs
-r--r--r-- 1 root root 65536 Mar  5 19:38 highest_perf
-r--r--r-- 1 root root 65536 Mar  5 19:38 lowest_freq
-r--r--r-- 1 root root 65536 Mar  5 19:38 lowest_nonlinear_perf
-r--r--r-- 1 root root 65536 Mar  5 19:38 lowest_perf
-r--r--r-- 1 root root 65536 Mar  5 19:38 nominal_freq
-r--r--r-- 1 root root 65536 Mar  5 19:38 nominal_perf
-r--r--r-- 1 root root 65536 Mar  5 19:38 reference_perf
-r--r--r-- 1 root root 65536 Mar  5 19:38 wraparound_time
```

- `highest_perf` : Highest performance of this processor (abstract scale).
- `nominal_perf` : Highest sustained performance of this processor (abstract scale).
- `lowest_nonlinear_perf` : Lowest performance of this processor with nonlinear power savings (abstract scale).
- `lowest_perf` : Lowest performance of this processor (abstract scale).
- `lowest_freq` : CPU frequency corresponding to `lowest_perf` (in MHz).
- `nominal_freq` : CPU frequency corresponding to `nominal_perf` (in MHz). The above frequencies should only be used to report processor performance in frequency instead of abstract scale. These values should not be used for any functional decisions.
- `feedback_ctrs` : Includes both Reference and delivered performance counter. Reference counter ticks up proportional to processor' s reference performance. Delivered counter ticks up proportional to processor' s delivered performance.
- `wraparound_time`: Minimum time for the feedback counters to wraparound (seconds).
- `reference_perf` : Performance level at which reference performance counter accumulates (abstract scale).

### 17.4.2 Computing Average Delivered Performance

Below describes the steps to compute the average performance delivered by taking two different snapshots of feedback counters at time T1 and T2.

**T1: Read `feedback_ctrs` as `fbc_t1`** Wait or run some workload

T2: Read `feedback_ctrs` as `fbc_t2`

```
delivered_counter_delta = fbc_t2[del] - fbc_t1[del]
reference_counter_delta = fbc_t2[ref] - fbc_t1[ref]

delivered_perf = (reference_perf x delivered_counter_delta) / reference_
→counter_delta
```

## 17.5 ACPI Fan Performance States

When the optional `_FPS` object is present under an ACPI device representing a fan (for example, `PNP0C0B` or `INT3404`), the ACPI fan driver creates additional “state\*” attributes in the `sysfs` directory of the ACPI device in question. These attributes list properties of fan performance states.

For more information on `_FPS` refer to the ACPI specification at:

<http://uefi.org/specifications>

For instance, the contents of the `INT3404` ACPI device `sysfs` directory may look as follows:

```
$ ls -l /sys/bus/acpi/devices/INT3404:00/
total 0
...
-r--r--r-- 1 root root 4096 Dec 13 20:38 state0
-r--r--r-- 1 root root 4096 Dec 13 20:38 state1
-r--r--r-- 1 root root 4096 Dec 13 20:38 state10
-r--r--r-- 1 root root 4096 Dec 13 20:38 state11
-r--r--r-- 1 root root 4096 Dec 13 20:38 state2
-r--r--r-- 1 root root 4096 Dec 13 20:38 state3
-r--r--r-- 1 root root 4096 Dec 13 20:38 state4
-r--r--r-- 1 root root 4096 Dec 13 20:38 state5
-r--r--r-- 1 root root 4096 Dec 13 20:38 state6
-r--r--r-- 1 root root 4096 Dec 13 20:38 state7
-r--r--r-- 1 root root 4096 Dec 13 20:38 state8
-r--r--r-- 1 root root 4096 Dec 13 20:38 state9
-r--r--r-- 1 root root 4096 Dec 13 01:00 status
...
```

where each of the “state\*” files represents one performance state of the fan and contains a colon-separated list of 5 integer numbers (fields) with the following interpretation:

```
control_percent:trip_point_index:speed_rpm:noise_level_mdb:power_mw
```

- `control_percent`: The percent value to be used to set the fan speed to a specific level using the `_FSL` object (0-100).
- `trip_point_index`: The active cooling trip point number that corresponds to this performance state (0-9).
- `speed_rpm`: Speed of the fan in rotations per minute.
- `noise_level_mdb`: Audible noise emitted by the fan in this state in millidecibels.
- `power_mw`: Power draw of the fan in this state in milliwatts.

For example:

```
$cat /sys/bus/acpi/devices/INT3404:00/state1  
25:0:3200:12500:1250
```

When a given field is not populated or its value provided by the platform firmware is invalid, the “not-defined” string is shown instead of the value.



## **ATA OVER ETHERNET (AOE)**

### **18.1 Introduction**

ATA over Ethernet is a network protocol that provides simple access to block storage on the LAN.

<http://support.coraid.com/documents/AoEr11.txt>

The EtherDrive (R) HOWTO for 2.6 and 3.x kernels is found at ...

<http://support.coraid.com/support/linux/EtherDrive-2.6-HOWTO.html>

It has many tips and hints! Please see, especially, recommended tunings for virtual memory:

<http://support.coraid.com/support/linux/EtherDrive-2.6-HOWTO-5.html#ss5.19>

The aoetools are userland programs that are designed to work with this driver. The aoetools are on sourceforge.

<http://aoetools.sourceforge.net/>

The scripts in this Documentation/admin-guide/aoe directory are intended to document the use of the driver and are not necessary if you install the aoetools.

### **18.2 Creating Device Nodes**

Users of udev should find the block device nodes created automatically, but to create all the necessary device nodes, use the udev configuration rules provided in udev.txt (in this directory).

There is a udev-install.sh script that shows how to install these rules on your system.

There is also an autoload script that shows how to edit /etc/modprobe.d/aoe.conf to ensure that the aoe module is loaded when necessary. Preloading the aoe module is preferable to autoloading, however, because AoE discovery takes a few seconds. It can be confusing when an AoE device is not present the first time the a command is run but appears a second later.

## 18.3 Using Device Nodes

“cat /dev/etherd/err” blocks, waiting for error diagnostic output, like any retransmitted packets.

“echo eth2 eth4 > /dev/etherd/interfaces” tells the aoe driver to limit ATA over Ethernet traffic to eth2 and eth4. AoE traffic from untrusted networks should be ignored as a matter of security. See also the aoe\_iflist driver option described below.

“echo > /dev/etherd/discover” tells the driver to find out what AoE devices are available.

In the future these character devices may disappear and be replaced by sysfs counterparts. Using the commands in aotools insulates users from these implementation details.

The block devices are named like this:

```
e{shelf}.{slot}
e{shelf}.{slot}p{part}
```

...so that “e0.2” is the third blade from the left (slot 2) in the first shelf (shelf address zero). That’s the whole disk. The first partition on that disk would be “e0.2p1” .

## 18.4 Using sysfs

Each aoe block device in /sys/block has the extra attributes of state, mac, and netif. The state attribute is “up” when the device is ready for I/O and “down” if detected but unusable. The “down,closewait” state shows that the device is still open and cannot come up again until it has been closed.

The mac attribute is the ethernet address of the remote AoE device. The netif attribute is the network interface on the localhost through which we are communicating with the remote AoE device.

There is a script in this directory that formats this information in a convenient way. Users with aotools should use the aoe-stat command:

```
root@makki root# sh Documentation/admin-guide/aoe/status.sh
e10.0          eth3          up
e10.1          eth3          up
e10.2          eth3          up
e10.3          eth3          up
e10.4          eth3          up
e10.5          eth3          up
e10.6          eth3          up
e10.7          eth3          up
e10.8          eth3          up
e10.9          eth3          up
e4.0           eth1          up
e4.1           eth1          up
```

(continues on next page)

(continued from previous page)

e4.2	eth1	up
e4.3	eth1	up
e4.4	eth1	up
e4.5	eth1	up
e4.6	eth1	up
e4.7	eth1	up
e4.8	eth1	up
e4.9	eth1	up

Use `/sys/module/aoe/parameters/aoe_iflist` (or better, the driver option discussed below) instead of `/dev/etherd/interfaces` to limit AoE traffic to the network interfaces in the given whitespace-separated list. Unlike the old character device, the `sysfs` entry can be read from as well as written to.

It's helpful to trigger discovery after setting the list of allowed interfaces. The `aoetools` package provides an `aoe-discover` script for this purpose. You can also directly use the `/dev/etherd/discover` special file described above.

## 18.5 Driver Options

There is a boot option for the built-in `aoe` driver and a corresponding module parameter, `aoe_iflist`. Without this option, all network interfaces may be used for ATA over Ethernet. Here is a usage example for the module parameter:

```
modprobe aoe_iflist="eth1 eth3"
```

The `aoe_deadsecs` module parameter determines the maximum number of seconds that the driver will wait for an AoE device to provide a response to an AoE command. After `aoe_deadsecs` seconds have elapsed, the AoE device will be marked as “down”. A value of zero is supported for testing purposes and makes the `aoe` driver keep trying AoE commands forever.

The `aoe_maxout` module parameter has a default of 128. This is the maximum number of unresponded packets that will be sent to an AoE target at one time.

The `aoe_dyndevs` module parameter defaults to 1, meaning that the driver will assign a block device minor number to a discovered AoE target based on the order of its discovery. With dynamic minor device numbers in use, a greater range of AoE shelf and slot addresses can be supported. Users with `udev` will never have to think about minor numbers. Using `aoe_dyndevs=0` allows device nodes to be pre-created using a static minor-number scheme with the `aoe-mkshelf` script in the `aoetools`.

## 18.6 TODO

There is a potential for deadlock when allocating a struct `sk_buff` for data that needs to be written out to aoe storage. If the data is being written from a dirty page in order to free that page, and if there are no other pages available, then deadlock may occur when a free page is needed for the `sk_buff` allocation. This situation has not been observed, but it would be nice to eliminate any potential for deadlock under memory pressure.

Because ATA over Ethernet is not fragmented by the kernel's IP code, the destructor member of the struct `sk_buff` is available to the aoe driver. By using a mempool for allocating all but the first few `sk_buffs`, and by registering a destructor, we should be able to efficiently allocate `sk_buffs` without introducing any potential for deadlock.

## 18.7 Example of udev rules

```
# These rules tell udev what device nodes to create for aoe_
↳support.
# They may be installed along the following lines. Check the_
↳section
# 8 udev manpage to see whether your udev supports SUBSYSTEM,_
↳and
# whether it uses one or two equal signs for SUBSYSTEM and_
↳KERNEL.
#
# ecashin@makki ~$ su
# Password:
# bash# find /etc -type f -name udev.conf
# /etc/udev/udev.conf
# bash# grep udev_rules= /etc/udev/udev.conf
# udev_rules="/etc/udev/rules.d/"
# bash# ls /etc/udev/rules.d/
# 10-wacom.rules 50-udev.rules
# bash# cp /path/to/linux/Documentation/admin-guide/aoe/udev.
↳txt \
#           /etc/udev/rules.d/60-aoe.rules
#
# aoe char devices
SUBSYSTEM=="aoe", KERNEL=="discover", NAME="etherd/%k",_
↳GROUP="disk", MODE="0220"
SUBSYSTEM=="aoe", KERNEL=="err", NAME="etherd/%k",_
↳GROUP="disk", MODE="0440"
SUBSYSTEM=="aoe", KERNEL=="interfaces", NAME="etherd/%k",_
↳GROUP="disk", MODE="0220"
SUBSYSTEM=="aoe", KERNEL=="revalidate", NAME="etherd/%k",_
↳GROUP="disk", MODE="0220"
SUBSYSTEM=="aoe", KERNEL=="flush", NAME="etherd/%k",_
↳GROUP="disk", MODE="0220"
```

```
# aoe block devices
KERNEL=="etherd*",      GROUP="disk"
```

## 18.8 Example of udev install rules script

```
# install the aoe-specific udev rules from udev.txt into
# the system's udev configuration
#
me="`basename $0`"

# find udev.conf, often /etc/udev/udev.conf
# (or environment can specify where to find udev.conf)
#
if test -z "$conf"; then
    if test -r /etc/udev/udev.conf; then
        conf=/etc/udev/udev.conf
    else
        conf="`find /etc -type f -name udev.conf 2> /dev/
↪null`"
        if test -z "$conf" || test ! -r "$conf"; then
            echo "$me Error: no udev.conf found" 1>&2
            exit 1
        fi
    fi
fi

# find the directory where udev rules are stored, often
# /etc/udev/rules.d
#
rules_d="`sed -n '/^udev_rules={ s!udev_rules=!!; s!\\"!!g; p; }'
↪$conf`"
if test -z "$rules_d" ; then
    rules_d=/etc/udev/rules.d
fi
if test ! -d "$rules_d"; then
    echo "$me Error: cannot find udev rules directory" 1>&2
    exit 1
fi
sh -xc "cp `dirname $0`/udev.txt $rules_d/60-aoe.rules"
```

## 18.9 Example script to get status

```
#!/bin/sh
# collate and present sysfs information about AoE storage
#
# A more complete version of this script is aoe-stat, in the
# aoetools.

set -e
```

(continues on next page)

(continued from previous page)

```
format="%8s\t%8s\t%8s\n"
me=`basename $0`
sysd=${sysfs_dir:-/sys}

# printf "$format" device mac netif state

# Suse 9.1 Pro doesn't put /sys in /etc/mtab
#test -z "`mount | grep sysfs`" && {
test ! -d "$sysd/block" && {
    echo "$me Error: sysfs is not mounted" 1>&2
    exit 1
}

for d in `ls -d $sysd/block/etherd* 2>/dev/null | grep -v p` end;
do
    # maybe ls comes up empty, so we use "end"
    test $d = end && continue

    dev=`echo "$d" | sed 's/.*!//`
    printf "$format" \
        "$dev" \
        "`cat \"$d/netif\"`" \
        "`cat \"$d/state\"`"
done | sort
```

## 18.10 Example of AoE autoloader script

```
#!/bin/sh
# set aoe to autoloader by installing the
# aliases in /etc/modprobe.d/

f=/etc/modprobe.d/aoe.conf

if test ! -r $f || test ! -w $f; then
    echo "cannot configure $f for module autoloading" 1>&2
    exit 1
fi

grep major-152 $f >/dev/null
if [ $? = 1 ]; then
    echo alias block-major-152 aoe >> $f
    echo alias char-major-152 aoe >> $f
fi
```

## **AUXILIARY DISPLAY SUPPORT**

### **19.1 ks0108 LCD Controller Driver Documentation**

**License** GPLv2

**Author & Maintainer** Miguel Ojeda Sandonis

**Date** 2006-10-27

#### **19.1.1 1. Driver Information**

This driver supports the ks0108 LCD controller.

#### **19.1.2 2. Device Information**

**Manufacturer** Samsung

**Device Name** KS0108 LCD Controller

**Device Code** ks0108

**Webpage**

•

**Device Webpage**

•

**Type** LCD Controller (Liquid Crystal Display Controller)

**Width** 64

**Height** 64

**Colors** 2 (B/N)

**Pages** 8

**Addresses** 64 each page

**Data size** 1 byte each address

**Memory size** 8 \* 64 \* 1 = 512 bytes

### 19.1.3 3. Wiring

The driver supports data parallel port wiring.

If you aren't building LCD related hardware, you should check your LCD specific wiring information in the same folder.

For example, check Documentation/admin-guide/auxdisplay/cfag12864b.rst

## 19.2 cfag12864b LCD Driver Documentation

**License** GPLv2

**Author & Maintainer** Miguel Ojeda Sandonis

**Date** 2006-10-27

### 19.2.1 1. Driver Information

This driver supports a cfag12864b LCD.

### 19.2.2 2. Device Information

**Manufacturer** Crystalfontz

**Device Name** Crystalfontz 12864b LCD Series

**Device Code** cfag12864b

**Webpage** <http://www.crystalfontz.com>

**Device Webpage** <http://www.crystalfontz.com/products/12864b/>

**Type** LCD (Liquid Crystal Display)

**Width** 128

**Height** 64

**Colors** 2 (B/N)

**Controller** ks0108

**Controllers** 2

**Pages** 8 each controller

**Addresses** 64 each page

**Data size** 1 byte each address

**Memory size**  $2 * 8 * 64 * 1 = 1024$  bytes = 1 Kbyte

### 19.2.3 3. Wiring

The cfag12864b LCD Series don' t have official wiring.

The common wiring is done to the parallel port as shown:

Parallel Port		cfag12864b	
Name	Pin#	Pin#	Name
Strobe	( 1)	(17)	Enable
Data 0	( 2)	( 4)	Data 0
Data 1	( 3)	( 5)	Data 1
Data 2	( 4)	( 6)	Data 2
Data 3	( 5)	( 7)	Data 3
Data 4	( 6)	( 8)	Data 4
Data 5	( 7)	( 9)	Data 5
Data 6	( 8)	(10)	Data 6
Data 7	( 9)	(11)	Data 7
	(10)	[+5v]	( 1) Vdd
	(11)	[GND]	( 2) Ground
	(12)	[+5v]	(14) Reset
	(13)	[GND]	(15) Read / Write
Line	(14)	(13)	Controller Select 1
	(15)		
Init	(16)	(12)	Controller Select 2
Select	(17)	(16)	Data / Instruction
Ground	(18) --- [GND]	[+5v]	(19) LED +
Ground	(19) --- [GND]		
Ground	(20) --- [GND]	E A	Values:
Ground	(21) --- [GND]	[GND] --- [P1] ---	(18) Vee - R = Resistor = 22 <sub>Ω</sub>
→ohm			
Ground	(22) --- [GND]		- P1 = Preset = 10 <sub>Ω</sub>
→Kohm			
Ground	(23) --- [GND]	---- S -----	( 3) V0 - P2 = Preset = 1 Kohm
Ground	(24) --- [GND]		
Ground	(25) --- [GND]	[GND] --- [P2] --- [R] ---	(20) LED -

### 19.2.4 4. Userspace Programming

The cfag12864bfb describes a framebuffer device (/dev/fbX).

It has a size of 1024 bytes = 1 Kbyte. Each bit represents one pixel. If the bit is high, the pixel will turn on. If the pixel is low, the pixel will turn off.

You can use the framebuffer as a file: fopen, fwrite, fclose...Although the LCD won' t get updated until the next refresh time arrives.

Also, you can mmap the framebuffer: open & mmap, munmap & close...which is the best option for most uses.

Check samples/auxdisplay/cfag12864b-example.c for a real working userspace complete program with usage examples.



## A BLOCK LAYER CACHE (BCACHE)

Say you' ve got a big slow raid 6, and an ssd or three. Wouldn' t it be nice if you could use them as cache...Hence bcache.

Wiki and git repositories are at:

- <https://bcache.evilpiepirate.org>
- <http://evilpiepirate.org/git/linux-bcache.git>
- <https://evilpiepirate.org/git/bcache-tools.git>

It' s designed around the performance characteristics of SSDs - it only allocates in erase block sized buckets, and it uses a hybrid btree/log to track cached extents (which can be anywhere from a single sector to the bucket size). It' s designed to avoid random writes at all costs; it fills up an erase block sequentially, then issues a discard before reusing it.

Both writethrough and writeback caching are supported. Writeback defaults to off, but can be switched on and off arbitrarily at runtime. Bcache goes to great lengths to protect your data - it reliably handles unclean shutdown. (It doesn' t even have a notion of a clean shutdown; bcache simply doesn' t return writes as completed until they' re on stable storage).

Writeback caching can use most of the cache for buffering writes - writing dirty data to the backing device is always done sequentially, scanning from the start to the end of the index.

Since random IO is what SSDs excel at, there generally won' t be much benefit to caching large sequential IO. Bcache detects sequential IO and skips it; it also keeps a rolling average of the IO sizes per task, and as long as the average is above the cutoff it will skip all IO from that task - instead of caching the first 512k after every seek. Backups and large file copies should thus entirely bypass the cache.

In the event of a data IO error on the flash it will try to recover by reading from disk or invalidating cache entries. For unrecoverable errors (meta data or dirty data), caching is automatically disabled; if dirty data was present in the cache it first disables writeback caching and waits for all dirty data to be flushed.

Getting started: You' ll need make-bcache from the bcache-tools repository. Both the cache device and backing device must be formatted before use:

```
make-bcache -B /dev/sdb  
make-bcache -C /dev/sdc
```

make-bcache has the ability to format multiple devices at the same time - if you format your backing devices and cache device at the same time, you won't have to manually attach:

```
make-bcache -B /dev/sda /dev/sdb -C /dev/sdc
```

bcache-tools now ships udev rules, and bcache devices are known to the kernel immediately. Without udev, you can manually register devices like this:

```
echo /dev/sdb > /sys/fs/bcache/register
echo /dev/sdc > /sys/fs/bcache/register
```

Registering the backing device makes the bcache device show up in /dev; you can now format it and use it as normal. But the first time using a new bcache device, it'll be running in passthrough mode until you attach it to a cache. If you are thinking about using bcache later, it is recommended to setup all your slow devices as bcache backing devices without a cache, and you can choose to add a caching device later. See 'ATTACHING' section below.

The devices show up as:

```
/dev/bcache<N>
```

As well as (with udev):

```
/dev/bcache/by-uuid/<uuid>
/dev/bcache/by-label/<label>
```

To get started:

```
mkfs.ext4 /dev/bcache0
mount /dev/bcache0 /mnt
```

You can control bcache devices through sysfs at /sys/block/bcache<N>/bcache . You can also control them through /sys/fs/bcache/<cset-uuid>/ .

Cache devices are managed as sets; multiple caches per set isn't supported yet but will allow for mirroring of metadata and dirty data in the future. Your new cache set shows up as /sys/fs/bcache/<UUID>

## 20.1 Attaching

After your cache device and backing device are registered, the backing device must be attached to your cache set to enable caching. Attaching a backing device to a cache set is done thusly, with the UUID of the cache set in /sys/fs/bcache:

```
echo <CSET-UUID> > /sys/block/bcache0/bcache/attach
```

This only has to be done once. The next time you reboot, just reregister all your bcache devices. If a backing device has data in a cache somewhere, the /dev/bcache<N> device won't be created until the cache shows up - particularly important if you have writeback caching turned on.

If you're booting up and your cache device is gone and never coming back, you can force run the backing device:

```
echo 1 > /sys/block/sdb/bcache/running
```

(You need to use `/sys/block/sdb` (or whatever your backing device is called), not `/sys/block/bcache0`, because `bcache0` doesn't exist yet. If you're using a partition, the `bcache` directory would be at `/sys/block/sdb/sdb2/bcache`)

The backing device will still use that cache set if it shows up in the future, but all the cached data will be invalidated. If there was dirty data in the cache, don't expect the filesystem to be recoverable - you will have massive filesystem corruption, though `ext4`'s `fsck` does work miracles.

## 20.2 Error Handling

Bcache tries to transparently handle IO errors to/from the cache device without affecting normal operation; if it sees too many errors (the threshold is configurable, and defaults to 0) it shuts down the cache device and switches all the backing devices to passthrough mode.

- For reads from the cache, if they error we just retry the read from the backing device.
- For writethrough writes, if the write to the cache errors we just switch to invalidating the data at that lba in the cache (i.e. the same thing we do for a write that bypasses the cache)
- For writeback writes, we currently pass that error back up to the filesystem/userspace. This could be improved - we could retry it as a write that skips the cache so we don't have to error the write.
- When we detach, we first try to flush any dirty data (if we were running in writeback mode). It currently doesn't do anything intelligent if it fails to read some of the dirty data, though.

## 20.3 Howto/cookbook

### A) Starting a bcache with a missing caching device

If registering the backing device doesn't help, it's already there, you just need to force it to run without the cache:

```
host:~# echo /dev/sdb1 > /sys/fs/bcache/register
[ 119.844831] bcache: register_bcache() error opening /dev/sdb1: device_
↪already registered
```

Next, you try to register your caching device if it's present. However if it's absent, or registration fails for some reason, you can still start your bcache without its cache, like so:

```
host:/sys/block/sdb/sdb1/bcache# echo 1 > running
```

Note that this may cause data loss if you were running in writeback mode.

### B) Bcache does not find its cache:

```
host:/sys/block/md5/bcache# echo 0226553a-37cf-41d5-b3ce-8b1e944543a8 ↵
↵> attach
[ 1933.455082] bcache: bch_cached_dev_attach() Couldn't find uuid for ↵
↵md5 in set
[ 1933.478179] bcache: __cached_dev_store() Can't attach 0226553a-
↵37cf-41d5-b3ce-8b1e944543a8
[ 1933.478179] : cache set not found
```

In this case, the caching device was simply not registered at boot or disappeared and came back, and needs to be (re-)registered:

```
host:/sys/block/md5/bcache# echo /dev/sdh2 > /sys/fs/bcache/register
```

### C) Corrupt bcache crashes the kernel at device registration time:

This should never happen. If it does happen, then you have found a bug! Please report it to the bcache development list: [linux-bcache@vger.kernel.org](mailto:linux-bcache@vger.kernel.org)

Be sure to provide as much information that you can including kernel dmesg output if available so that we may assist.

### D) Recovering data without bcache:

If bcache is not available in the kernel, a filesystem on the backing device is still available at an 8KiB offset. So either via a loopdev of the backing device created with `-offset 8K`, or any value defined by `-data-offset` when you originally formatted bcache with `make-bcache`.

For example:

```
losetup -o 8192 /dev/loop0 /dev/your_bcache_backing_dev
```

This should present your unmodified backing device data in `/dev/loop0`

If your cache is in writethrough mode, then you can safely discard the cache device without losing data.

### E) Wiping a cache device

```
host:~# wipefs -a /dev/sdh2
16 bytes were erased at offset 0x1018 (bcache)
they were: c6 85 73 f6 4e 1a 45 ca 82 65 f5 7f 48 ba 6d 81
```

After you boot back with bcache enabled, you recreate the cache and attach it:

```
host:~# make-bcache -C /dev/sdh2
UUID:                7be7e175-8f4c-4f99-94b2-9c904d227045
Set UUID:            5bc072a8-ab17-446d-9744-e247949913c1
version:             0
nbuckets:            106874
block_size:          1
```

(continues on next page)

(continued from previous page)

```

bucket_size:          1024
nr_in_set:            1
nr_this_dev:          0
first_bucket:         1
[ 650.511912] bcache: run_cache_set() invalidating existing data
[ 650.549228] bcache: register_cache() registered cache device sdh2

```

start backing device with missing cache:

```
host:/sys/block/md5/bcache# echo 1 > running
```

attach new cache:

```

host:/sys/block/md5/bcache# echo 5bc072a8-ab17-446d-9744-e247949913c1 >
↳attach
[ 865.276616] bcache: bch_cached_dev_attach() Caching md5 as bcache0 on
↳set 5bc072a8-ab17-446d-9744-e247949913c1

```

F) Remove or replace a caching device:

```

host:/sys/block/sda/sda7/bcache# echo 1 > detach
[ 695.872542] bcache: cached_dev_detach_finish() Caching disabled
↳for sda7

host:~# wipefs -a /dev/nvme0n1p4
wipefs: error: /dev/nvme0n1p4: probing initialization failed: Device
↳or resource busy
Oops, it's disabled, but not unregistered, so it's still protected

```

We need to go and unregister it:

```

host:/sys/fs/bcache/b7ba27a1-2398-4649-8ae3-0959f57ba128# ls -l cache0
lrwxrwxrwx 1 root root 0 Feb 25 18:33 cache0 -> ../../../../devices/
↳pci0000:00/0000:00:1d.0/0000:70:00.0/nvme/nvme0/nvme0n1/nvme0n1p4/bcache/
host:/sys/fs/bcache/b7ba27a1-2398-4649-8ae3-0959f57ba128# echo 1 > stop
kernel: [ 917.041908] bcache: cache_set_free() Cache set b7ba27a1-2398-
↳4649-8ae3-0959f57ba128 unregistered

```

Now we can wipe it:

```

host:~# wipefs -a /dev/nvme0n1p4
/dev/nvme0n1p4: 16 bytes were erased at offset 0x00001018 (bcache): c6 85
↳73 f6 4e 1a 45 ca 82 65 f5 7f 48 ba 6d 81

```

G) dm-crypt and bcache

First setup bcache unencrypted and then install dmccrypt on top of /dev/bcache<N> This will work faster than if you dmccrypt both the backing and caching devices and then install bcache on top. [benchmarks?]

H) Stop/free a registered bcache to wipe and/or recreate it

Suppose that you need to free up all bcache references so that you can fdisk run and re-register a changed partition table, which won't work if there are any active backing or caching devices left on it:

- 1) Is it present in `/dev/bcache*` ? (there are times where it won't be)

If so, it's easy:

```
host:/sys/block/bcache0/bcache# echo 1 > stop
```

- 2) But if your backing device is gone, this won't work:

```
host:/sys/block/bcache0# cd bcache
bash: cd: bcache: No such file or directory
```

In this case, you may have to unregister the dmccrypt block device that references this bcache to free it up:

```
host:~# dmsetup remove oldds1
bcache: bcache_device_free() bcache0 stopped
bcache: cache_set_free() Cache set 5bc072a8-ab17-446d-9744-
↳e247949913c1 unregistered
```

This causes the backing bcache to be removed from `/sys/fs/bcache` and then it can be reused. This would be true of any block device stacking where bcache is a lower device.

- 3) In other cases, you can also look in `/sys/fs/bcache/`:

```
host:/sys/fs/bcache# ls -l */{cache?,bdev?}
lrwxrwxrwx 1 root root 0 Mar  5 09:39 0226553a-37cf-41d5-b3ce-
↳8b1e944543a8/bdev1 -> ../../../../devices/virtual/block/dm-1/bcache/
lrwxrwxrwx 1 root root 0 Mar  5 09:39 0226553a-37cf-41d5-b3ce-
↳8b1e944543a8/cache0 -> ../../../../devices/virtual/block/dm-4/bcache/
lrwxrwxrwx 1 root root 0 Mar  5 09:39 5bc072a8-ab17-446d-9744-
↳e247949913c1/cache0 -> ../../../../devices/pci0000:00/0000:00:01.0/
↳0000:01:00.0/ata10/host9/target9:0:0/9:0:0:0/block/sdl/sdl2/bcache/
```

The device names will show which UUID is relevant, cd in that directory and stop the cache:

```
host:/sys/fs/bcache/5bc072a8-ab17-446d-9744-e247949913c1# echo 1 >
↳stop
```

This will free up bcache references and let you reuse the partition for other purposes.

## 20.4 Troubleshooting performance

Bcache has a bunch of config options and tunables. The defaults are intended to be reasonable for typical desktop and server workloads, but they're not what you want for getting the best possible numbers when benchmarking.

- Backing device alignment

The default metadata size in bcache is 8k. If your backing device is RAID based, then be sure to align this by a multiple of your stride width using `make-bcache -data-offset`. If you intend to expand your disk array in the future, then

multiply a series of primes by your raid stripe size to get the disk multiples that you would like.

For example: If you have a 64k stripe size, then the following offset would provide alignment for many common RAID5 data spindle counts:

```
64k * 2*2*2*3*3*5*7 bytes = 161280k
```

That space is wasted, but for only 157.5MB you can grow your RAID 5 volume to the following data-spindle counts without re-aligning:

```
3,4,5,6,7,8,9,10,12,14,15,18,20,21 ...
```

- Bad write performance

If write performance is not what you expected, you probably wanted to be running in writeback mode, which isn't the default (not due to a lack of maturity, but simply because in writeback mode you'll lose data if something happens to your SSD):

```
# echo writeback > /sys/block/bcache0/bcache/cache_mode
```

- Bad performance, or traffic not going to the SSD that you'd expect

By default, bcache doesn't cache everything. It tries to skip sequential IO - because you really want to be caching the random IO, and if you copy a 10 gigabyte file you probably don't want that pushing 10 gigabytes of randomly accessed data out of your cache.

But if you want to benchmark reads from cache, and you start out with fio writing an 8 gigabyte test file - so you want to disable that:

```
# echo 0 > /sys/block/bcache0/bcache/sequential_cutoff
```

To set it back to the default (4 mb), do:

```
# echo 4M > /sys/block/bcache0/bcache/sequential_cutoff
```

- Traffic's still going to the spindle/still getting cache misses

In the real world, SSDs don't always keep up with disks - particularly with slower SSDs, many disks being cached by one SSD, or mostly sequential IO. So you want to avoid being bottlenecked by the SSD and having it slow everything down.

To avoid that bcache tracks latency to the cache device, and gradually throttles traffic if the latency exceeds a threshold (it does this by cranking down the sequential bypass).

You can disable this if you need to by setting the thresholds to 0:

```
# echo 0 > /sys/fs/bcache/<cache set>/congested_read_threshold_us
# echo 0 > /sys/fs/bcache/<cache set>/congested_write_threshold_us
```

The default is 2000 us (2 milliseconds) for reads, and 20000 for writes.

- Still getting cache misses, of the same data

One last issue that sometimes trips people up is actually an old bug, due to the way cache coherency is handled for cache misses. If a btree node is full, a cache miss won't be able to insert a key for the new data and the data won't be written to the cache.

In practice this isn't an issue because as soon as a write comes along it'll cause the btree node to be split, and you need almost no write traffic for this to not show up enough to be noticeable (especially since bcache's btree nodes are huge and index large regions of the device). But when you're benchmarking, if you're trying to warm the cache by reading a bunch of data and there's no other traffic - that can be a problem.

Solution: warm the cache by doing writes, or use the testing branch (there's a fix for the issue there).

## 20.5 Sysfs - backing device

Available at `/sys/block/<bdev>/bcache`, `/sys/block/bcache*/bcache` and (if attached) `/sys/fs/bcache/<cset-uuid>/bdev*`

**attach** Echo the UUID of a cache set to this file to enable caching.

**cache\_mode** Can be one of either writethrough, writeback, writearound or none.

**clear\_stats** Writing to this file resets the running total stats (not the day/hour/5 minute decaying versions).

**detach** Write to this file to detach from a cache set. If there is dirty data in the cache, it will be flushed first.

**dirty\_data** Amount of dirty data for this backing device in the cache. Continuously updated unlike the cache set's version, but may be slightly off.

**label** Name of underlying device.

**readahead** Size of readahead that should be performed. Defaults to 0. If set to e.g. 1M, it will round cache miss reads up to that size, but without overlapping existing cache entries.

**running** 1 if bcache is running (i.e. whether the `/dev/bcache` device exists, whether it's in passthrough mode or caching).

**sequential\_cutoff** A sequential IO will bypass the cache once it passes this threshold; the most recent 128 IOs are tracked so sequential IO can be detected even when it isn't all done at once.

**sequential\_merge** If non zero, bcache keeps a list of the last 128 requests submitted to compare against all new requests to determine which new requests are sequential continuations of previous requests for the purpose of determining sequential cutoff. This is necessary if the sequential cutoff value is greater than the maximum acceptable sequential size for any single request.

**state** The backing device can be in one of four different states:

no cache: Has never been attached to a cache set.

clean: Part of a cache set, and there is no cached dirty data.

**dirty:** Part of a cache set, and there is cached dirty data.

**inconsistent:** The backing device was forcibly run by the user when there was dirty data cached but the cache set was unavailable; whatever data was on the backing device has likely been corrupted.

**stop** Write to this file to shut down the bcache device and close the backing device.

**writeback\_delay** When dirty data is written to the cache and it previously did not contain any, waits some number of seconds before initiating writeback. Defaults to 30.

**writeback\_percent** If nonzero, bcache tries to keep around this percentage of the cache dirty by throttling background writeback and using a PD controller to smoothly adjust the rate.

**writeback\_rate** Rate in sectors per second - if writeback\_percent is nonzero, background writeback is throttled to this rate. Continuously adjusted by bcache but may also be set by the user.

**writeback\_running** If off, writeback of dirty data will not take place at all. Dirty data will still be added to the cache until it is mostly full; only meant for benchmarking. Defaults to on.

### 20.5.1 Sysfs - backing device stats

There are directories with these numbers for a running total, as well as versions that decay over the past day, hour and 5 minutes; they're also aggregated in the cache set directory as well.

**bypassed** Amount of IO (both reads and writes) that has bypassed the cache

**cache\_hits, cache\_misses, cache\_hit\_ratio** Hits and misses are counted per individual IO as bcache sees them; a partial hit is counted as a miss.

**cache\_bypass\_hits, cache\_bypass\_misses** Hits and misses for IO that is intended to skip the cache are still counted, but broken out here.

**cache\_miss\_collisions** Counts instances where data was going to be inserted into the cache from a cache miss, but raced with a write and data was already present (usually 0 since the synchronization for cache misses was rewritten)

**cache\_readaheads** Count of times readahead occurred.

### 20.5.2 Sysfs - cache set

Available at `/sys/fs/bcache/<cset-uuid>`

**average\_key\_size** Average data per key in the btree.

**bdev<0..n>** Symlink to each of the attached backing devices.

**block\_size** Block size of the cache devices.

**btree\_cache\_size** Amount of memory currently used by the btree cache

**bucket\_size** Size of buckets

**cache<0..n>** Symlink to each of the cache devices comprising this cache set.

**cache\_available\_percent** Percentage of cache device which doesn't contain dirty data, and could potentially be used for writeback. This doesn't mean this space isn't used for clean cached data; the unused statistic (in `priority_stats`) is typically much lower.

**clear\_stats** Clears the statistics associated with this cache

**dirty\_data** Amount of dirty data is in the cache (updated when garbage collection runs).

**flash\_vol\_create** Echoing a size to this file (in human readable units, k/M/G) creates a thinly provisioned volume backed by the cache set.

**io\_error\_halflife, io\_error\_limit** These determines how many errors we accept before disabling the cache. Each error is decayed by the half life (in # ios). If the decaying count reaches `io_error_limit` dirty data is written out and the cache is disabled.

**journal\_delay\_ms** Journal writes will delay for up to this many milliseconds, unless a cache flush happens sooner. Defaults to 100.

**root\_usage\_percent** Percentage of the root btree node in use. If this gets too high the node will split, increasing the tree depth.

**stop** Write to this file to shut down the cache set - waits until all attached backing devices have been shut down.

**tree\_depth** Depth of the btree (A single node btree has depth 0).

**unregister** Detaches all backing devices and closes the cache devices; if dirty data is present it will disable writeback caching and wait for it to be flushed.

### 20.5.3 Sysfs - cache set internal

This directory also exposes timings for a number of internal operations, with separate files for average duration, average frequency, last occurrence and max duration: garbage collection, btree read, btree node sorts and btree splits.

**active\_journal\_entries** Number of journal entries that are newer than the index.

**btree\_nodes** Total nodes in the btree.

**btree\_used\_percent** Average fraction of btree in use.

**bset\_tree\_stats** Statistics about the auxiliary search trees

**btree\_cache\_max\_chain** Longest chain in the btree node cache's hash table

**cache\_read\_races** Counts instances where while data was being read from the cache, the bucket was reused and invalidated - i.e. where the pointer was stale after the read completed. When this occurs the data is reread from the backing device.

**trigger\_gc** Writing to this file forces garbage collection to run.

## 20.5.4 Sysfs - Cache device

Available at `/sys/block/<cdev>/bcache`

**block\_size** Minimum granularity of writes - should match hardware sector size.

**btree\_written** Sum of all btree writes, in (kilo/mega/giga) bytes

**bucket\_size** Size of buckets

**cache\_replacement\_policy** One of either lru, fifo or random.

**discard** Boolean; if on a discard/TRIM will be issued to each bucket before it is reused. Defaults to off, since SATA TRIM is an unqueued command (and thus slow).

**freelist\_percent** Size of the freelist as a percentage of nbuckets. Can be written to to increase the number of buckets kept on the freelist, which lets you artificially reduce the size of the cache at runtime. Mostly for testing purposes (i.e. testing how different size caches affect your hit rate), but since buckets are discarded when they move on to the freelist will also make the SSD's garbage collection easier by effectively giving it more reserved space.

**io\_errors** Number of errors that have occurred, decayed by `io_error_halflife`.

**metadata\_written** Sum of all non data writes (btree writes and all other metadata).

**nbuckets** Total buckets in this cache

**priority\_stats** Statistics about how recently data in the cache has been accessed. This can reveal your working set size. Unused is the percentage of the cache that doesn't contain any data. Metadata is bcache's metadata overhead. Average is the average priority of cache buckets. Next is a list of quantiles with the priority threshold of each.

**written** Sum of all data that has been written to the cache; comparison with `btree_written` gives the amount of write inflation in bcache.



## THE ANDROID BINDERFS FILESYSTEM

Android binderfs is a filesystem for the Android binder IPC mechanism. It allows to dynamically add and remove binder devices at runtime. Binder devices located in a new binderfs instance are independent of binder devices located in other binderfs instances. Mounting a new binderfs instance makes it possible to get a set of private binder devices.

### 21.1 Mounting binderfs

Android binderfs can be mounted with:

```
mkdir /dev/binderfs
mount -t binder binder /dev/binderfs
```

at which point a new instance of binderfs will show up at `/dev/binderfs`. In a fresh instance of binderfs no binder devices will be present. There will only be a `binder-control` device which serves as the request handler for binderfs. Mounting another binderfs instance at a different location will create a new and separate instance from all other binderfs mounts. This is identical to the behavior of e.g. `devpts` and `tmpfs`. The Android binderfs filesystem can be mounted in user namespaces.

### 21.2 Options

**max** binderfs instances can be mounted with a limit on the number of binder devices that can be allocated. The `max=<count>` mount option serves as a per-instance limit. If `max=<count>` is set then only `<count>` number of binder devices can be allocated in this binderfs instance.

**stats** Using `stats=global` enables global binder statistics. `stats=global` is only available for a binderfs instance mounted in the initial user namespace. An attempt to use the option to mount a binderfs instance in another user namespace will return a permission error.

## 21.3 Allocating binder Devices

To allocate a new binder device in a binderfs instance a request needs to be sent through the binder-control device node. A request is sent in the form of an `ioctl()`.

What a program needs to do is to open the binder-control device node and send a `BINDER_CTL_ADD` request to the kernel. Users of binderfs need to tell the kernel which name the new binder device should get. By default a name can only contain up to `BINDERFS_MAX_NAME` chars including the terminating zero byte.

Once the request is made via an `ioctl()` passing a `struct binder_device` with the name to the kernel it will allocate a new binder device and return the major and minor number of the new device in the struct (This is necessary because binderfs allocates a major device number dynamically.). After the `ioctl()` returns there will be a new binder device located under `/dev/binderfs` with the chosen name.

## 21.4 Deleting binder Devices

Binderfs binder devices can be deleted via `unlink()`. This means that the `rm()` tool can be used to delete them. Note that the binder-control device cannot be deleted since this would make the binderfs instance unuseable. The binder-control device will be deleted when the binderfs instance is unmounted and all references to it have been dropped.

## KERNEL SUPPORT FOR MISCELLANEOUS BINARY FORMATS (BINFMT\_MISC)

This Kernel feature allows you to invoke almost (for restrictions see below) every program by simply typing its name in the shell. This includes for example compiled Java(TM), Python or Emacs programs.

To achieve this you must tell `binfmt_misc` which interpreter has to be invoked with which binary. `Binfmt_misc` recognises the binary-type by matching some bytes at the beginning of the file with a magic byte sequence (masking out specified bits) you have supplied. `Binfmt_misc` can also recognise a filename extension aka `.com` or `.exe`.

First you must mount `binfmt_misc`:

```
mount binfmt_misc -t binfmt_misc /proc/sys/fs/binfmt_misc
```

To actually register a new binary type, you have to set up a string looking like `:name:type:offset:magic:mask:interpreter:flags` (where you can choose the `:` upon your needs) and echo it to `/proc/sys/fs/binfmt_misc/register`.

Here is what the fields mean:

- **name** is an identifier string. A new `/proc` file will be created with this name below `/proc/sys/fs/binfmt_misc`; cannot contain slashes `/` for obvious reasons.
- **type** is the type of recognition. Give `M` for magic and `E` for extension.
- **offset** is the offset of the magic/mask in the file, counted in bytes. This defaults to 0 if you omit it (i.e. you write `:name:type::magic...`). Ignored when using filename extension matching.
- **magic** is the byte sequence `binfmt_misc` is matching for. The magic string may contain hex-encoded characters like `\x0a` or `\xA4`. Note that you must escape any NUL bytes; parsing halts at the first one. In a shell environment you might have to write `\\x0a` to prevent the shell from eating your `\`. If you chose filename extension matching, this is the extension to be recognised (without the `.`, the `\x0a` specials are not allowed). Extension matching is case sensitive, and slashes `/` are not allowed!
- **mask** is an (optional, defaults to all `0xff`) mask. You can mask out some bits from matching by supplying a string like `magic` and as long as `magic`. The mask is anded with the byte sequence of the file. Note that you must

escape any NUL bytes; parsing halts at the first one. Ignored when using filename extension matching.

- **interpreter** is the program that should be invoked with the binary as first argument (specify the full path)
- **flags** is an optional field that controls several aspects of the invocation of the interpreter. It is a string of capital letters, each controls a certain aspect. The following flags are supported:

**P - preserve-argv[0]** Legacy behavior of `binfmt_misc` is to overwrite the original `argv[0]` with the full path to the binary. When this flag is included, `binfmt_misc` will add an argument to the argument vector for this purpose, thus preserving the original `argv[0]`. e.g. If your `interp` is set to `/bin/foo` and you run `blah` (which is in `/usr/local/bin`), then the kernel will execute `/bin/foo` with `argv[]` set to `["/bin/foo", "/usr/local/bin/blah", "blah"]`. The `interp` has to be aware of this so it can execute `/usr/local/bin/blah` with `argv[]` set to `["blah"]`.

**O - open-binary** Legacy behavior of `binfmt_misc` is to pass the full path of the binary to the interpreter as an argument. When this flag is included, `binfmt_misc` will open the file for reading and pass its descriptor as an argument, instead of the full path, thus allowing the interpreter to execute non-readable binaries. This feature should be used with care - the interpreter has to be trusted not to emit the contents of the non-readable binary.

**C - credentials** Currently, the behavior of `binfmt_misc` is to calculate the credentials and security token of the new process according to the interpreter. When this flag is included, these attributes are calculated according to the binary. It also implies the `O` flag. This feature should be used with care as the interpreter will run with root permissions when a `setuid` binary owned by root is run with `binfmt_misc`.

**F - fix binary** The usual behaviour of `binfmt_misc` is to spawn the binary lazily when the `misc` format file is invoked. However, this doesn't work very well in the face of mount namespaces and `changeroots`, so the `F` mode opens the binary as soon as the emulation is installed and uses the opened image to spawn the emulator, meaning it is always available once installed, regardless of how the environment changes.

There are some restrictions:

- the whole register string may not exceed 1920 characters
- the magic must reside in the first 128 bytes of the file, i.e. `offset+size(magic)` has to be less than 128
- the interpreter string may not exceed 127 characters

To use `binfmt_misc` you have to mount it first. You can mount it with `mount -t binfmt_misc none /proc/sys/fs/binfmt_misc` command, or you can add





## **THE LINUX RAPIDIO SUBSYSTEM**

### **23.1 Floppy Driver**

#### **23.1.1 FAQ list:**

A FAQ list may be found in the fdutils package (see below), and also at <<http://fdutils.linux.lu/faq.html>>.

#### **23.1.2 LILO configuration options (Thinkpad users, read this)**

The floppy driver is configured using the 'floppy=' option in lilo. This option can be typed at the boot prompt, or entered in the lilo configuration file.

Example: If your kernel is called linux-2.6.9, type the following line at the lilo boot prompt (if you have a thinkpad):

```
linux-2.6.9 floppy=thinkpad
```

You may also enter the following line in /etc/lilo.conf, in the description of linux-2.6.9:

```
append = "floppy=thinkpad"
```

Several floppy related options may be given, example:

```
linux-2.6.9 floppy=daring floppy=two_fdc  
append = "floppy=daring floppy=two_fdc"
```

If you give options both in the lilo config file and on the boot prompt, the option strings of both places are concatenated, the boot prompt options coming last. That's why there are also options to restore the default behavior.

### 23.1.3 Module configuration options

If you use the floppy driver as a module, use the following syntax:

```
modprobe floppy floppy="<options>"
```

Example:

```
modprobe floppy floppy="omnibook messages"
```

If you need certain options enabled every time you load the floppy driver, you can put:

```
options floppy floppy="omnibook messages"
```

in a configuration file in `/etc/modprobe.d/`.

The floppy driver related options are:

**floppy=asus\_pci** Sets the bit mask to allow only units 0 and 1. (default)

**floppy=daring** Tells the floppy driver that you have a well behaved floppy controller. This allows more efficient and smoother operation, but may fail on certain controllers. This may speed up certain operations.

**floppy=0,daring** Tells the floppy driver that your floppy controller should be used with caution.

**floppy=one\_fdc** Tells the floppy driver that you have only one floppy controller. (default)

**floppy=two\_fdc / floppy=<address>,two\_fdc** Tells the floppy driver that you have two floppy controllers. The second floppy controller is assumed to be at `<address>`. This option is not needed if the second controller is at address `0x370`, and if you use the `'cmos'` option.

**floppy=thinkpad** Tells the floppy driver that you have a Thinkpad. Thinkpads use an inverted convention for the disk change line.

**floppy=0,thinkpad** Tells the floppy driver that you don't have a Thinkpad.

**floppy=omnibook / floppy=nodma** Tells the floppy driver not to use Dma for data transfers. This is needed on HP Omnibooks, which don't have a workable DMA channel for the floppy driver. This option is also useful if you frequently get "Unable to allocate DMA memory" messages. Indeed, dma memory needs to be continuous in physical memory, and is thus harder to find, whereas non-dma buffers may be allocated in virtual memory. However, I advise against this if you have an FDC without a FIFO (8272A or 82072). 82072A and later are OK. You also need at least a 486 to use nodma. If you use nodma mode, I suggest you also set the FIFO threshold to 10 or lower, in order to limit the number of data transfer interrupts.

If you have a FIFO-able FDC, the floppy driver automatically falls back on non DMA mode if no DMA-able memory can be found. If you want to avoid this, explicitly ask for 'yesdma' .

**floppy=yesdma** Tells the floppy driver that a workable DMA channel is available. (default)

**floppy=nofifo** Disables the FIFO entirely. This is needed if you get "Bus master arbitration error" messages from your Ethernet card (or from other devices) while accessing the floppy.

**floppy=usefifo** Enables the FIFO. (default)

**floppy=<threshold>,fifo\_depth** Sets the FIFO threshold. This is mostly relevant in DMA mode. If this is higher, the floppy driver tolerates more interrupt latency, but it triggers more interrupts (i.e. it imposes more load on the rest of the system). If this is lower, the interrupt latency should be lower too (faster processor). The benefit of a lower threshold is less interrupts.

To tune the fifo threshold, switch on over/underrun messages using 'floppycontrol -messages' . Then access a floppy disk. If you get a huge amount of "Over/Underrun - retrying" messages, then the fifo threshold is too low. Try with a higher value, until you only get an occasional Over/Underrun. It is a good idea to compile the floppy driver as a module when doing this tuning. Indeed, it allows to try different fifo values without rebooting the machine for each test. Note that you need to do 'floppycontrol -messages' every time you re-insert the module.

Usually, tuning the fifo threshold should not be needed, as the default (0xa) is reasonable.

**floppy=<drive>,<type>,cmos** Sets the CMOS type of <drive> to <type>. This is mandatory if you have more than two floppy drives (only two can be described in the physical CMOS), or if your BIOS uses non-standard CMOS types. The CMOS types are:

0	Use the value of the physical CMOS
1	5 1/4 DD
2	5 1/4 HD
3	3 1/2 DD
4	3 1/2 HD
5	3 1/2 ED
6	3 1/2 ED
16	unknown or not installed

(Note: there are two valid types for ED drives. This is because 5 was initially chosen to represent floppy tapes, and 6 for ED drives. AMI ignored this, and used 5 for ED drives. That's why the floppy driver handles both.)

**floppy=unexpected\_interrupts** Print a warning message when an unexpected interrupt is received. (default)

**floppy=no\_unexpected\_interrupts / floppy=L40SX** Don't print a message when an unexpected interrupt is received. This is needed on IBM L40SX laptops in certain video modes. (There seems to be an interaction between video and floppy. The unexpected interrupts affect only performance, and can be safely ignored.)

**floppy=broken\_dcl** Don't use the disk change line, but assume that the disk was changed whenever the device node is reopened. Needed on some boxes where the disk change line is broken or unsupported. This should be regarded as a stopgap measure, indeed it makes floppy operation less efficient due to unneeded cache flushings, and slightly more unreliable. Please verify your cable, connection and jumper settings if you have any DCL problems. However, some older drives, and also some laptops are known not to have a DCL.

**floppy=debug** Print debugging messages.

**floppy=messages** Print informational messages for some operations (disk change notifications, warnings about over and underruns, and about autodetection).

**floppy=silent\_dcl\_clear** Uses a less noisy way to clear the disk change line (which doesn't involve seeks). Implied by 'daring' option.

**floppy=<nr>,irq** Sets the floppy IRQ to <nr> instead of 6.

**floppy=<nr>,dma** Sets the floppy DMA channel to <nr> instead of 2.

**floppy=slow** Use PS/2 stepping rate:

PS/2 floppies have much slower step rates than regular  
↪ floppies.  
It's been recommended that take about 1/4 of the default speed  
in some more extreme cases.

### 23.1.4 Supporting utilities and additional documentation:

Additional parameters of the floppy driver can be configured at runtime. Utilities which do this can be found in the fdutils package. This package also contains a new version of mtools which allows to access high capacity disks (up to 1992K on a high density 3 1/2 disk!). It also contains additional documentation about the floppy driver.

The latest version can be found at fdutils homepage:

<http://fdutils.linux.lu>

The fdutils releases can be found at:

<http://fdutils.linux.lu/download.html>

<http://www.tux.org/pub/knaff/fdutils/>

<ftp://metalab.unc.edu/pub/Linux/utils/disk-management/>

### 23.1.5 Reporting problems about the floppy driver

If you have a question or a bug report about the floppy driver, mail me at [Alain.Knaff@poboxes.com](mailto:Alain.Knaff@poboxes.com) . If you post to Usenet, preferably use comp.os.linux.hardware. As the volume in these groups is rather high, be sure to include the word “floppy” (or “FLOPPY” ) in the subject line. If the reported problem happens when mounting floppy disks, be sure to mention also the type of the filesystem in the subject line.

Be sure to read the FAQ before mailing/posting any bug reports!

Alain

### 23.1.6 Changelog

**10-30-2004** : Cleanup, updating, add reference to module configuration. James Nelson <[james4765@gmail.com](mailto:james4765@gmail.com)>

**6-3-2000** : Original Document

## 23.2 Network Block Device (TCP version)

### 23.2.1 1) Overview

What is it: With this compiled in the kernel (or as a module), Linux can use a remote server as one of its block devices. So every time the client computer wants to read, e.g., /dev/nb0, it sends a request over TCP to the server, which will reply with the data read. This can be used for stations with low disk space (or even diskless) to borrow disk space from another computer. Unlike NFS, it is possible to put any filesystem on it, etc.

For more information, or to download the nbd-client and nbd-server tools, go to <http://nbd.sf.net/>.

The nbd kernel module need only be installed on the client system, as the nbd-server is completely in userspace. In fact, the nbd-server has been successfully ported to other operating systems, including Windows.

### 23.2.2 A) NBD parameters

**max\_part** Number of partitions per device (default: 0).

**nbd\_max** Number of block devices that should be initialized (default: 16).

## 23.3 Linux and parallel port IDE devices

PARIDE v1.03 (c) 1997-8 Grant Guenther <[grant@torque.net](mailto:grant@torque.net)>

### 23.3.1 1. Introduction

Owing to the simplicity and near universality of the parallel port interface to personal computers, many external devices such as portable hard-disk, CD-ROM, LS-120 and tape drives use the parallel port to connect to their host computer. While some devices (notably scanners) use ad-hoc methods to pass commands and data through the parallel port interface, most external devices are actually identical to an internal model, but with a parallel-port adapter chip added in. Some of the original parallel port adapters were little more than mechanisms for multiplexing a SCSI bus. (The Iomega PPA-3 adapter used in the ZIP drives is an example of this approach). Most current designs, however, take a different approach. The adapter chip reproduces a small ISA or IDE bus in the external device and the communication protocol provides operations for reading and writing device registers, as well as data block transfer functions. Sometimes, the device being addressed via the parallel cable is a standard SCSI controller like an NCR 5380. The “ditto” family of external tape drives use the ISA replicator to interface a floppy disk controller, which is then connected to a floppy-tape mechanism. The vast majority of external parallel port devices, however, are now based on standard IDE type devices, which require no intermediate controller. If one were to open up a parallel port CD-ROM drive, for instance, one would find a standard ATAPI CD-ROM drive, a power supply, and a single adapter that interconnected a standard PC parallel port cable and a standard IDE cable. It is usually possible to exchange the CD-ROM device with any other device using the IDE interface.

The document describes the support in Linux for parallel port IDE devices. It does not cover parallel port SCSI devices, “ditto” tape drives or scanners. Many different devices are supported by the parallel port IDE subsystem, including:

- MicroSolutions backpack CD-ROM
- MicroSolutions backpack PD/CD
- MicroSolutions backpack hard-drives
- MicroSolutions backpack 8000t tape drive
- SyQuest EZ-135, EZ-230 & SparQ drives
- Avatar Shark
- Imation Superdisk LS-120
- Maxell Superdisk LS-120
- FreeCom Power CD
- Hewlett-Packard 5GB and 8GB tape drives
- Hewlett-Packard 7100 and 7200 CD-RW drives

as well as most of the clone and no-name products on the market.

To support such a wide range of devices, PARIDE, the parallel port IDE subsystem, is actually structured in three parts. There is a base paride module which provides a registry and some common methods for accessing the parallel ports. The second component is a set of high-level drivers for each of the different types of supported devices:

pd	IDE disk
pcd	ATAPI CD-ROM
pf	ATAPI disk
pt	ATAPI tape
pg	ATAPI generic

(Currently, the pg driver is only used with CD-R drives).

The high-level drivers function according to the relevant standards. The third component of PARIDE is a set of low-level protocol drivers for each of the parallel port IDE adapter chips. Thanks to the interest and encouragement of Linux users from many parts of the world, support is available for almost all known adapter protocols:

aten	ATEN EH-100	(HK)
bpck	Microsolutions backpack	(US)
comm	DataStor (old-type) "commuter" adapter	(TW)
dstr	DataStor EP-2000	(TW)
epat	Shuttle EPAT	(UK)
epia	Shuttle EPIA	(UK)
fit2	FIT TD-2000	(US)
fit3	FIT TD-3000	(US)
friq	Freecom IQ cable	(DE)
frpw	Freecom Power	(DE)
kbic	KingByte KBIC-951A and KBIC-971A	(TW)
ktti	KT Technology PHd adapter	(SG)
on20	OnSpec 90c20	(US)
on26	OnSpec 90c26	(US)

### 23.3.2 2. Using the PARIDE subsystem

While configuring the Linux kernel, you may choose either to build the PARIDE drivers into your kernel, or to build them as modules.

In either case, you will need to select "Parallel port IDE device support" as well as at least one of the high-level drivers and at least one of the parallel port communication protocols. If you do not know what kind of parallel port adapter is used in your drive, you could begin by checking the file names and any text files on your DOS installation floppy. Alternatively, you can look at the markings on the adapter chip itself. That's usually sufficient to identify the correct device.

You can actually select all the protocol modules, and allow the PARIDE subsystem to try them all for you.

For the “brand-name” products listed above, here are the protocol and high-level drivers that you would use:

Manufacturer	Model	Driver	Protocol
MicroSolutions	CD-ROM	pcd	bpck
MicroSolutions	PD drive	pf	bpck
MicroSolutions	hard-drive	pd	bpck
MicroSolutions	8000t tape	pt	bpck
SyQuest	EZ, SparQ	pd	epat
Imation	Superdisk	pf	epat
Maxell	Superdisk	pf	friq
Avatar	Shark	pd	epat
FreeCom	CD-ROM	pcd	frpw
Hewlett-Packard	5GB Tape	pt	epat
Hewlett-Packard	7200e (CD)	pcd	epat
Hewlett-Packard	7200e (CD-R)	pg	epat

### 2.1 Configuring built-in drivers

We recommend that you get to know how the drivers work and how to configure them as loadable modules, before attempting to compile a kernel with the drivers built-in.

If you built all of your PARIDE support directly into your kernel, and you have just a single parallel port IDE device, your kernel should locate it automatically for you. If you have more than one device, you may need to give some command line options to your bootloader (eg: LILO), how to do that is beyond the scope of this document.

The high-level drivers accept a number of command line parameters, all of which are documented in the source files in `linux/drivers/block/paride`. By default, each driver will automatically try all parallel ports it can find, and all protocol types that have been installed, until it finds a parallel port IDE adapter. Once it finds one, the probe stops. So, if you have more than one device, you will need to tell the drivers how to identify them. This requires specifying the port address, the protocol identification number and, for some devices, the drive’s chain ID. While your system is booting, a number of messages are displayed on the console. Like all such messages, they can be reviewed with the ‘`dmesg`’ command. Among those messages will be some lines like:

```
paride: bpck registered as protocol 0
paride: epat registered as protocol 1
```

The numbers will always be the same until you build a new kernel with different protocol selections. You should note these numbers as you will need them to identify the devices.

If you happen to be using a MicroSolutions backpack device, you will also need to know the unit ID number for each drive. This is usually the last two digits of the drive’s serial number (but read MicroSolutions’ documentation about this).

As an example, let’s assume that you have a MicroSolutions PD/CD drive with unit

ID number 36 connected to the parallel port at 0x378, a SyQuest EZ-135 connected to the chained port on the PD/CD drive and also an Imation Superdisk connected to port 0x278. You could give the following options on your boot command:

```
pd.drive0=0x378,1 pf.drive0=0x278,1 pf.drive1=0x378,0,36
```

In the last option, pf.drive1 configures device /dev/pf1, the 0x378 is the parallel port base address, the 0 is the protocol registration number and 36 is the chain ID.

Please note: while PARIDE will work both with and without the PARPORT parallel port sharing system that is included by the “Parallel port support” option, PARPORT must be included and enabled if you want to use chains of devices on the same parallel port.

## 2.2 Loading and configuring PARIDE as modules

It is much faster and simpler to get to understand the PARIDE drivers if you use them as loadable kernel modules.

**Note 1:** using these drivers with the “kerneld” automatic module loading system is not recommended for beginners, and is not documented here.

**Note 2:** if you build PARPORT support as a loadable module, PARIDE must also be built as loadable modules, and PARPORT must be loaded before the PARIDE modules.

To use PARIDE, you must begin by:

```
insmod paride
```

this loads a base module which provides a registry for the protocols, among other tasks.

Then, load as many of the protocol modules as you think you might need. As you load each module, it will register the protocols that it supports, and print a log message to your kernel log file and your console. For example:

```
# insmod epat
paride: epat registered as protocol 0
# insmod kbic
paride: k951 registered as protocol 1
paride: k971 registered as protocol 2
```

Finally, you can load high-level drivers for each kind of device that you have connected. By default, each driver will autoprobe for a single device, but you can support up to four similar devices by giving their individual co-ordinates when you load the driver.

For example, if you had two no-name CD-ROM drives both using the KingByte KBIC-951A adapter, one on port 0x378 and the other on 0x3bc you could give the following command:

```
# insmod pcd drive0=0x378,1 drive1=0x3bc,1
```

For most adapters, giving a port address and protocol number is sufficient, but check the source files in `linux/drivers/block/paride` for more information. (Hopefully someone will write some man pages one day !).

As another example, here's what happens when PARPORT is installed, and a SyQuest EZ-135 is attached to port 0x378:

```
# insmod paride
paride: version 1.0 installed
# insmod epat
paride: epat registered as protocol 0
# insmod pd
pd: pd version 1.0, major 45, cluster 64, nice 0
pda: Sharing parport1 at 0x378
pda: epat 1.0, Shuttle EPAT chip c3 at 0x378, mode 5 (EPP-32), delay 1
pda: SyQuest EZ135A, 262144 blocks [128M], (512/16/32), removable media
pda: pda1
```

Note that the last line is the output from the generic partition table scanner - in this case it reports that it has found a disk with one partition.

### 2.3 Using a PARIDE device

Once the drivers have been loaded, you can access PARIDE devices in the same way as their traditional counterparts. You will probably need to create the device "special files". Here is a simple script that you can cut to a file and execute:

```
#!/bin/bash
#
# mkd -- a script to create the device special files for the PARIDE_
↳ subsystem
#
function mkdev {
    mknod $1 $2 $3 $4 ; chmod 0660 $1 ; chown root:disk $1
}
#
function pd {
    D=$( printf "\\$( printf "x%03x" ${1 + 97} ) )
    mkdev pd$D b 45 ${1 * 16}
    for P in 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
    do mkdev pd$D$P b 45 ${1 * 16 + $P}
    done
}
#
cd /dev
#
for u in 0 1 2 3 ; do pd $u ; done
for u in 0 1 2 3 ; do mkdev pcd$u b 46 $u ; done
for u in 0 1 2 3 ; do mkdev pf$u b 47 $u ; done
for u in 0 1 2 3 ; do mkdev pt$u c 96 $u ; done
for u in 0 1 2 3 ; do mkdev npt$u c 96 ${u + 128} ; done
for u in 0 1 2 3 ; do mkdev pg$u c 97 $u ; done
#
# end of mkd
```

With the device files and drivers in place, you can access PARIDE devices like any

other Linux device. For example, to mount a CD-ROM in pcd0, use:

```
mount /dev/pcd0 /cdrom
```

If you have a fresh Avatar Shark cartridge, and the drive is pda, you might do something like:

```
fdisk /dev/pda          -- make a new partition table with
                        partition 1 of type 83
mke2fs /dev/pda1       -- to build the file system
mkdir /shark           -- make a place to mount the disk
mount /dev/pda1 /shark
```

Devices like the Imation superdisk work in the same way, except that they do not have a partition table. For example to make a 120MB floppy that you could share with a DOS system:

```
mkdosfs /dev/pf0
mount /dev/pf0 /mnt
```

## 2.4 The pf driver

The pf driver is intended for use with parallel port ATAPI disk devices. The most common devices in this category are PD drives and LS-120 drives. Traditionally, media for these devices are not partitioned. Consequently, the pf driver does not support partitioned media. This may be changed in a future version of the driver.

## 2.5 Using the pt driver

The pt driver for parallel port ATAPI tape drives is a minimal driver. It does not yet support many of the standard tape ioctl operations. For best performance, a block size of 32KB should be used. You will probably want to set the parallel port delay to 0, if you can.

## 2.6 Using the pg driver

The pg driver can be used in conjunction with the cdrecord program to create CD-ROMs. Please get cdrecord version 1.6.1 or later from <ftp://ftp.fokus.gmd.de/pub/unix/cdrecord/>. To record CD-R media your parallel port should ideally be set to EPP mode, and the “port delay” should be set to 0. With those settings it is possible to record at 2x speed without any buffer underruns. If you cannot get the driver to work in EPP mode, try to use “bidirectional” or “PS/2” mode and 1x speeds only.

### 23.3.3 3. Troubleshooting

#### 3.1 Use EPP mode if you can

The most common problems that people report with the PARIDE drivers concern the parallel port CMOS settings. At this time, none of the PARIDE protocol modules support ECP mode, or any ECP combination modes. If you are able to do so, please set your parallel port into EPP mode using your CMOS setup procedure.

#### 3.2 Check the port delay

Some parallel ports cannot reliably transfer data at full speed. To offset the errors, the PARIDE protocol modules introduce a “port delay” between each access to the i/o ports. Each protocol sets a default value for this delay. In most cases, the user can override the default and set it to 0 - resulting in somewhat higher transfer rates. In some rare cases (especially with older 486 systems) the default delays are not long enough. If you experience corrupt data transfers, or unexpected failures, you may wish to increase the port delay. The delay can be programmed using the “driveN” parameters to each of the high-level drivers. Please see the notes above, or read the comments at the beginning of the driver source files in `linux/drivers/block/paride`.

#### 3.3 Some drives need a printer reset

There appear to be a number of “noname” external drives on the market that do not always power up correctly. We have noticed this with some drives based on OnSpec and older Freecom adapters. In these rare cases, the adapter can often be reinitialised by issuing a “printer reset” on the parallel port. As the reset operation is potentially disruptive in multiple device environments, the PARIDE drivers will not do it automatically. You can however, force a printer reset by doing:

```
insmod lp reset=1
rmmod lp
```

If you have one of these marginal cases, you should probably build your paride drivers as modules, and arrange to do the printer reset before loading the PARIDE drivers.

#### 3.4 Use the verbose option and dmesg if you need help

While a lot of testing has gone into these drivers to make them work as smoothly as possible, problems will arise. If you do have problems, please check all the obvious things first: does the drive work in DOS with the manufacturer’s drivers? If that doesn’t yield any useful clues, then please make sure that only one drive is hooked to your system, and that either (a) PARPORT is enabled or (b) no other device driver is using your parallel port (check in `/proc/ioports`). Then, load the appropriate drivers (you can load several protocol modules if you want) as in:

```
# insmod paride
# insmod epat
# insmod bpck
# insmod kbic
...
# insmod pd verbose=1
```

(using the correct driver for the type of device you have, of course). The `verbose=1` parameter will cause the drivers to log a trace of their activity as they attempt to locate your drive.

Use `dmesg` to capture a log of all the PARIDE messages (any messages beginning with `paride:`, a protocol module's name or a driver's name) and include that with your bug report. You can submit a bug report in one of two ways. Either send it directly to the author of the PARIDE suite, by e-mail to [grant@torque.net](mailto:grant@torque.net), or join the linux-parport mailing list and post your report there.

### 3.5 For more information or help

You can join the linux-parport mailing list by sending a mail message to:

[linux-parport-request@torque.net](mailto:linux-parport-request@torque.net)

with the single word:

```
subscribe
```

in the body of the mail message (not in the subject line). Please be sure that your mail program is correctly set up when you do this, as the list manager is a robot that will subscribe you using the reply address in your mail headers. REMOVE any anti-spam gimmicks you may have in your mail headers, when sending mail to the list server.

You might also find some useful information on the linux-parport web pages (although they are not always up to date) at

<http://web.archive.org/web/%2E/http://www.torque.net/parport/>

## 23.4 Using the RAM disk block device with Linux

### 23.4.1 1) Overview

The RAM disk driver is a way to use main system memory as a block device. It is required for `initrd`, an initial filesystem used if you need to load modules in order to access the root filesystem (see `Documentation/admin-guide/initrd.rst`). It can also be used for a temporary filesystem for crypto work, since the contents are erased on reboot.

The RAM disk dynamically grows as more space is required. It does this by using RAM from the buffer cache. The driver marks the buffers it is using as dirty so that the VM subsystem does not try to reclaim them later.

The RAM disk supports up to 16 RAM disks by default, and can be reconfigured to support an unlimited number of RAM disks (at your own risk). Just change the configuration symbol `BLK_DEV_RAM_COUNT` in the Block drivers config menu and (re)build the kernel.

To use RAM disk support with your system, run `./MAKEDEV ram` from the `/dev` directory. RAM disks are all major number 1, and start with minor number 0 for `/dev/ram0`, etc. If used, modern kernels use `/dev/ram0` for an `initrd`.

The new RAM disk also has the ability to load compressed RAM disk images, allowing one to squeeze more programs onto an average installation or rescue floppy disk.

### 23.4.2 2) Parameters

#### 2a) Kernel Command Line Parameters

**ramdisk\_size=N** Size of the ramdisk.

This parameter tells the RAM disk driver to set up RAM disks of N k size. The default is 4096 (4 MB).

#### 2b) Module parameters

**rd\_nr** /dev/ramX devices created.

**max\_part** Maximum partition number.

**rd\_size** See `ramdisk_size`.

### 23.4.3 3) Using “rdev -r”

The usage of the word (two bytes) that “rdev -r” sets in the kernel image is as follows. The low 11 bits (0 -> 10) specify an offset (in 1 k blocks) of up to 2 MB ( $2^{11}$ ) of where to find the RAM disk (this used to be the size). Bit 14 indicates that a RAM disk is to be loaded, and bit 15 indicates whether a prompt/wait sequence is to be given before trying to read the RAM disk. Since the RAM disk dynamically grows as data is being written into it, a size field is not required. Bits 11 to 13 are not currently used and may as well be zero. These numbers are no magical secrets, as seen below:

```
./arch/x86/kernel/setup.c:#define RAMDISK_IMAGE_START_MASK    0x07FF
./arch/x86/kernel/setup.c:#define RAMDISK_PROMPT_FLAG      0x8000
./arch/x86/kernel/setup.c:#define RAMDISK_LOAD_FLAG        0x4000
```

Consider a typical two floppy disk setup, where you will have the kernel on disk one, and have already put a RAM disk image onto disk #2.

Hence you want to set bits 0 to 13 as 0, meaning that your RAM disk starts at an offset of 0 kB from the beginning of the floppy. The command line equivalent is: “`ramdisk_start=0`”

You want bit 14 as one, indicating that a RAM disk is to be loaded. The command line equivalent is: “`load_ramdisk=1`”

You want bit 15 as one, indicating that you want a prompt/keypress sequence so that you have a chance to switch floppy disks. The command line equivalent is: "prompt\_ramdisk=1"

Putting that together gives  $2^{15} + 2^{14} + 0 = 49152$  for an rdev word. So to create disk one of the set, you would do:

```
/usr/src/linux# cat arch/x86/boot/zImage > /dev/fd0
/usr/src/linux# rdev /dev/fd0 /dev/fd0
/usr/src/linux# rdev -r /dev/fd0 49152
```

If you make a boot disk that has LILO, then for the above, you would use:

```
append = "ramdisk_start=0 load_ramdisk=1 prompt_ramdisk=1"
```

Since the default start = 0 and the default prompt = 1, you could use:

```
append = "load_ramdisk=1"
```

#### **23.4.4 4) An Example of Creating a Compressed RAM Disk**

To create a RAM disk image, you will need a spare block device to construct it on. This can be the RAM disk device itself, or an unused disk partition (such as an unmounted swap partition). For this example, we will use the RAM disk device, "/dev/ram0" .

Note: This technique should not be done on a machine with less than 8 MB of RAM. If using a spare disk partition instead of /dev/ram0, then this restriction does not apply.

- a) Decide on the RAM disk size that you want. Say 2 MB for this example. Create it by writing to the RAM disk device. (This step is not currently required, but may be in the future.) It is wise to zero out the area (esp. for disks) so that maximal compression is achieved for the unused blocks of the image that you are about to create:

```
dd if=/dev/zero of=/dev/ram0 bs=1k count=2048
```

- b) Make a filesystem on it. Say ext2fs for this example:

```
mke2fs -vm0 /dev/ram0 2048
```

- c) Mount it, copy the files you want to it (eg: /etc/\* /dev/\* ...) and unmount it again.
- d) Compress the contents of the RAM disk. The level of compression will be approximately 50% of the space used by the files. Unused space on the RAM disk will compress to almost nothing:

```
dd if=/dev/ram0 bs=1k count=2048 | gzip -v9 > /tmp/ram_image.gz
```

- e) Put the kernel onto the floppy:

```
dd if=zImage of=/dev/fd0 bs=1k
```

- f) Put the RAM disk image onto the floppy, after the kernel. Use an offset that is slightly larger than the kernel, so that you can put another (possibly larger) kernel onto the same floppy later without overlapping the RAM disk image. An offset of 400 kB for kernels about 350 kB in size would be reasonable. Make sure offset+size of ram\_image.gz is not larger than the total space on your floppy (usually 1440 kB):

```
dd if=/tmp/ram_image.gz of=/dev/fd0 bs=1k seek=400
```

- g) Use “rdev” to set the boot device, RAM disk offset, prompt flag, etc. For prompt\_ramdisk=1, load\_ramdisk=1, ramdisk\_start=400, one would have  $2^{15} + 2^{14} + 400 = 49552$ :

```
rdev /dev/fd0 /dev/fd0
rdev -r /dev/fd0 49552
```

That is it. You now have your boot/root compressed RAM disk floppy. Some users may wish to combine steps (d) and (f) by using a pipe.

Paul Gortmaker 12/95

### 23.4.5 Changelog:

**10-22-04** : Updated to reflect changes in command line options, remove obsolete references, general cleanup. James Nelson ([james4765@gmail.com](mailto:james4765@gmail.com))

**12-95** : Original Document

## 23.5 zram: Compressed RAM-based block devices

### 23.5.1 Introduction

The zram module creates RAM-based block devices named /dev/zram<id> (<id> = 0, 1, ...). Pages written to these disks are compressed and stored in memory itself. These disks allow very fast I/O and compression provides good amounts of memory savings. Some of the use cases include /tmp storage, use as swap disks, various caches under /var and maybe many more. :)

Statistics for individual zram devices are exported through sysfs nodes at /sys/block/zram<id>/

### 23.5.2 Usage

There are several ways to configure and manage zram device(-s):

- a) using zram and zram\_control sysfs attributes
- b) using zramctl utility, provided by util-linux ([util-linux@vger.kernel.org](mailto:util-linux@vger.kernel.org)).

In this document we will describe only ‘manual’ zram configuration steps, IOW, zram and zram\_control sysfs attributes.

In order to get a better idea about zramctl please consult util-linux documentation, zramctl man-page or zramctl -help. Please be informed that zram maintainers do not develop/maintain util-linux or zramctl, should you have any questions please contact [util-linux@vger.kernel.org](mailto:util-linux@vger.kernel.org)

Following shows a typical sequence of steps for using zram.

### 23.5.3 WARNING

For the sake of simplicity we skip error checking parts in most of the examples below. However, it is your sole responsibility to handle errors.

zram sysfs attributes always return negative values in case of errors. The list of possible return codes:

- EBUSY	an attempt to modify an attribute that cannot be changed once the device has been initialised. Please reset device first.
- ENOMEM	zram was not able to allocate enough memory to fulfil your needs.
- EINVAL	invalid input has been provided.

If you use 'echo', the returned value is set by the 'echo' utility, and, in general case, something like:

```
echo 3 > /sys/block/zram0/max_comp_streams
if [ $? -ne 0 ]; then
    handle_error
fi
```

should suffice.

### 23.5.4 1) Load Module

```
modprobe zram num_devices=4
```

This creates 4 devices: /dev/zram{0,1,2,3}

num\_devices parameter is optional and tells zram how many devices should be pre-created. Default: 1.

### 23.5.5 2) Set max number of compression streams

Regardless of the value passed to this attribute, ZRAM will always allocate multiple compression streams - one per online CPU - thus allowing several concurrent compression operations. The number of allocated compression streams goes down when some of the CPUs become offline. There is no single-compression-stream mode anymore, unless you are running a UP system or have only 1 CPU online.

To find out how many streams are currently available:

```
cat /sys/block/zram0/max_comp_streams
```

### 23.5.6 3) Select compression algorithm

Using `comp_algorithm` device attribute one can see available and currently selected (shown in square brackets) compression algorithms, or change the selected compression algorithm (once the device is initialised there is no way to change compression algorithm).

Examples:

```
#show supported compression algorithms
cat /sys/block/zram0/comp_algorithm
lzo [lz4]

#select lzo compression algorithm
echo lzo > /sys/block/zram0/comp_algorithm
```

For the time being, the `comp_algorithm` content does not necessarily show every compression algorithm supported by the kernel. We keep this list primarily to simplify device configuration and one can configure a new device with a compression algorithm that is not listed in `comp_algorithm`. The thing is that, internally, ZRAM uses Crypto API and, if some of the algorithms were built as modules, it's impossible to list all of them using, for instance, `/proc/crypto` or any other method. This, however, has an advantage of permitting the usage of custom crypto compression modules (implementing S/W or H/W compression).

### 23.5.7 4) Set Disksize

Set disk size by writing the value to sysfs node `'disksize'`. The value can be either in bytes or you can use mem suffixes. Examples:

```
# Initialize /dev/zram0 with 50MB disksize
echo $((50*1024*1024)) > /sys/block/zram0/disksize

# Using mem suffixes
echo 256K > /sys/block/zram0/disksize
echo 512M > /sys/block/zram0/disksize
echo 1G > /sys/block/zram0/disksize
```

Note: There is little point creating a zram of greater than twice the size of memory since we expect a 2:1 compression ratio. Note that zram uses about 0.1% of the size of the disk when not in use so a huge zram is wasteful.

### 23.5.8 5) Set memory limit: Optional

Set memory limit by writing the value to sysfs node 'mem\_limit'. The value can be either in bytes or you can use mem suffixes. In addition, you could change the value in runtime. Examples:

```
# limit /dev/zram0 with 50MB memory
echo $((50*1024*1024)) > /sys/block/zram0/mem_limit

# Using mem suffixes
echo 256K > /sys/block/zram0/mem_limit
echo 512M > /sys/block/zram0/mem_limit
echo 1G > /sys/block/zram0/mem_limit

# To disable memory limit
echo 0 > /sys/block/zram0/mem_limit
```

### 23.5.9 6) Activate

```
mkswap /dev/zram0
swapon /dev/zram0

mkfs.ext4 /dev/zram1
mount /dev/zram1 /tmp
```

### 23.5.10 7) Add/remove zram devices

zram provides a control interface, which enables dynamic (on-demand) device addition and removal.

In order to add a new /dev/zramX device, perform a read operation on the hot\_add attribute. This will return either the new device's device id (meaning that you can use /dev/zram<id>) or an error code.

Example:

```
cat /sys/class/zram-control/hot_add
1
```

To remove the existing /dev/zramX device (where X is a device id) execute:

```
echo X > /sys/class/zram-control/hot_remove
```

### 23.5.11 8) Stats

Per-device statistics are exported as various nodes under `/sys/block/zram<id>/`

A brief description of exported device attributes follows. For more details please read `Documentation/ABI/testing/sysfs-block-zram`.

Name	access	description
<code>disksize</code>	RW	show and set the device' s disk size
<code>initstate</code>	RO	shows the initialization state of the device
<code>reset</code>	WO	trigger device reset
<code>mem_used_max</code>	WO	reset the <code>mem_used_max</code> counter (see later)
<code>mem_limit</code>	WO	specifies the maximum amount of memory ZRAM can use to store the compressed data
<code>write-back_limit</code>	WO	specifies the maximum amount of write IO zram can write out to backing device as 4KB unit
<code>write-back_limit_enable</code>	RW	show and set <code>writeback_limit</code> feature
<code>max_comp_streams</code>	RW	the number of possible concurrent compress operations
<code>comp_algorithm</code>	RW	show and change the compression algorithm
<code>compact</code>	WO	trigger memory compaction
<code>debug_stat</code>	RO	this file is used for zram debugging purposes
<code>backing_dev</code>	RW	set up backend storage for zram to write out
<code>idle</code>	WO	mark allocated slot as idle

User space is advised to use the following files to read the device statistics.

File `/sys/block/zram<id>/stat`

Represents block layer statistics. Read `Documentation/block/stat.rst` for details.

File `/sys/block/zram<id>/io_stat`

The `stat` file represents device' s I/O statistics not accounted by block layer and, thus, not available in `zram<id>/stat` file. It consists of a single line of text and contains the following stats separated by whitespace:

failed_reads	The number of failed reads
failed_writes	The number of failed writes
invalid_io	The number of non-page-size-aligned I/O requests
notify_free	Depending on device usage scenario it may account <ul style="list-style-type: none"> <li>a) the number of pages freed because of swap slot free notifications</li> <li>b) the number of pages freed because of REQ_OP_DISCARD requests sent by bio. The former ones are sent to a swap block device when a swap slot is freed, which implies that this disk is being used as a swap disk.</li> </ul> <p>The latter ones are sent by filesystem mounted with discard option, whenever some data blocks are getting discarded.</p>

File `/sys/block/zram<id>/mm_stat`

The `mm_stat` file represents the device's mm statistics. It consists of a single line of text and contains the following stats separated by whitespace:

orig_data_size	Uncompressed size of data stored in this disk. Unit: bytes
compr_data_size	Compressed size of data stored in this disk
mem_used_total	Amount of memory allocated for this disk. This includes allocator fragmentation and metadata overhead, allocated for this disk. So, allocator space efficiency can be calculated using <code>compr_data_size</code> and this statistic. Unit: bytes
mem_limit	The maximum amount of memory ZRAM can use to store the compressed data
mem_used_max	The maximum amount of memory zram has consumed to store the data
same_page	Number of same element filled pages written to this disk. No memory is allocated for such pages.
pages_compacted	Number of pages freed during compaction
huge_pages	Number of incompressible pages

File `/sys/block/zram<id>/bd_stat`

The `bd_stat` file represents a device's backing device statistics. It consists of a single line of text and contains the following stats separated by whitespace:

bd_count	size of data written in backing device. Unit: 4K bytes
bd_reads	the number of reads from backing device Unit: 4K bytes
bd_writes	the number of writes to backing device Unit: 4K bytes

### 23.5.12 9) Deactivate

```
swapoff /dev/zram0
umount /dev/zram1
```

### 23.5.13 10) Reset

Write any positive value to 'reset' sysfs node:

```
echo 1 > /sys/block/zram0/reset
echo 1 > /sys/block/zram1/reset
```

This frees all the memory allocated for the given device and resets the disksize to zero. You must set the disksize again before reusing the device.

### 23.5.14 Optional Feature

#### writeback

With CONFIG\_ZRAM\_WRITEBACK, zram can write idle/incompressible page to backing storage rather than keeping it in memory. To use the feature, admin should set up backing device via:

```
echo /dev/sda5 > /sys/block/zramX/backing_dev
```

before disksize setting. It supports only partition at this moment. If admin wants to use incompressible page writeback, they could do via:

```
echo huge > /sys/block/zramX/writeback
```

To use idle page writeback, first, user need to declare zram pages as idle:

```
echo all > /sys/block/zramX/idle
```

From now on, any pages on zram are idle pages. The idle mark will be removed until someone requests access of the block. IOW, unless there is access request, those pages are still idle pages.

Admin can request writeback of those idle pages at right timing via:

```
echo idle > /sys/block/zramX/writeback
```

With the command, zram writeback idle pages from memory to the storage.

If there are lots of write IO with flash device, potentially, it has flash wearout problem so that admin needs to design write limitation to guarantee storage health for entire product life.

To overcome the concern, zram supports “writeback\_limit” feature. The “writeback\_limit\_enable”’s default value is 0 so that it doesn’t limit any writeback. IOW, if admin wants to apply writeback budget, he should enable writeback\_limit\_enable via:

```
$ echo 1 > /sys/block/zramX/writeback_limit_enable
```

Once writeback\_limit\_enable is set, zram doesn’t allow any writeback until admin sets the budget via /sys/block/zramX/writeback\_limit.

(If admin doesn’t enable writeback\_limit\_enable, writeback\_limit’s value assigned via /sys/block/zramX/writeback\_limit is meaningless.)

If admin want to limit writeback as per-day 400M, he could do it like below:

```
$ MB_SHIFT=20
$ 4K_SHIFT=12
$ echo $((400<<MB_SHIFT>>4K_SHIFT)) > \
    /sys/block/zram0/writeback_limit.
$ echo 1 > /sys/block/zram0/writeback_limit_enable
```

If admins want to allow further write again once the budget is exhausted, he could do it like below:

```
$ echo $((400<<MB_SHIFT>>4K_SHIFT)) > \
    /sys/block/zram0/writeback_limit
```

If admin wants to see remaining writeback budget since last set:

```
$ cat /sys/block/zramX/writeback_limit
```

If admin want to disable writeback limit, he could do:

```
$ echo 0 > /sys/block/zramX/writeback_limit_enable
```

The writeback\_limit count will reset whenever you reset zram (e.g., system reboot, echo 1 > /sys/block/zramX/reset) so keeping how many of writeback happened until you reset the zram to allocate extra writeback budget in next setting is user’s job.

If admin wants to measure writeback count in a certain period, he could know it via /sys/block/zram0/bd\_stat’s 3rd column.

### 23.5.15 memory tracking

With CONFIG\_ZRAM\_MEMORY\_TRACKING, user can know information of the zram block. It could be useful to catch cold or incompressible pages of the process with \*pagemap.

If you enable the feature, you could see block state via /sys/kernel/debug/zram/zram0/block\_state". The output is as follows:

```
300    75.033841 .wh.
301    63.806904 s...
302    63.806919 ..hi
```

**First column** zram' s block index.

**Second column** access time since the system was booted

**Third column** state of the block:

**s:** same page

**w:** written page to backing store

**h:** huge page

**i:** idle page

First line of above example says 300th block is accessed at 75.033841sec and the block' s state is huge so it is written back to the backing storage. It' s a debugging feature so anyone shouldn' t rely on it to work properly.

Nitin Gupta [ngupta@vflare.org](mailto:ngupta@vflare.org)

## 23.6 Distributed Replicated Block Device - DRBD

### 23.6.1 Description

DRBD is a shared-nothing, synchronously replicated block device. It is designed to serve as a building block for high availability clusters and in this context, is a "drop-in" replacement for shared storage. Simplistically, you could see it as a network RAID 1.

Please visit <http://www.drbd.org> to find out more.

### kernel data structure for DRBD-9

This describes the in kernel data structure for DRBD-9. Starting with Linux v3.14 we are reorganizing DRBD to use this data structure.

## Basic Data Structure

A node has a number of DRBD resources. Each such resource has a number of devices (aka volumes) and connections to other nodes (“peer nodes”). Each DRBD device is represented by a block device locally.

The DRBD objects are interconnected to form a matrix as depicted below; a `drbd_peer_device` object sits at each intersection between a `drbd_device` and a `drbd_connection`:

/-----+-----+.....+-----\   resource   device     device
+-----+-----+.....+-----+
connection   peer_device     peer_device
+-----+-----+.....+-----+
: : : : :
: : : : :
+-----+-----+.....+-----+
connection   peer_device     peer_device
\-----+-----+.....+-----/

In this table, horizontally, devices can be accessed from resources by their volume number. Likewise, peer\_devices can be accessed from connections by their volume number. Objects in the vertical direction are connected by double linked lists. There are back pointers from peer\_devices to their connections a devices, and from connections and devices to their resource.

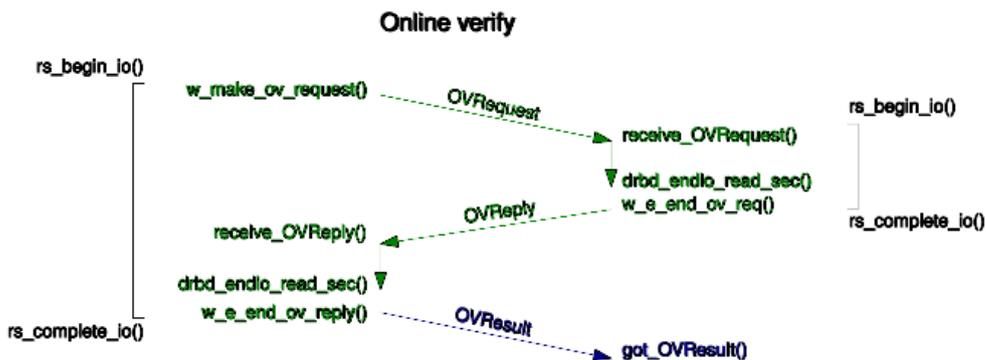
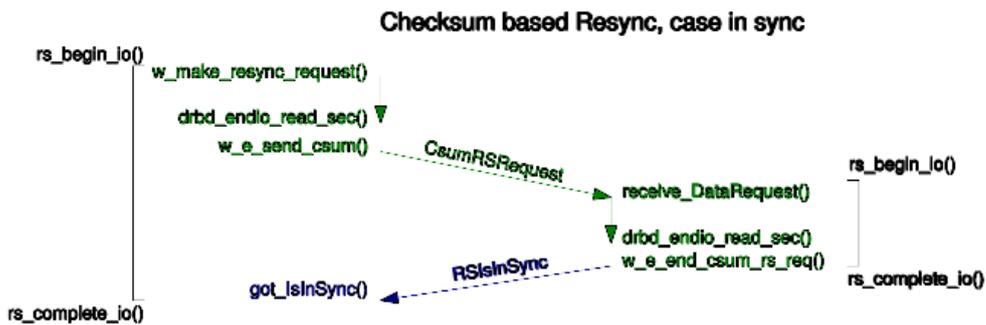
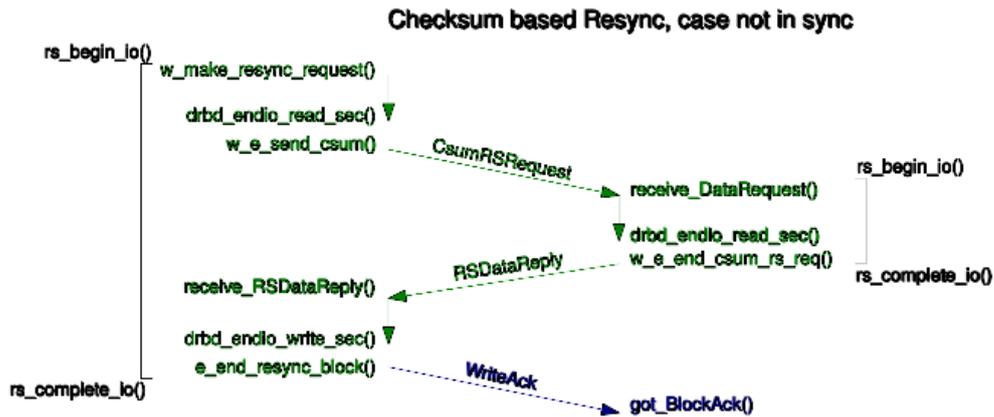
All resources are in the `drbd_resources` double-linked list. In addition, all devices can be accessed by their minor device number via the `drbd_devices` `idr`.

The `drbd_resource`, `drbd_connection`, and `drbd_device` objects are reference counted. The `peer_device` objects only serve to establish the links between devices and connections; their lifetime is determined by the lifetime of the device and connection which they reference.

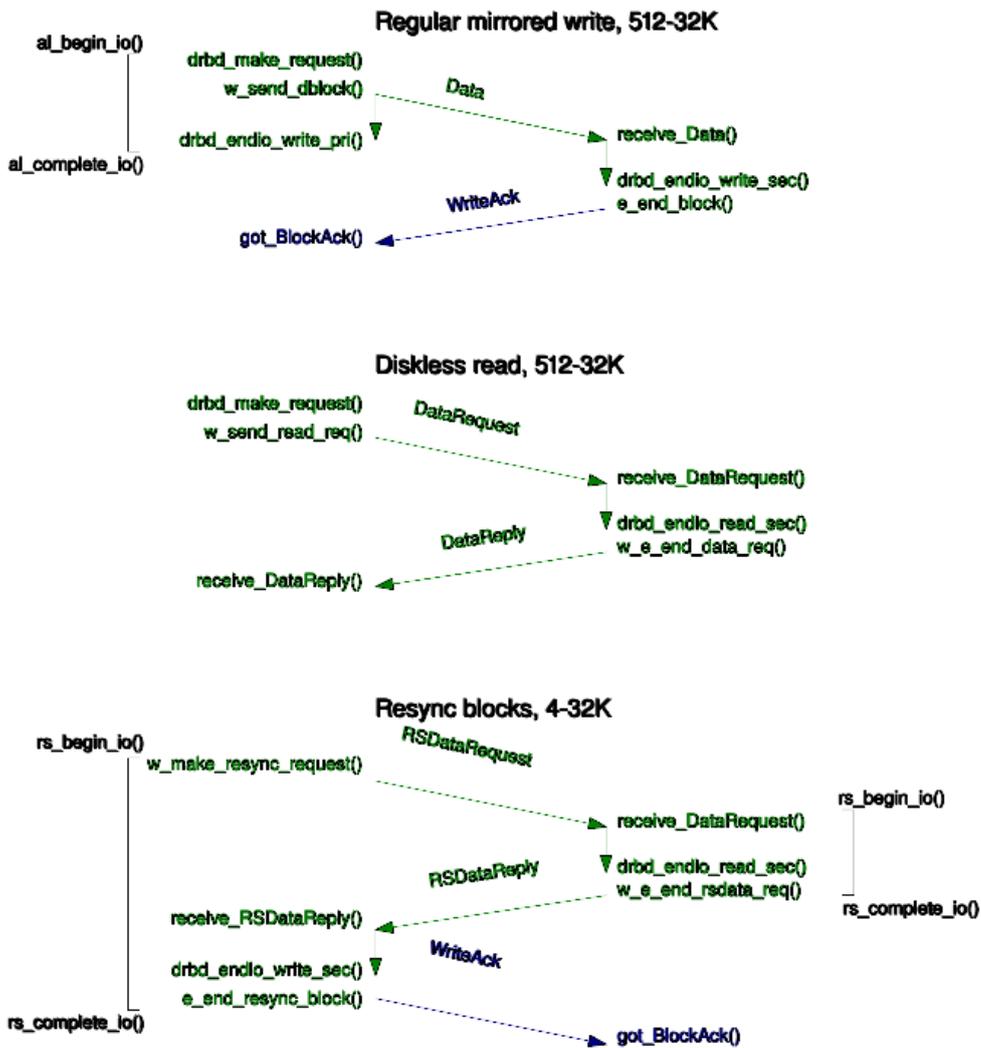
## Data flows that Relate some functions, and write packets

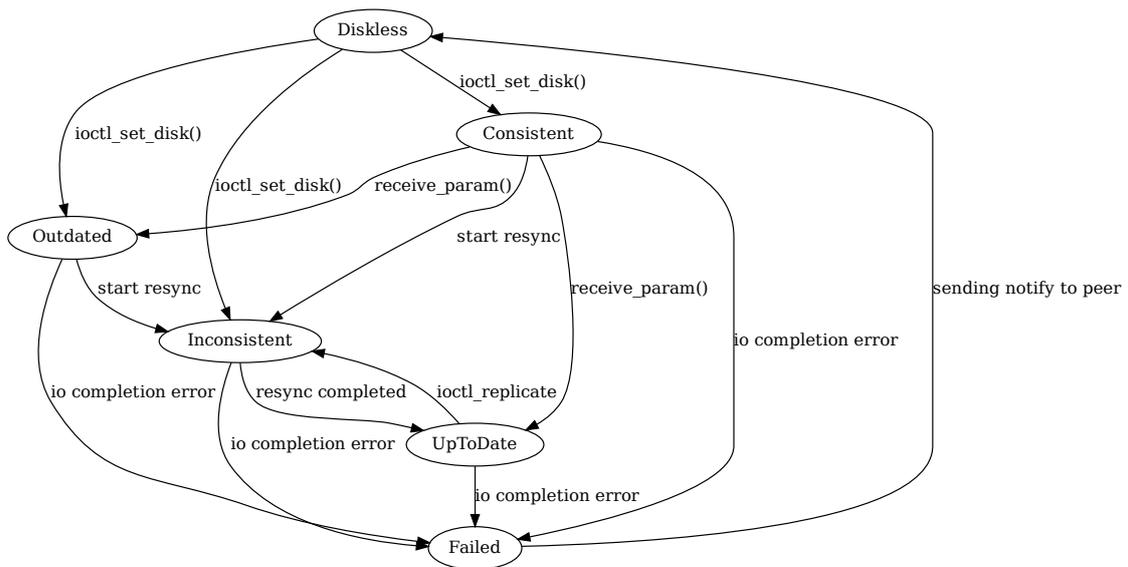
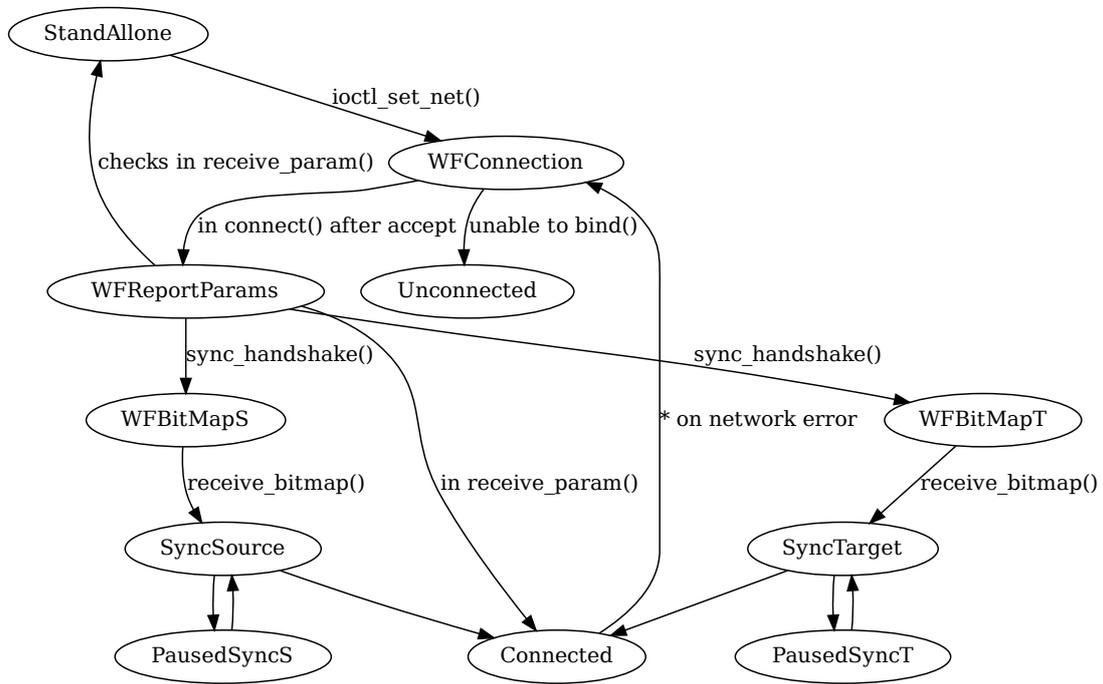
### Sub graphs of DRBD' s state transitions

DRBD-8.3 data flow



DRBD 8 data flow









## **BOOT CONFIGURATION**

**Author** Masami Hiramatsu <[mhiramat@kernel.org](mailto:mhiramat@kernel.org)>

### **24.1 Overview**

The boot configuration expands the current kernel command line to support additional key-value data when booting the kernel in an efficient way. This allows administrators to pass a structured-Key config file.

### **24.2 Config File Syntax**

The boot config syntax is a simple structured key-value. Each key consists of dot-connected-words, and key and value are connected by =. The value has to be terminated by semi-colon (;) or newline (\n). For array value, array entries are separated by comma (,).

```
KEY[.WORD[...]] = VALUE[, VALUE2[...]][;]
```

Unlike the kernel command line syntax, spaces are OK around the comma and =.

Each key word must contain only alphabets, numbers, dash (-) or underscore (\_). And each value only contains printable characters or spaces except for delimiters such as semi-colon (;), new-line (\n), comma (,), hash (#) and closing brace (}).

If you want to use those delimiters in a value, you can use either double-quotes ("VALUE") or single-quotes ('VALUE') to quote it. Note that you can not escape these quotes.

There can be a key which doesn't have value or has an empty value. Those keys are used for checking if the key exists or not (like a boolean).

### 24.2.1 Key-Value Syntax

The boot config file syntax allows user to merge partially same word keys by brace. For example:

```
foo.bar.baz = value1
foo.bar.qux.quux = value2
```

These can be written also in:

```
foo.bar {
    baz = value1
    qux.quux = value2
}
```

Or more shorter, written as following:

```
foo.bar { baz = value1; qux.quux = value2 }
```

In both styles, same key words are automatically merged when parsing it at boot time. So you can append similar trees or key-values.

### 24.2.2 Same-key Values

It is prohibited that two or more values or arrays share a same-key. For example,:

```
foo = bar, baz
foo = qux # !ERROR! we can not re-define same key
```

If you want to append the value to existing key as an array member, you can use += operator. For example:

```
foo = bar, baz
foo += qux
```

In this case, the key foo has bar, baz and qux.

However, a sub-key and a value can not co-exist under a parent key. For example, following config is NOT allowed.:

```
foo = value1
foo.bar = value2 # !ERROR! subkey "bar" and value "value1" can NOT co-exist
```

### 24.2.3 Comments

The config syntax accepts shell-script style comments. The comments starting with hash ( “#” ) until newline ( “n” ) will be ignored.

```
# comment line
foo = value # value is set to foo.
bar = 1, # 1st element
      2, # 2nd element
      3 # 3rd element
```

This is parsed as below:

```
foo = value
bar = 1, 2, 3
```

Note that you can not put a comment between value and delimiter(, or ;). This means following config has a syntax error

```
key = 1 # comment
      ,2
```

## 24.3 /proc/bootconfig

/proc/bootconfig is a user-space interface of the boot config. Unlike /proc/cmdline, this file shows the key-value style list. Each key-value pair is shown in each line with following style:

```
KEY[.WORDS...] = "[VALUE]"[, "VALUE2"...]
```

## 24.4 Boot Kernel With a Boot Config

Since the boot configuration file is loaded with initrd, it will be added to the end of the initrd (initramfs) image file with size, checksum and 12-byte magic word as below.

```
[initrd][bootconfig][size(u32)][checksum(u32)][#BOOTCONFIGn]
```

The Linux kernel decodes the last part of the initrd image in memory to get the boot configuration data. Because of this “piggyback” method, there is no need to change or update the boot loader and the kernel image itself.

To do this operation, Linux kernel provides “bootconfig” command under tools/bootconfig, which allows admin to apply or delete the config file to/from initrd image. You can build it by the following command:

```
# make -C tools/bootconfig
```

To add your boot config file to initrd image, run bootconfig as below (Old data is removed automatically if exists):

```
# tools/bootconfig/bootconfig -a your-config /boot/initrd.img-X.Y.Z
```

To remove the config from the image, you can use -d option as below:

```
# tools/bootconfig/bootconfig -d /boot/initrd.img-X.Y.Z
```

Then add “bootconfig” on the normal kernel command line to tell the kernel to look for the bootconfig at the end of the initrd file.

## 24.5 Config File Limitation

Currently the maximum config size size is 32KB and the total key-words (not key-value entries) must be under 1024 nodes. Note: this is not the number of entries but nodes, an entry must consume more than 2 nodes (a key-word and a value). So theoretically, it will be up to 512 key-value pairs. If keys contains 3 words in average, it can contain 256 key-value pairs. In most cases, the number of config items will be under 100 entries and smaller than 8KB, so it would be enough. If the node number exceeds 1024, parser returns an error even if the file size is smaller than 32KB. Anyway, since bootconfig command verifies it when appending a boot config to initrd image, user can notice it before boot.

## 24.6 Bootconfig APIs

User can query or loop on key-value pairs, also it is possible to find a root (prefix) key node and find key-values under that node.

If you have a key string, you can query the value directly with the key using `xbc_find_value()`. If you want to know what keys exist in the boot config, you can use `xbc_for_each_key_value()` to iterate key-value pairs. Note that you need to use `xbc_array_for_each_value()` for accessing each array' s value, e.g.:

```
vnode = NULL;
xbc_find_value("key.word", &vnode);
if (vnode && xbc_node_is_array(vnode))
    xbc_array_for_each_value(vnode, value) {
        printk("%s ", value);
    }
```

If you want to focus on keys which have a prefix string, you can use `xbc_find_node()` to find a node by the prefix string, and iterate keys under the prefix node with `xbc_node_for_each_key_value()`.

But the most typical usage is to get the named value under prefix or get the named array under prefix as below:

```
root = xbc_find_node("key.prefix");
value = xbc_node_find_value(root, "option", &vnode);
...
xbc_node_for_each_array_value(root, "array-option", value, anode) {
    ...
}
```

This accesses a value of “key.prefix.option” and an array of “key.prefix.array-option” .

Locking is not needed, since after initialization, the config becomes read-only. All data and keys must be copied if you need to modify it.

## 24.7 Functions and structures

bool **xbc\_node\_is\_value**(struct xbc\_node \* node)

Test the node is a value node

### Parameters

**struct xbc\_node \* node** An XBC node.

### Description

Test the **node** is a value node and return true if a value node, false if not.

bool **xbc\_node\_is\_key**(struct xbc\_node \* node)

Test the node is a key node

### Parameters

**struct xbc\_node \* node** An XBC node.

### Description

Test the **node** is a key node and return true if a key node, false if not.

bool **xbc\_node\_is\_array**(struct xbc\_node \* node)

Test the node is an arraied value node

### Parameters

**struct xbc\_node \* node** An XBC node.

### Description

Test the **node** is an arraied value node.

bool **xbc\_node\_is\_leaf**(struct xbc\_node \* node)

Test the node is a leaf key node

### Parameters

**struct xbc\_node \* node** An XBC node.

### Description

Test the **node** is a leaf key node which is a key node and has a value node or no child. Returns true if it is a leaf node, or false if not.

const char \* **xbc\_find\_value**(const char \* key, struct xbc\_node \*\* vnode)

Find a value which matches the key

### Parameters

**const char \* key** Search key

**struct xbc\_node \*\* vnode** A container pointer of XBC value node.

### Description

Search a value whose key matches **key** from whole of XBC tree and return the value if found. Found value node is stored in **\*vnode**. Note that this can return 0-length string and store NULL in **\*vnode** for key-only (non-value) entry.

struct xbc\_node \* **xbc\_find\_node**(const char \* key)

Find a node which matches the key

### Parameters

**const char \* key** Search key

### Description

Search a (key) node whose key matches **key** from whole of XBC tree and return the node if found. If not found, returns NULL.

**xbc\_array\_for\_each\_value**(anode, value)

Iterate value nodes on an array

### Parameters

**anode** An XBC arraied value node

**value** A value

### Description

Iterate array value nodes and values starts from **anode**. This is expected to be used with `xbc_find_value()` and `xbc_node_find_value()`, so that user can process each array entry node.

**xbc\_node\_for\_each\_child**(parent, child)

Iterate child nodes

### Parameters

**parent** An XBC node.

**child** Iterated XBC node.

### Description

Iterate child nodes of **parent**. Each child nodes are stored to **child**.

**xbc\_node\_for\_each\_array\_value**(node, key, anode, value)

Iterate array entries of given key

### Parameters

**node** An XBC node.

**key** A key string searched under **node**

**anode** Iterated XBC node of array entry.

**value** Iterated value of array entry.

### Description

Iterate array entries of given **key** under **node**. Each array entry node is stroed to **anode** and **value**. If the **node** doesn' t have **key** node, it does nothing. Note that even if the found key node has only one value (not array) this executes block once. Hoever, if the found key node has no value (key-only node), this does nothing. So don' t use this for testing the key-value pair existence.

**xbc\_node\_for\_each\_key\_value**(node, knode, value)

Iterate key-value pairs under a node

### Parameters

**node** An XBC node.

**knode** Iterated key node

**value** Iterated value string

### Description

Iterate key-value pairs under **node**. Each key node and value string are stored in **knode** and **value** respectively.

**xbc\_for\_each\_key\_value**(knode, value)  
Iterate key-value pairs

### Parameters

**knode** Iterated key node

**value** Iterated value string

### Description

Iterate key-value pairs in whole XBC tree. Each key node and value string are stored in **knode** and **value** respectively.

int **xbc\_node\_compose\_key**(struct xbc\_node \* node, char \* buf, size\_t size)  
Compose full key string of the XBC node

### Parameters

**struct xbc\_node \* node** An XBC node.

**char \* buf** A buffer to store the key.

**size\_t size** The size of the **buf**.

### Description

Compose the full-length key of the **node** into **buf**. Returns the total length of the key stored in **buf**. Or returns -EINVAL if **node** is NULL, and -ERANGE if the key depth is deeper than max depth.

struct xbc\_node \* **xbc\_root\_node**(void)  
Get the root node of extended boot config

### Parameters

**void** no arguments

### Description

Return the address of root node of extended boot config. If the extended boot config is not initialized, return NULL.

int **xbc\_node\_index**(struct xbc\_node \* node)  
Get the index of XBC node

### Parameters

**struct xbc\_node \* node** A target node of getting index.

### Description

Return the index number of **node** in XBC node list.

struct xbc\_node \* **xbc\_node\_get\_parent**(struct xbc\_node \* node)  
Get the parent XBC node

### Parameters

**struct xbc\_node \* node** An XBC node.

### Description

Return the parent node of **node**. If the node is top node of the tree, return NULL.

```
struct xbc_node * xbc_node_get_child(struct xbc_node * node)
    Get the child XBC node
```

### Parameters

**struct xbc\_node \* node** An XBC node.

### Description

Return the first child node of **node**. If the node has no child, return NULL.

```
struct xbc_node * xbc_node_get_next(struct xbc_node * node)
    Get the next sibling XBC node
```

### Parameters

**struct xbc\_node \* node** An XBC node.

### Description

Return the NEXT sibling node of **node**. If the node has no next sibling, return NULL. Note that even if this returns NULL, it doesn't mean **node** has no siblings. (You also has to check whether the parent's child node is **node** or not.)

```
const char * xbc_node_get_data(struct xbc_node * node)
    Get the data of XBC node
```

### Parameters

**struct xbc\_node \* node** An XBC node.

### Description

Return the data (which is always a null terminated string) of **node**. If the node has invalid data, warn and return NULL.

```
struct xbc_node * xbc_node_find_child(struct xbc_node * parent, const
                                     char * key)
    Find a child node which matches given key
```

### Parameters

**struct xbc\_node \* parent** An XBC node.

**const char \* key** A key string.

### Description

Search a node under **parent** which matches **key**. The **key** can contain several words jointed with '.'. If **parent** is NULL, this searches the node from whole tree. Return NULL if no node is matched.

```
const char * xbc_node_find_value(struct xbc_node * parent, const char
                                  * key, struct xbc_node ** vnode)
    Find a value node which matches given key
```

**Parameters**

**struct xbc\_node \* parent** An XBC node.

**const char \* key** A key string.

**struct xbc\_node \*\* vnode** A container pointer of found XBC node.

**Description**

Search a value node under **parent** whose (parent) key node matches **key**, store it in **\*vnode**, and returns the value string. The **key** can contain several words jointed with **'.'**. If **parent** is NULL, this searches the node from whole tree. Return the value string if a matched key found, return NULL if no node is matched. Note that this returns 0-length string and stores NULL in **\*vnode** if the key has no value. And also it will return the value of the first entry if the value is an array.

```
int xbc_node_compose_key_after(struct xbc_node * root, struct xbc_node
                             * node, char * buf, size_t size)
```

Compose partial key string of the XBC node

**Parameters**

**struct xbc\_node \* root** Root XBC node

**struct xbc\_node \* node** Target XBC node.

**char \* buf** A buffer to store the key.

**size\_t size** The size of the **buf**.

**Description**

Compose the partial key of the **node** into **buf**, which is starting right after **root** (**root** is not included.) If **root** is NULL, this returns full key words of **node**. Returns the total length of the key stored in **buf**. Returns -EINVAL if **node** is NULL or **root** is not the ancestor of **node** or **root** is **node**, or returns -ERANGE if the key depth is deeper than max depth. This is expected to be used with `xbc_find_node()` to list up all (child) keys under given key.

```
struct xbc_node * xbc_node_find_next_leaf(struct xbc_node * root, struct
                                         xbc_node * node)
```

Find the next leaf node under given node

**Parameters**

**struct xbc\_node \* root** An XBC root node

**struct xbc\_node \* node** An XBC node which starts from.

**Description**

Search the next leaf node (which means the terminal key node) of **node** under **root** node (including **root** node itself). Return the next node or NULL if next leaf node is not found.

```
const char * xbc_node_find_next_key_value(struct xbc_node * root, struct
                                         xbc_node ** leaf)
```

Find the next key-value pair nodes

**Parameters**

**struct xbc\_node \* root** An XBC root node

**struct xbc\_node \*\* leaf** A container pointer of XBC node which starts from.

### Description

Search the next leaf node (which means the terminal key node) of **\*leaf** under **root** node. Returns the value and update **\*leaf** if next leaf node is found, or NULL if no next leaf node is found. Note that this returns 0-length string if the key has no value, or the value of the first entry if the value is an array.

```
void xbc_destroy_all(void)
    Clean up all parsed bootconfig
```

### Parameters

**void** no arguments

### Description

This clears all data structures of parsed bootconfig on memory. If you need to reuse `xbc_init()` with new boot config, you can use this.

```
int xbc_init(char * buf, const char ** emsg, int * epos)
    Parse given XBC file and build XBC internal tree
```

### Parameters

**char \* buf** boot config text

**const char \*\* emsg** A pointer of const char \* to store the error message

**int \* epos** A pointer of int to store the error position

### Description

This parses the boot config text in **buf**. **buf** must be a null terminated string and smaller than `XBC_DATA_MAX`. Return the number of stored nodes (>0) if succeeded, or `-errno` if there is any error. In error cases, **emsg** will be updated with an error message and **epos** will be updated with the error position which is the byte offset of **buf**. If the error is not a parser error, **epos** will be -1.

```
void xbc_debug_dump(void)
    Dump current XBC node list
```

### Parameters

**void** no arguments

### Description

Dump the current XBC node list on printk buffer for debug.

## **LINUX BRAILLE CONSOLE**

To get early boot messages on a braille device (before userspace screen readers can start), you first need to compile the support for the usual serial console (see [Documentation/admin-guide/serial-console.rst](#)), and for braille device (in [Device Drivers](#) → [Accessibility support](#) → [Console on braille device](#)).

Then you need to specify a `console=brl`, option on the kernel command line, the format is:

```
console=brl,serial_options...
```

where `serial_options...` are the same as described in [Documentation/admin-guide/serial-console.rst](#).

So for instance you can use `console=brl,ttyS0` if the braille device is connected to the first serial port, and `console=brl,ttyS0,115200` to override the baud rate to 115200, etc.

By default, the braille device will just show the last kernel message (console mode). To review previous messages, press the Insert key to switch to the VT review mode. In review mode, the arrow keys permit to browse in the VT content, PAGE-UP/PAGE-DOWN keys go at the top/bottom of the screen, and the HOME key goes back to the cursor, hence providing very basic screen reviewing facility.

Sound feedback can be obtained by adding the `braille_console.sound=1` kernel parameter.

For simplicity, only one braille console can be enabled, other uses of `console=brl,...` will be discarded. Also note that it does not interfere with the console selection mechanism described in [Documentation/admin-guide/serial-console.rst](#).

For now, only the VisioBraille device is supported.

Samuel Thibault <[samuel.thibault@ens-lyon.org](mailto:samuel.thibault@ens-lyon.org)>



## BTMRVL DRIVER

All commands are used via debugfs interface.

### 26.1 Set/get driver configurations

Path: /debug/btmrvl/config/

**gpiogap=[n], hscfgcmd** These commands are used to configure the host sleep parameters:: bit 8:0 - Gap bit 16:8 - GPIO

where GPIO is the pin number of GPIO used to wake up the host. It could be any valid GPIO pin# (e.g. 0-7) or 0xff (SDIO interface wakeup will be used instead).

where Gap is the gap in milli seconds between wakeup signal and wakeup event, or 0xff for special host sleep setting.

Usage:

```
# Use SDIO interface to wake up the host and set GAP to 0x80:
echo 0xff80 > /debug/btmrvl/config/gpiogap
echo 1 > /debug/btmrvl/config/hscfgcmd

# Use GPIO pin #3 to wake up the host and set GAP to 0xff:
echo 0x03ff > /debug/btmrvl/config/gpiogap
echo 1 > /debug/btmrvl/config/hscfgcmd
```

**psmode=[n], pscmd** These commands are used to enable/disable auto sleep mode

where the option is:

```
1      -- Enable auto sleep mode
0      -- Disable auto sleep mode
```

Usage:

```
# Enable auto sleep mode
echo 1 > /debug/btmrvl/config/psmode
echo 1 > /debug/btmrvl/config/pscmd

# Disable auto sleep mode
echo 0 > /debug/btmrvl/config/psmode
echo 1 > /debug/btmrvl/config/pscmd
```

**hsmode=[n], hscmd** These commands are used to enable host sleep or wake up firmware

where the option is:

```
1      -- Enable host sleep
0      -- Wake up firmware
```

Usage:

```
# Enable host sleep
echo 1 > /debug/btmrvl/config/hsmode
echo 1 > /debug/btmrvl/config/hscmd

# Wake up firmware
echo 0 > /debug/btmrvl/config/hsmode
echo 1 > /debug/btmrvl/config/hscmd
```

## 26.2 Get driver status

Path: /debug/btmrvl/status/

Usage:

```
cat /debug/btmrvl/status/<args>
```

where the args are:

**curpsmode** This command displays current auto sleep status.

**psstate** This command display the power save state.

**hsstate** This command display the host sleep state.

**txdnldrdy** This command displays the value of Tx download ready flag.

## 26.3 Issuing a raw hci command

Use hcitool to issue raw hci command, refer to hcitool manual

Usage:

```
Hcitool cmd <ogf> <ocf> [Parameters]
```

Interface Control Command:

```
hcitool cmd 0x3f 0x5b 0xf5 0x01 0x00 --Enable All interface
hcitool cmd 0x3f 0x5b 0xf5 0x01 0x01 --Enable Wlan interface
hcitool cmd 0x3f 0x5b 0xf5 0x01 0x02 --Enable BT interface
hcitool cmd 0x3f 0x5b 0xf5 0x00 0x00 --Disable All interface
hcitool cmd 0x3f 0x5b 0xf5 0x00 0x01 --Disable Wlan interface
hcitool cmd 0x3f 0x5b 0xf5 0x00 0x02 --Disable BT interface
```

## 26.4 SD8688 firmware

Images:

- `/lib/firmware/sd8688_helper.bin`
- `/lib/firmware/sd8688.bin`

The images can be downloaded from:

[git.infradead.org/users/dwmw2/linux-firmware.git/libertas/](https://git.infradead.org/users/dwmw2/linux-firmware.git/libertas/)



## **CONTROL GROUPS VERSION 1**

### **27.1 Control Groups**

Written by Paul Menage <[menage@google.com](mailto:menage@google.com)> based on Documentation/admin-guide/cgroup-v1/cpusets.rst

Original copyright statements from cpusets.txt:

Portions Copyright (C) 2004 BULL SA.

Portions Copyright (c) 2004-2006 Silicon Graphics, Inc.

Modified by Paul Jackson <[pj@sgi.com](mailto:pj@sgi.com)>

Modified by Christoph Lameter <[cl@linux.com](mailto:cl@linux.com)>

#### **27.1.1 1. Control Groups**

##### **1.1 What are cgroups ?**

Control Groups provide a mechanism for aggregating/partitioning sets of tasks, and all their future children, into hierarchical groups with specialized behaviour.

Definitions:

A cgroup associates a set of tasks with a set of parameters for one or more subsystems.

A subsystem is a module that makes use of the task grouping facilities provided by cgroups to treat groups of tasks in particular ways. A subsystem is typically a “resource controller” that schedules a resource or applies per-cgroup limits, but it may be anything that wants to act on a group of processes, e.g. a virtualization subsystem.

A hierarchy is a set of cgroups arranged in a tree, such that every task in the system is in exactly one of the cgroups in the hierarchy, and a set of subsystems; each subsystem has system-specific state attached to each cgroup in the hierarchy. Each hierarchy has an instance of the cgroup virtual filesystem associated with it.

At any one time there may be multiple active hierarchies of task cgroups. Each hierarchy is a partition of all tasks in the system.

User-level code may create and destroy cgroups by name in an instance of the cgroup virtual file system, specify and query to which cgroup a task is assigned,

and list the task PIDs assigned to a cgroup. Those creations and assignments only affect the hierarchy associated with that instance of the cgroup file system.

On their own, the only use for cgroups is for simple job tracking. The intention is that other subsystems hook into the generic cgroup support to provide new attributes for cgroups, such as accounting/limiting the resources which processes in a cgroup can access. For example, cpusets (see Documentation/admin-guide/cgroup-v1/cpusets.rst) allow you to associate a set of CPUs and a set of memory nodes with the tasks in each cgroup.

### 1.2 Why are cgroups needed ?

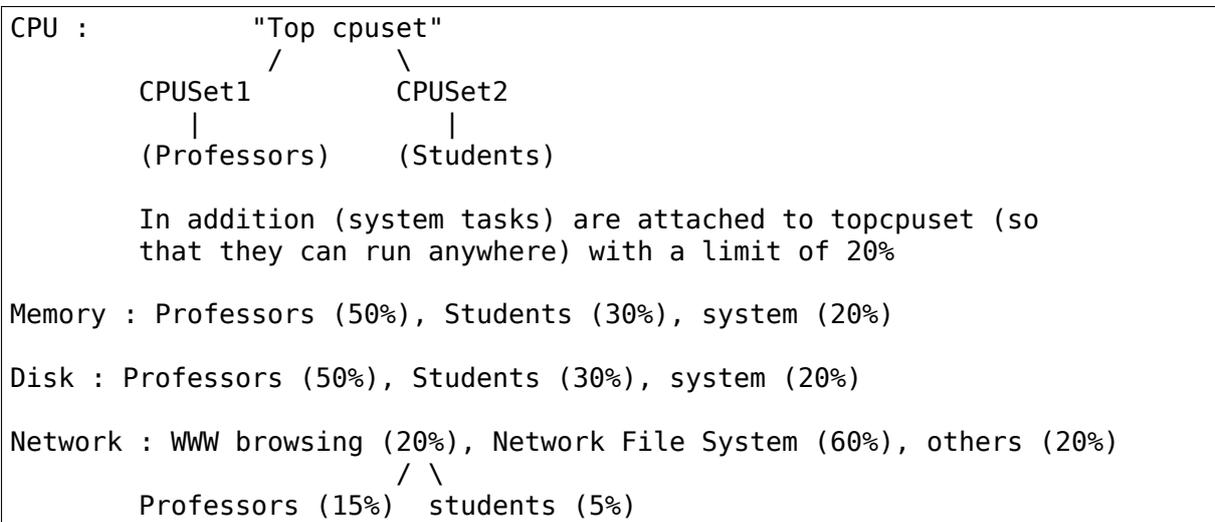
There are multiple efforts to provide process aggregations in the Linux kernel, mainly for resource-tracking purposes. Such efforts include cpusets, CKRM/ResGroups, UserBeanCounters, and virtual server namespaces. These all require the basic notion of a grouping/partitioning of processes, with newly forked processes ending up in the same group (cgroup) as their parent process.

The kernel cgroup patch provides the minimum essential kernel mechanisms required to efficiently implement such groups. It has minimal impact on the system fast paths, and provides hooks for specific subsystems such as cpusets to provide additional behaviour as desired.

Multiple hierarchy support is provided to allow for situations where the division of tasks into cgroups is distinctly different for different subsystems - having parallel hierarchies allows each hierarchy to be a natural division of tasks, without having to handle complex combinations of tasks that would be present if several unrelated subsystems needed to be forced into the same tree of cgroups.

At one extreme, each resource controller or subsystem could be in a separate hierarchy; at the other extreme, all subsystems would be attached to the same hierarchy.

As an example of a scenario (originally proposed by [vatsa@in.ibm.com](mailto:vatsa@in.ibm.com)) that can benefit from multiple hierarchies, consider a large university server with various users - students, professors, system tasks etc. The resource planning for this server could be along the following lines:



Browsers like Firefox/Lynx go into the WWW network class, while (k)nfsd goes into the NFS network class.

At the same time Firefox/Lynx will share an appropriate CPU/Memory class depending on who launched it (prof/student).

With the ability to classify tasks differently for different resources (by putting those resource subsystems in different hierarchies), the admin can easily set up a script which receives exec notifications and depending on who is launching the browser he can:

```
# echo browser_pid > /sys/fs/cgroup/<restype>/<userclass>/tasks
```

With only a single hierarchy, he now would potentially have to create a separate cgroup for every browser launched and associate it with appropriate network and other resource class. This may lead to proliferation of such cgroups.

Also let's say that the administrator would like to give enhanced network access temporarily to a student's browser (since it is night and the user wants to do online gaming :) ) OR give one of the student's simulation apps enhanced CPU power.

With ability to write PIDs directly to resource classes, it's just a matter of:

```
# echo pid > /sys/fs/cgroup/network/<new_class>/tasks  
(after some time)  
# echo pid > /sys/fs/cgroup/network/<orig_class>/tasks
```

Without this ability, the administrator would have to split the cgroup into multiple separate ones and then associate the new cgroups with the new resource classes.

### 1.3 How are cgroups implemented ?

Control Groups extends the kernel as follows:

- Each task in the system has a reference-counted pointer to a `css_set`.
- A `css_set` contains a set of reference-counted pointers to `cgroup_subsys_state` objects, one for each cgroup subsystem registered in the system. There is no direct link from a task to the cgroup of which it's a member in each hierarchy, but this can be determined by following pointers through the `cgroup_subsys_state` objects. This is because accessing the subsystem state is something that's expected to happen frequently and in performance-critical code, whereas operations that require a task's actual cgroup assignments (in particular, moving between cgroups) are less common. A linked list runs through the `cg_list` field of each `task_struct` using the `css_set`, anchored at `css_set->tasks`.
- A cgroup hierarchy filesystem can be mounted for browsing and manipulation from user space.
- You can list all the tasks (by PID) attached to any cgroup.

The implementation of cgroups requires a few, simple hooks into the rest of the kernel, none in performance-critical paths:

- in `init/main.c`, to initialize the root cgroups and initial `css_set` at system boot.

- in fork and exit, to attach and detach a task from its `css_set`.

In addition, a new file system of type “cgroup” may be mounted, to enable browsing and modifying the cgroups presently known to the kernel. When mounting a cgroup hierarchy, you may specify a comma-separated list of subsystems to mount as the filesystem mount options. By default, mounting the cgroup filesystem attempts to mount a hierarchy containing all registered subsystems.

If an active hierarchy with exactly the same set of subsystems already exists, it will be reused for the new mount. If no existing hierarchy matches, and any of the requested subsystems are in use in an existing hierarchy, the mount will fail with `-EBUSY`. Otherwise, a new hierarchy is activated, associated with the requested subsystems.

It’s not currently possible to bind a new subsystem to an active cgroup hierarchy, or to unbind a subsystem from an active cgroup hierarchy. This may be possible in future, but is fraught with nasty error-recovery issues.

When a cgroup filesystem is unmounted, if there are any child cgroups created below the top-level cgroup, that hierarchy will remain active even though unmounted; if there are no child cgroups then the hierarchy will be deactivated.

No new system calls are added for cgroups - all support for querying and modifying cgroups is via this cgroup file system.

Each task under `/proc` has an added file named ‘cgroup’ displaying, for each active hierarchy, the subsystem names and the cgroup name as the path relative to the root of the cgroup file system.

Each cgroup is represented by a directory in the cgroup file system containing the following files describing that cgroup:

- `tasks`: list of tasks (by PID) attached to that cgroup. This list is not guaranteed to be sorted. Writing a thread ID into this file moves the thread into this cgroup.
- `cgroup.procs`: list of thread group IDs in the cgroup. This list is not guaranteed to be sorted or free of duplicate TGIDs, and userspace should sort/uniquify the list if this property is required. Writing a thread group ID into this file moves all threads in that group into this cgroup.
- `notify_on_release` flag: run the release agent on exit?
- `release_agent`: the path to use for release notifications (this file exists in the top cgroup only)

Other subsystems such as `cpuset`s may add additional files in each cgroup dir.

New cgroups are created using the `mkdir` system call or shell command. The properties of a cgroup, such as its flags, are modified by writing to the appropriate file in that cgroups directory, as listed above.

The named hierarchical structure of nested cgroups allows partitioning a large system into nested, dynamically changeable, “soft-partitions” .

The attachment of each task, automatically inherited at fork by any children of that task, to a cgroup allows organizing the work load on a system into related sets of tasks. A task may be re-attached to any other cgroup, if allowed by the permissions on the necessary cgroup file system directories.

When a task is moved from one cgroup to another, it gets a new `css_set` pointer - if there's an already existing `css_set` with the desired collection of cgroups then that group is reused, otherwise a new `css_set` is allocated. The appropriate existing `css_set` is located by looking into a hash table.

To allow access from a cgroup to the `css_sets` (and hence tasks) that comprise it, a set of `cg_cgroup_link` objects form a lattice; each `cg_cgroup_link` is linked into a list of `cg_cgroup_links` for a single cgroup on its `cgrp_link_list` field, and a list of `cg_cgroup_links` for a single `css_set` on its `cg_link_list`.

Thus the set of tasks in a cgroup can be listed by iterating over each `css_set` that references the cgroup, and sub-iterating over each `css_set`'s task set.

The use of a Linux virtual file system (vfs) to represent the cgroup hierarchy provides for a familiar permission and name space for cgroups, with a minimum of additional kernel code.

### 1.4 What does `notify_on_release` do ?

If the `notify_on_release` flag is enabled (1) in a cgroup, then whenever the last task in the cgroup leaves (exits or attaches to some other cgroup) and the last child cgroup of that cgroup is removed, then the kernel runs the command specified by the contents of the “`release_agent`” file in that hierarchy's root directory, supplying the pathname (relative to the mount point of the cgroup file system) of the abandoned cgroup. This enables automatic removal of abandoned cgroups. The default value of `notify_on_release` in the root cgroup at system boot is disabled (0). The default value of other cgroups at creation is the current value of their parents' `notify_on_release` settings. The default value of a cgroup hierarchy's `release_agent` path is empty.

### 1.5 What does `clone_children` do ?

This flag only affects the cgroup controller. If the `clone_children` flag is enabled (1) in a cgroup, a new cgroup will copy its configuration from the parent during initialization.

### 1.6 How do I use cgroups ?

To start a new job that is to be contained within a cgroup, using the “cgroup” cgroup subsystem, the steps are something like:

- 1) `mount -t tmpfs cgroup_root /sys/fs/cgroup`
- 2) `mkdir /sys/fs/cgroup/cpuset`
- 3) `mount -t cgroup -ocpuset cpuset /sys/fs/cgroup/cpuset`
- 4) Create the new cgroup by doing `mkdir`'s and `write`'s (or `echo`'s) in the `/sys/fs/cgroup/cpuset` virtual file system.
- 5) Start a task that will be the “founding father” of the new job.
- 6) Attach that task to the new cgroup by writing its PID to the `/sys/fs/cgroup/cpuset/tasks` file for that cgroup.
- 7) `fork`, `exec` or `clone` the job tasks from this founding father task.

For example, the following sequence of commands will setup a cgroup named “Charlie” , containing just CPUs 2 and 3, and Memory Node 1, and then start a subshell ‘sh’ in that cgroup:

```
mount -t tmpfs cgroup_root /sys/fs/cgroup
mkdir /sys/fs/cgroup/cpuset
mount -t cgroup cpuset -ocpuset /sys/fs/cgroup/cpuset
cd /sys/fs/cgroup/cpuset
mkdir Charlie
cd Charlie
/bin/echo 2-3 > cpuset.cpus
/bin/echo 1 > cpuset.mems
/bin/echo $$ > tasks
sh
# The subshell 'sh' is now running in cgroup Charlie
# The next line should display '/Charlie'
cat /proc/self/cgroup
```

### 27.1.2 2. Usage Examples and Syntax

#### 2.1 Basic Usage

Creating, modifying, using cgroups can be done through the cgroup virtual filesystem.

To mount a cgroup hierarchy with all available subsystems, type:

```
# mount -t cgroup xxx /sys/fs/cgroup
```

The “xxx” is not interpreted by the cgroup code, but will appear in `/proc/mounts` so may be any useful identifying string that you like.

Note: Some subsystems do not work without some user input first. For instance, if cpusets are enabled the user will have to populate the cpus and mems files for each new cgroup created before that group can be used.

As explained in section 1.2 Why are cgroups needed? you should create different hierarchies of cgroups for each single resource or group of resources you want to control. Therefore, you should mount a tmpfs on `/sys/fs/cgroup` and create directories for each cgroup resource or resource group:

```
# mount -t tmpfs cgroup_root /sys/fs/cgroup
# mkdir /sys/fs/cgroup/rg1
```

To mount a cgroup hierarchy with just the cpuset and memory subsystems, type:

```
# mount -t cgroup -o cpuset,memory hier1 /sys/fs/cgroup/rg1
```

While remounting cgroups is currently supported, it is not recommend to use it. Remounting allows changing bound subsystems and `release_agent`. Rebinding is hardly useful as it only works when the hierarchy is empty and `release_agent` itself should be replaced with conventional `fsnotify`. The support for remounting will be removed in the future.

To Specify a hierarchy’ s `release_agent`:

```
# mount -t cgroup -o cpuset,release_agent="/sbin/cpuset_release_agent" \
xxx /sys/fs/cgroup/rg1
```

Note that specifying ‘release\_agent’ more than once will return failure.

Note that changing the set of subsystems is currently only supported when the hierarchy consists of a single (root) cgroup. Supporting the ability to arbitrarily bind/unbind subsystems from an existing cgroup hierarchy is intended to be implemented in the future.

Then under /sys/fs/cgroup/rg1 you can find a tree that corresponds to the tree of the cgroups in the system. For instance, /sys/fs/cgroup/rg1 is the cgroup that holds the whole system.

If you want to change the value of release\_agent:

```
# echo "/sbin/new_release_agent" > /sys/fs/cgroup/rg1/release_agent
```

It can also be changed via remount.

If you want to create a new cgroup under /sys/fs/cgroup/rg1:

```
# cd /sys/fs/cgroup/rg1
# mkdir my_cgroup
```

Now you want to do something with this cgroup:

```
# cd my_cgroup
```

In this directory you can find several files:

```
# ls
cgroup.procs notify_on_release tasks
(plus whatever files added by the attached subsystems)
```

Now attach your shell to this cgroup:

```
# /bin/echo $$ > tasks
```

You can also create cgroups inside your cgroup by using mkdir in this directory:

```
# mkdir my_sub_cs
```

To remove a cgroup, just use rmdir:

```
# rmdir my_sub_cs
```

This will fail if the cgroup is in use (has cgroups inside, or has processes attached, or is held alive by other subsystem-specific reference).

### 2.2 Attaching processes

```
# /bin/echo PID > tasks
```

Note that it is PID, not PIDs. You can only attach ONE task at a time. If you have several tasks to attach, you have to do it one after another:

```
# /bin/echo PID1 > tasks
# /bin/echo PID2 > tasks
...
# /bin/echo PIDn > tasks
```

You can attach the current shell task by echoing 0:

```
# echo 0 > tasks
```

You can use the `cgroup.procs` file instead of the `tasks` file to move all threads in a threadgroup at once. Echoing the PID of any task in a threadgroup to `cgroup.procs` causes all tasks in that threadgroup to be attached to the cgroup. Writing 0 to `cgroup.procs` moves all tasks in the writing task's threadgroup.

Note: Since every task is always a member of exactly one cgroup in each mounted hierarchy, to remove a task from its current cgroup you must move it into a new cgroup (possibly the root cgroup) by writing to the new cgroup's `tasks` file.

Note: Due to some restrictions enforced by some cgroup subsystems, moving a process to another cgroup can fail.

### 2.3 Mounting hierarchies by name

Passing the `name=<x>` option when mounting a cgroups hierarchy associates the given name with the hierarchy. This can be used when mounting a pre-existing hierarchy, in order to refer to it by name rather than by its set of active subsystems. Each hierarchy is either nameless, or has a unique name.

The name should match `[w.-]+`

When passing a `name=<x>` option for a new hierarchy, you need to specify subsystems manually; the legacy behaviour of mounting all subsystems when none are explicitly specified is not supported when you give a subsystem a name.

The name of the subsystem appears as part of the hierarchy description in `/proc/mounts` and `/proc/<pid>/cgroups`.

## 27.1.3 3. Kernel API

### 3.1 Overview

Each kernel subsystem that wants to hook into the generic cgroup system needs to create a `cgroup_subsys` object. This contains various methods, which are callbacks from the cgroup system, along with a subsystem ID which will be assigned by the cgroup system.

Other fields in the `cgroup_subsys` object include:

- `subsys_id`: a unique array index for the subsystem, indicating which entry in `cgroup->subsys[]` this subsystem should be managing.
- `name`: should be initialized to a unique subsystem name. Should be no longer than `MAX_CGROUP_TYPE_NAMELEN`.
- `early_init`: indicate if the subsystem needs early initialization at system boot.

Each cgroup object created by the system has an array of pointers, indexed by subsystem ID; this pointer is entirely managed by the subsystem; the generic cgroup code will never touch this pointer.

### 3.2 Synchronization

There is a global mutex, `cgroup_mutex`, used by the cgroup system. This should be taken by anything that wants to modify a cgroup. It may also be taken to prevent cgroups from being modified, but more specific locks may be more appropriate in that situation.

See `kernel/cgroup.c` for more details.

Subsystems can take/release the `cgroup_mutex` via the functions `cgroup_lock()/cgroup_unlock()`.

Accessing a task's cgroup pointer may be done in the following ways: - while holding `cgroup_mutex` - while holding the task's `alloc_lock` (via `task_lock()`) - inside an `rcu_read_lock()` section via `rcu_dereference()`

### 3.3 Subsystem API

Each subsystem should:

- add an entry in `linux/cgroup_subsys.h`
- define a `cgroup_subsys` object called `<name>_cgrp_subsys`

Each subsystem may export the following methods. The only mandatory methods are `css_alloc/free`. Any others that are null are presumed to be successful no-ops.

```
struct cgroup_subsys_state *css_alloc(struct cgroup *cgrp)
(cgroup_mutex held by caller)
```

Called to allocate a subsystem state object for a cgroup. The subsystem should allocate its subsystem state object for the passed cgroup, returning a pointer to the new object on success or a `ERR_PTR()` value. On success, the subsystem pointer should point to a structure of type `cgroup_subsys_state` (typically embedded in a larger subsystem-specific object), which will be initialized by the cgroup system. Note that this will be called at initialization to create the root subsystem state for this subsystem; this case can be identified by the passed cgroup object having a NULL parent (since it's the root of the hierarchy) and may be an appropriate place for initialization code.

```
int css_online(struct cgroup *cgrp) (cgroup_mutex held by caller)
```

Called after `@cgrp` successfully completed all allocations and made visible to `cgroup_for_each_child/descendant_*`() iterators. The subsystem may choose to

fail creation by returning `-errno`. This callback can be used to implement reliable state sharing and propagation along the hierarchy. See the comment on `cgroup_for_each_descendant_pre()` for details.

```
void css_offline(struct cgroup *cgrp); (cgroup_mutex held by caller)
```

This is the counterpart of `css_online()` and called iff `css_online()` has succeeded on `@cgrp`. This signifies the beginning of the end of `@cgrp`. `@cgrp` is being removed and the subsystem should start dropping all references it's holding on `@cgrp`. When all references are dropped, `cgroup` removal will proceed to the next step - `css_free()`. After this callback, `@cgrp` should be considered dead to the subsystem.

```
void css_free(struct cgroup *cgrp) (cgroup_mutex held by caller)
```

The `cgroup` system is about to free `@cgrp`; the subsystem should free its subsystem state object. By the time this method is called, `@cgrp` is completely unused; `@cgrp->parent` is still valid. (Note - can also be called for a newly-created `cgroup` if an error occurs after this subsystem's `create()` method has been called for the new `cgroup`).

```
int can_attach(struct cgroup *cgrp, struct cgroup_taskset *tset)
(cgroup_mutex held by caller)
```

Called prior to moving one or more tasks into a `cgroup`; if the subsystem returns an error, this will abort the attach operation. `@tset` contains the tasks to be attached and is guaranteed to have at least one task in it.

### If there are multiple tasks in the taskset, then:

- it's guaranteed that all are from the same thread group
- `@tset` contains all tasks from the thread group whether or not they're switching `cgroups`
- the first task is the leader

Each `@tset` entry also contains the task's old `cgroup` and tasks which aren't switching `cgroup` can be skipped easily using the `cgroup_taskset_for_each()` iterator. Note that this isn't called on a fork. If this method returns 0 (success) then this should remain valid while the caller holds `cgroup_mutex` and it is ensured that either `attach()` or `cancel_attach()` will be called in future.

```
void css_reset(struct cgroup_subsys_state *css) (cgroup_mutex held by caller)
```

An optional operation which should restore `@css`'s configuration to the initial state. This is currently only used on the unified hierarchy when a subsystem is disabled on a `cgroup` through "cgroup.subtree\_control" but should remain enabled because other subsystems depend on it. `cgroup` core makes such a `css` invisible by removing the associated interface files and invokes this callback so that the hidden subsystem can return to the initial neutral state. This prevents unexpected resource control from a hidden `css` and ensures that the configuration is in the initial state when it is made visible again later.

```
void cancel_attach(struct cgroup *cgrp, struct cgroup_taskset *tset)
(cgroup_mutex held by caller)
```

Called when a task attach operation has failed after `can_attach()` has succeeded. A subsystem whose `can_attach()` has some side-effects should provide this function,

so that the subsystem can implement a rollback. If not, not necessary. This will be called only about subsystems whose `can_attach()` operation have succeeded. The parameters are identical to `can_attach()`.

```
void attach(struct cgroup *cgrp, struct cgroup_taskset *tset)
(cgroup_mutex held by caller)
```

Called after the task has been attached to the cgroup, to allow any post-attachment activity that requires memory allocations or blocking. The parameters are identical to `can_attach()`.

```
void fork(struct task_struct *task)
```

Called when a task is forked into a cgroup.

```
void exit(struct task_struct *task)
```

Called during task exit.

```
void free(struct task_struct *task)
```

Called when the `task_struct` is freed.

```
void bind(struct cgroup *root) (cgroup_mutex held by caller)
```

Called when a cgroup subsystem is rebound to a different hierarchy and root cgroup. Currently this will only involve movement between the default hierarchy (which never has sub-cgroups) and a hierarchy that is being created/destroyed (and hence has no sub-cgroups).

#### **27.1.4 4. Extended attribute usage**

cgroup filesystem supports certain types of extended attributes in its directories and files. The current supported types are:

- Trusted (XATTR\_TRUSTED)
- Security (XATTR\_SECURITY)

Both require `CAP_SYS_ADMIN` capability to set.

Like in tmpfs, the extended attributes in cgroup filesystem are stored using kernel memory and it's advised to keep the usage at minimum. This is the reason why user defined extended attributes are not supported, since any user can do it and there's no limit in the value size.

The current known users for this feature are SELinux to limit cgroup usage in containers and systemd for assorted meta data like main PID in a cgroup (systemd creates a cgroup per service).

### 27.1.5 5. Questions

```
Q: what's up with this '/bin/echo' ?
A: bash's builtin 'echo' command does not check calls to write() against
   errors. If you use it in the cgroup file system, you won't be
   able to tell whether a command succeeded or failed.

Q: When I attach processes, only the first of the line gets really
   ↳attached !
A: We can only return one error code per call to write(). So you should
   ↳also
   put only ONE PID.
```

## 27.2 Block IO Controller

### 27.2.1 Overview

cgroup subsys “blkio” implements the block io controller. There seems to be a need of various kinds of IO control policies (like proportional BW, max BW) both at leaf nodes as well as at intermediate nodes in a storage hierarchy. Plan is to use the same cgroup based management interface for blkio controller and based on user options switch IO policies in the background.

One IO control policy is throttling policy which can be used to specify upper IO rate limits on devices. This policy is implemented in generic block layer and can be used on leaf nodes as well as higher level logical devices like device mapper.

### 27.2.2 HOWTO

#### Throttling/Upper Limit policy

- Enable Block IO controller:

```
CONFIG_BLK_CGROUP=y
```

- Enable throttling in block layer:

```
CONFIG_BLK_DEV_THROTTLING=y
```

- Mount blkio controller (see cgroups.txt, Why are cgroups needed?):

```
mount -t cgroup -o blkio none /sys/fs/cgroup/blkio
```

- Specify a bandwidth rate on particular device for root group. The format for policy is “<major>:<minor> <bytes\_per\_second>” :

```
echo "8:16 1048576" > /sys/fs/cgroup/blkio/blkio.throttle.read_bps_
   ↳device
```

Above will put a limit of 1MB/second on reads happening for root group on device having major/minor number 8:16.

- Run dd to read a file and see if rate is throttled to 1MB/s or not:

```
# dd iflag=direct if=/mnt/common/zerofile of=/dev/null bs=4K
↪count=1024
1024+0 records in
1024+0 records out
4194304 bytes (4.2 MB) copied, 4.0001 s, 1.0 MB/s
```

Limits for writes can be put using `blkio.throttle.write_bps_device` file.

### 27.2.3 Hierarchical Cgroups

Throttling implements hierarchy support; however, throttling's hierarchy support is enabled iff "sane\_behavior" is enabled from cgroup side, which currently is a development option and not publicly available.

If somebody created a hierarchy like as follows:

```
  root
  /  \
test1 test2
  |
test3
```

Throttling with "sane\_behavior" will handle the hierarchy correctly. For throttling, all limits apply to the whole subtree while all statistics are local to the IOs directly generated by tasks in that cgroup.

Throttling without "sane\_behavior" enabled from cgroup side will practically treat all groups at same level as if it looks like the following:

```
      pivot
     /  /  \  \
root test1 test2 test3
```

### 27.2.4 Various user visible config options

#### CONFIG\_BLK\_CGROUP

- Block IO controller.

#### CONFIG\_BFQ\_CGROUP\_DEBUG

- Debug help. Right now some additional stats file show up in cgroup if this option is enabled.

#### CONFIG\_BLK\_DEV\_THROTTLING

- Enable block device throttling support in block layer.

## 27.2.5 Details of cgroup files

### Proportional weight policy files

- **blkio.weight**

- Specifies per cgroup weight. This is default weight of the group on all the devices until and unless overridden by per device rule. (See `blkio.weight_device`). Currently allowed range of weights is from 10 to 1000.

- **blkio.weight\_device**

- One can specify per cgroup per device rules using this interface. These rules override the default value of group weight as specified by `blkio.weight`.

Following is the format:

```
# echo dev_maj:dev_minor weight > blkio.weight_device
```

Configure weight=300 on /dev/sdb (8:16) in this cgroup:

```
# echo 8:16 300 > blkio.weight_device
# cat blkio.weight_device
dev    weight
8:16   300
```

Configure weight=500 on /dev/sda (8:0) in this cgroup:

```
# echo 8:0 500 > blkio.weight_device
# cat blkio.weight_device
dev    weight
8:0    500
8:16   300
```

Remove specific weight for /dev/sda in this cgroup:

```
# echo 8:0 0 > blkio.weight_device
# cat blkio.weight_device
dev    weight
8:16   300
```

- **blkio.time**

- disk time allocated to cgroup per device in milliseconds. First two fields specify the major and minor number of the device and third field specifies the disk time allocated to group in milliseconds.

- **blkio.sectors**

- number of sectors transferred to/from disk by the group. First two fields specify the major and minor number of the device and third field specifies the number of sectors transferred by the group to/from the device.

- **blkio.io\_service\_bytes**

- Number of bytes transferred to/from the disk by the group. These are further divided by the type of operation - read or write, sync or async. First two fields specify the major and minor number of the device, third field specifies the operation type and the fourth field specifies the number of bytes.

- **blkio.io\_serviced**

- Number of IOs (bio) issued to the disk by the group. These are further divided by the type of operation - read or write, sync or async. First two fields specify the major and minor number of the device, third field specifies the operation type and the fourth field specifies the number of IOs.

- **blkio.io\_service\_time**

- Total amount of time between request dispatch and request completion for the IOs done by this cgroup. This is in nanoseconds to make it meaningful for flash devices too. For devices with queue depth of 1, this time represents the actual service time. When queue\_depth > 1, that is no longer true as requests may be served out of order. This may cause the service time for a given IO to include the service time of multiple IOs when served out of order which may result in total io\_service\_time > actual time elapsed. This time is further divided by the type of operation - read or write, sync or async. First two fields specify the major and minor number of the device, third field specifies the operation type and the fourth field specifies the io\_service\_time in ns.

- **blkio.io\_wait\_time**

- Total amount of time the IOs for this cgroup spent waiting in the scheduler queues for service. This can be greater than the total time elapsed since it is cumulative io\_wait\_time for all IOs. It is not a measure of total time the cgroup spent waiting but rather a measure of the wait\_time for its individual IOs. For devices with queue\_depth > 1 this metric does not include the time spent waiting for service once the IO is dispatched to the device but till it actually gets serviced (there might be a time lag here due to re-ordering of requests by the device). This is in nanoseconds to make it meaningful for flash devices too. This time is further divided by the type of operation - read or write, sync or async. First two fields specify the major and minor number of the device, third field specifies the operation type and the fourth field specifies the io\_wait\_time in ns.

- **blkio.io\_merged**

- Total number of bios/requests merged into requests belonging to this cgroup. This is further divided by the type of operation - read or write, sync or async.

- **blkio.io\_queued**

- Total number of requests queued up at any given instant for this cgroup. This is further divided by the type of operation - read or write, sync or async.

- **blkio.avg\_queue\_size**
  - Debugging aid only enabled if CONFIG\_BFQ\_CGROUP\_DEBUG=y. The average queue size for this cgroup over the entire time of this cgroup's existence. Queue size samples are taken each time one of the queues of this cgroup gets a timeslice.
- **blkio.group\_wait\_time**
  - Debugging aid only enabled if CONFIG\_BFQ\_CGROUP\_DEBUG=y. This is the amount of time the cgroup had to wait since it became busy (i.e., went from 0 to 1 request queued) to get a timeslice for one of its queues. This is different from the io\_wait\_time which is the cumulative total of the amount of time spent by each IO in that cgroup waiting in the scheduler queue. This is in nanoseconds. If this is read when the cgroup is in a waiting (for timeslice) state, the stat will only report the group\_wait\_time accumulated till the last time it got a timeslice and will not include the current delta.
- **blkio.empty\_time**
  - Debugging aid only enabled if CONFIG\_BFQ\_CGROUP\_DEBUG=y. This is the amount of time a cgroup spends without any pending requests when not being served, i.e., it does not include any time spent idling for one of the queues of the cgroup. This is in nanoseconds. If this is read when the cgroup is in an empty state, the stat will only report the empty\_time accumulated till the last time it had a pending request and will not include the current delta.
- **blkio.idle\_time**
  - Debugging aid only enabled if CONFIG\_BFQ\_CGROUP\_DEBUG=y. This is the amount of time spent by the IO scheduler idling for a given cgroup in anticipation of a better request than the existing ones from other queues/cgroups. This is in nanoseconds. If this is read when the cgroup is in an idling state, the stat will only report the idle\_time accumulated till the last idle period and will not include the current delta.
- **blkio.dequeue**
  - Debugging aid only enabled if CONFIG\_BFQ\_CGROUP\_DEBUG=y. This gives the statistics about how many a times a group was dequeued from service tree of the device. First two fields specify the major and minor number of the device and third field specifies the number of times a group was dequeued from a particular device.
- **blkio.\*\_recursive**
  - Recursive version of various stats. These files show the same information as their non-recursive counterparts but include stats from all the descendant cgroups.

## Throttling/Upper limit policy files

- **blkio.throttle.read\_bps\_device**

- Specifies upper limit on READ rate from the device. IO rate is specified in bytes per second. Rules are per device. Following is the format:

```
echo "<major>:<minor> <rate_bytes_per_second>" > /cgrp/blkio.  
→throttle.read_bps_device
```

- **blkio.throttle.write\_bps\_device**

- Specifies upper limit on WRITE rate to the device. IO rate is specified in bytes per second. Rules are per device. Following is the format:

```
echo "<major>:<minor> <rate_bytes_per_second>" > /cgrp/blkio.  
→throttle.write_bps_device
```

- **blkio.throttle.read\_iops\_device**

- Specifies upper limit on READ rate from the device. IO rate is specified in IO per second. Rules are per device. Following is the format:

```
echo "<major>:<minor> <rate_io_per_second>" > /cgrp/blkio.  
→throttle.read_iops_device
```

- **blkio.throttle.write\_iops\_device**

- Specifies upper limit on WRITE rate to the device. IO rate is specified in io per second. Rules are per device. Following is the format:

```
echo "<major>:<minor> <rate_io_per_second>" > /cgrp/blkio.  
→throttle.write_iops_device
```

**Note:** If both BW and IOPS rules are specified for a device, then IO is subjected to both the constraints.

- **blkio.throttle.io\_serviced**

- Number of IOs (bio) issued to the disk by the group. These are further divided by the type of operation - read or write, sync or async. First two fields specify the major and minor number of the device, third field specifies the operation type and the fourth field specifies the number of IOs.

- **blkio.throttle.io\_service\_bytes**

- Number of bytes transferred to/from the disk by the group. These are further divided by the type of operation - read or write, sync or async. First two fields specify the major and minor number of the device, third field specifies the operation type and the fourth field specifies the number of bytes.

### Common files among various policies

- **blkio.reset\_stats**

- Writing an int to this file will result in resetting all the stats for that cgroup.

## 27.3 CPU Accounting Controller

The CPU accounting controller is used to group tasks using cgroups and account the CPU usage of these groups of tasks.

The CPU accounting controller supports multi-hierarchy groups. An accounting group accumulates the CPU usage of all of its child groups and the tasks directly present in its group.

Accounting groups can be created by first mounting the cgroup filesystem:

```
# mount -t cgroup -ocpuacct none /sys/fs/cgroup
```

With the above step, the initial or the parent accounting group becomes visible at `/sys/fs/cgroup`. At bootup, this group includes all the tasks in the system. `/sys/fs/cgroup/tasks` lists the tasks in this cgroup. `/sys/fs/cgroup/cpuacct.usage` gives the CPU time (in nanoseconds) obtained by this group which is essentially the CPU time obtained by all the tasks in the system.

New accounting groups can be created under the parent group `/sys/fs/cgroup`:

```
# cd /sys/fs/cgroup
# mkdir g1
# echo $$ > g1/tasks
```

The above steps create a new group `g1` and move the current shell process (bash) into it. CPU time consumed by this bash and its children can be obtained from `g1/cpuacct.usage` and the same is accumulated in `/sys/fs/cgroup/cpuacct.usage` also.

`cpuacct.stat` file lists a few statistics which further divide the CPU time obtained by the cgroup into user and system times. Currently the following statistics are supported:

user: Time spent by tasks of the cgroup in user mode. system: Time spent by tasks of the cgroup in kernel mode.

user and system are in `USER_HZ` unit.

cpuacct controller uses `percpu_counter` interface to collect user and system times. This has two side effects:

- It is theoretically possible to see wrong values for user and system times. This is because `percpu_counter_read()` on 32bit systems isn't safe against concurrent writes.
- It is possible to see slightly outdated values for user and system times due to the batch processing nature of `percpu_counter`.

## 27.4 CPUSSETS

Copyright (C) 2004 BULL SA.

Written by [Simon.Derr@bull.net](mailto:Simon.Derr@bull.net)

- Portions Copyright (c) 2004-2006 Silicon Graphics, Inc.
- Modified by Paul Jackson <[pj@sgi.com](mailto:pj@sgi.com)>
- Modified by Christoph Lameter <[cl@linux.com](mailto:cl@linux.com)>
- Modified by Paul Menage <[menage@google.com](mailto:menage@google.com)>
- Modified by Hidetoshi Seto <[seto.hidetoshi@jp.fujitsu.com](mailto:seto.hidetoshi@jp.fujitsu.com)>

### 27.4.1 1. Cpusets

#### 1.1 What are cpusets ?

Cpusets provide a mechanism for assigning a set of CPUs and Memory Nodes to a set of tasks. In this document “Memory Node” refers to an on-line node that contains memory.

Cpusets constrain the CPU and Memory placement of tasks to only the resources within a task’s current cpuset. They form a nested hierarchy visible in a virtual file system. These are the essential hooks, beyond what is already present, required to manage dynamic job placement on large systems.

Cpusets use the generic cgroup subsystem described in Documentation/admin-guide/cgroup-v1/cgroups.rst.

Requests by a task, using the `sched_setaffinity(2)` system call to include CPUs in its CPU affinity mask, and using the `mbind(2)` and `set_mempolicy(2)` system calls to include Memory Nodes in its memory policy, are both filtered through that task’s cpuset, filtering out any CPUs or Memory Nodes not in that cpuset. The scheduler will not schedule a task on a CPU that is not allowed in its `cpus_allowed` vector, and the kernel page allocator will not allocate a page on a node that is not allowed in the requesting task’s `mems_allowed` vector.

User level code may create and destroy cpusets by name in the cgroup virtual file system, manage the attributes and permissions of these cpusets and which CPUs and Memory Nodes are assigned to each cpuset, specify and query to which cpuset a task is assigned, and list the task pids assigned to a cpuset.

### 1.2 Why are cpusets needed ?

The management of large computer systems, with many processors (CPUs), complex memory cache hierarchies and multiple Memory Nodes having non-uniform access times (NUMA) presents additional challenges for the efficient scheduling and memory placement of processes.

Frequently more modest sized systems can be operated with adequate efficiency just by letting the operating system automatically share the available CPU and Memory resources amongst the requesting tasks.

But larger systems, which benefit more from careful processor and memory placement to reduce memory access times and contention, and which typically represent a larger investment for the customer, can benefit from explicitly placing jobs on properly sized subsets of the system.

This can be especially valuable on:

- Web Servers running multiple instances of the same web application,
- Servers running different applications (for instance, a web server and a database), or
- NUMA systems running large HPC applications with demanding performance characteristics.

These subsets, or “soft partitions” must be able to be dynamically adjusted, as the job mix changes, without impacting other concurrently executing jobs. The location of the running jobs pages may also be moved when the memory locations are changed.

The kernel cpuset patch provides the minimum essential kernel mechanisms required to efficiently implement such subsets. It leverages existing CPU and Memory Placement facilities in the Linux kernel to avoid any additional impact on the critical scheduler or memory allocator code.

### 1.3 How are cpusets implemented ?

Cpusets provide a Linux kernel mechanism to constrain which CPUs and Memory Nodes are used by a process or set of processes.

The Linux kernel already has a pair of mechanisms to specify on which CPUs a task may be scheduled (`sched_setaffinity`) and on which Memory Nodes it may obtain memory (`mbind`, `set_mempolicy`).

Cpusets extends these two mechanisms as follows:

- Cpusets are sets of allowed CPUs and Memory Nodes, known to the kernel.
- Each task in the system is attached to a cpuset, via a pointer in the task structure to a reference counted cgroup structure.
- Calls to `sched_setaffinity` are filtered to just those CPUs allowed in that task's cpuset.
- Calls to `mbind` and `set_mempolicy` are filtered to just those Memory Nodes allowed in that task's cpuset.

- The root cpuset contains all the systems CPUs and Memory Nodes.
- For any cpuset, one can define child cpusets containing a subset of the parents CPU and Memory Node resources.
- The hierarchy of cpusets can be mounted at /dev/cpuset, for browsing and manipulation from user space.
- A cpuset may be marked exclusive, which ensures that no other cpuset (except direct ancestors and descendants) may contain any overlapping CPUs or Memory Nodes.
- You can list all the tasks (by pid) attached to any cpuset.

The implementation of cpusets requires a few, simple hooks into the rest of the kernel, none in performance critical paths:

- in init/main.c, to initialize the root cpuset at system boot.
- in fork and exit, to attach and detach a task from its cpuset.
- in sched\_setaffinity, to mask the requested CPUs by what's allowed in that task's cpuset.
- in sched.c migrate\_live\_tasks(), to keep migrating tasks within the CPUs allowed by their cpuset, if possible.
- in the mbind and set\_mempolicy system calls, to mask the requested Memory Nodes by what's allowed in that task's cpuset.
- in page\_alloc.c, to restrict memory to allowed nodes.
- in vmscan.c, to restrict page recovery to the current cpuset.

You should mount the “cgroup” filesystem type in order to enable browsing and modifying the cpusets presently known to the kernel. No new system calls are added for cpusets - all support for querying and modifying cpusets is via this cpuset file system.

The /proc/<pid>/status file for each task has four added lines, displaying the task's cpus\_allowed (on which CPUs it may be scheduled) and mems\_allowed (on which Memory Nodes it may obtain memory), in the two formats seen in the following example:

```
Cpus_allowed:    ffffffff, ffffffff, ffffffff, ffffffff
Cpus_allowed_list:  0-127
Mems_allowed:    ffffffff, ffffffff
Mems_allowed_list:  0-63
```

Each cpuset is represented by a directory in the cgroup file system containing (on top of the standard cgroup files) the following files describing that cpuset:

- cpuset.cpus: list of CPUs in that cpuset
- cpuset.mems: list of Memory Nodes in that cpuset
- cpuset.memory\_migrate flag: if set, move pages to cpusets nodes
- cpuset.cpu\_exclusive flag: is cpu placement exclusive?
- cpuset.mem\_exclusive flag: is memory placement exclusive?

- `cpuset.mem_hardwall` flag: is memory allocation hardwalled
- `cpuset.memory_pressure`: measure of how much paging pressure in cpuset
- `cpuset.memory_spread_page` flag: if set, spread page cache evenly on allowed nodes
- `cpuset.memory_spread_slab` flag: if set, spread slab cache evenly on allowed nodes
- `cpuset.sched_load_balance` flag: if set, load balance within CPUs on that cpuset
- `cpuset.sched_relax_domain_level`: the searching range when migrating tasks

In addition, only the root cpuset has the following file:

- `cpuset.memory_pressure_enabled` flag: compute memory\_pressure?

New cpusets are created using the `mkdir` system call or shell command. The properties of a cpuset, such as its flags, allowed CPUs and Memory Nodes, and attached tasks, are modified by writing to the appropriate file in that cpusets directory, as listed above.

The named hierarchical structure of nested cpusets allows partitioning a large system into nested, dynamically changeable, “soft-partitions” .

The attachment of each task, automatically inherited at fork by any children of that task, to a cpuset allows organizing the work load on a system into related sets of tasks such that each set is constrained to using the CPUs and Memory Nodes of a particular cpuset. A task may be re-attached to any other cpuset, if allowed by the permissions on the necessary cpuset file system directories.

Such management of a system “in the large” integrates smoothly with the detailed placement done on individual tasks and memory regions using the `sched_setaffinity`, `mbind` and `set_mempolicy` system calls.

The following rules apply to each cpuset:

- Its CPUs and Memory Nodes must be a subset of its parents.
- It can’ t be marked exclusive unless its parent is.
- If its cpu or memory is exclusive, they may not overlap any sibling.

These rules, and the natural hierarchy of cpusets, enable efficient enforcement of the exclusive guarantee, without having to scan all cpusets every time any of them change to ensure nothing overlaps a exclusive cpuset. Also, the use of a Linux virtual file system (vfs) to represent the cpuset hierarchy provides for a familiar permission and name space for cpusets, with a minimum of additional kernel code.

The `cpus` and `mems` files in the root (`top_cpuset`) cpuset are read-only. The `cpus` file automatically tracks the value of `cpu_online_mask` using a CPU hotplug notifier, and the `mems` file automatically tracks the value of `node_states[N_MEMORY]`-i.e., nodes with memory-using the `cpuset_track_online_nodes()` hook.

The `cpuset.effective_cpus` and `cpuset.effective_mems` files are normally read-only copies of `cpuset.cpus` and `cpuset.mems` files respectively. If the cpuset cgroup filesystem is mounted with the special “`cpuset_v2_mode`” option, the behavior of these files will become similar to the corresponding files in cpuset v2. In other

words, hotplug events will not change `cpuset.cpus` and `cpuset.mems`. Those events will only affect `cpuset.effective_cpus` and `cpuset.effective_mems` which show the actual cpus and memory nodes that are currently used by this cpuset. See [Documentation/admin-guide/cgroup-v2.rst](#) for more information about cpuset v2 behavior.

#### 1.4 What are exclusive cpusets ?

If a cpuset is `cpu` or `mem` exclusive, no other cpuset, other than a direct ancestor or descendant, may share any of the same CPUs or Memory Nodes.

A cpuset that is `cpuset.mem_exclusive` or `cpuset.mem_hardwall` is “hardwalled” , i.e. it restricts kernel allocations for page, buffer and other data commonly shared by the kernel across multiple users. All cpusets, whether hardwalled or not, restrict allocations of memory for user space. This enables configuring a system so that several independent jobs can share common kernel data, such as file system pages, while isolating each job’ s user allocation in its own cpuset. To do this, construct a large `mem_exclusive` cpuset to hold all the jobs, and construct child, non-`mem_exclusive` cpusets for each individual job. Only a small amount of typical kernel memory, such as requests from interrupt handlers, is allowed to be taken outside even a `mem_exclusive` cpuset.

#### 1.5 What is memory\_pressure ?

The `memory_pressure` of a cpuset provides a simple per-cpuset metric of the rate that the tasks in a cpuset are attempting to free up in use memory on the nodes of the cpuset to satisfy additional memory requests.

This enables batch managers monitoring jobs running in dedicated cpusets to efficiently detect what level of memory pressure that job is causing.

This is useful both on tightly managed systems running a wide mix of submitted jobs, which may choose to terminate or re-prioritize jobs that are trying to use more memory than allowed on the nodes assigned to them, and with tightly coupled, long running, massively parallel scientific computing jobs that will dramatically fail to meet required performance goals if they start to use more memory than allowed to them.

This mechanism provides a very economical way for the batch manager to monitor a cpuset for signs of memory pressure. It’ s up to the batch manager or other user code to decide what to do about it and take action.

==> Unless this feature is enabled by writing “1” to the special file `/dev/cpuset/memory_pressure_enabled`, the hook in the rebalance code of `__alloc_pages()` for this metric reduces to simply noticing that the `cpuset_memory_pressure_enabled` flag is zero. So only systems that enable this feature will compute the metric.

Why a per-cpuset, running average:

Because this meter is per-cpuset, rather than per-task or mm, the system load imposed by a batch scheduler monitoring this metric is sharply reduced on large systems, because a scan of the tasklist can be avoided on each set of queries.

Because this meter is a running average, instead of an accumulating counter, a batch scheduler can detect memory pressure with a single read, instead of having to read and accumulate results for a period of time.

Because this meter is per-cpuset rather than per-task or mm, the batch scheduler can obtain the key information, memory pressure in a cpuset, with a single read, rather than having to query and accumulate results over all the (dynamically changing) set of tasks in the cpuset.

A per-cpuset simple digital filter (requires a spinlock and 3 words of data per-cpuset) is kept, and updated by any task attached to that cpuset, if it enters the synchronous (direct) page reclaim code.

A per-cpuset file provides an integer number representing the recent (half-life of 10 seconds) rate of direct page reclaims caused by the tasks in the cpuset, in units of reclaims attempted per second, times 1000.

### 1.6 What is memory spread ?

There are two boolean flag files per cpuset that control where the kernel allocates pages for the file system buffers and related in kernel data structures. They are called `'cpuset.memory_spread_page'` and `'cpuset.memory_spread_slab'` .

If the per-cpuset boolean flag file `'cpuset.memory_spread_page'` is set, then the kernel will spread the file system buffers (page cache) evenly over all the nodes that the faulting task is allowed to use, instead of preferring to put those pages on the node where the task is running.

If the per-cpuset boolean flag file `'cpuset.memory_spread_slab'` is set, then the kernel will spread some file system related slab caches, such as for inodes and dentries evenly over all the nodes that the faulting task is allowed to use, instead of preferring to put those pages on the node where the task is running.

The setting of these flags does not affect anonymous data segment or stack segment pages of a task.

By default, both kinds of memory spreading are off, and memory pages are allocated on the node local to where the task is running, except perhaps as modified by the task's NUMA mempolicy or cpuset configuration, so long as sufficient free memory pages are available.

When new cpusets are created, they inherit the memory spread settings of their parent.

Setting memory spreading causes allocations for the affected page or slab caches to ignore the task's NUMA mempolicy and be spread instead. Tasks using `mbind()` or `set_mempolicy()` calls to set NUMA mempolicies will not notice any change in these calls as a result of their containing task's memory spread settings. If memory spreading is turned off, then the currently specified NUMA mempolicy once again applies to memory page allocations.

Both `'cpuset.memory_spread_page'` and `'cpuset.memory_spread_slab'` are boolean flag files. By default they contain `"0"` , meaning that the feature is off for that cpuset. If a `"1"` is written to that file, then that turns the named feature on.

The implementation is simple.

Setting the flag `'cpuset.memory_spread_page'` turns on a per-process flag `PFA_SPREAD_PAGE` for each task that is in that cpuset or subsequently joins that cpuset. The page allocation calls for the page cache is modified to perform an in-line check for this `PFA_SPREAD_PAGE` task flag, and if set, a call to a new routine `cpuset_mem_spread_node()` returns the node to prefer for the allocation.

Similarly, setting `'cpuset.memory_spread_slab'` turns on the flag `PFA_SPREAD_SLAB`, and appropriately marked slab caches will allocate pages from the node returned by `cpuset_mem_spread_node()`.

The `cpuset_mem_spread_node()` routine is also simple. It uses the value of a per-task rotor `cpuset_mem_spread_rotor` to select the next node in the current task's `mems_allowed` to prefer for the allocation.

This memory placement policy is also known (in other contexts) as round-robin or interleave.

This policy can provide substantial improvements for jobs that need to place thread local data on the corresponding node, but that need to access large file system data sets that need to be spread across the several nodes in the jobs cpuset in order to fit. Without this policy, especially for jobs that might have one thread reading in the data set, the memory allocation across the nodes in the jobs cpuset can become very uneven.

## 1.7 What is sched\_load\_balance ?

The kernel scheduler (`kernel/sched/core.c`) automatically load balances tasks. If one CPU is underutilized, kernel code running on that CPU will look for tasks on other more overloaded CPUs and move those tasks to itself, within the constraints of such placement mechanisms as cpusets and `sched_setaffinity`.

The algorithmic cost of load balancing and its impact on key shared kernel data structures such as the task list increases more than linearly with the number of CPUs being balanced. So the scheduler has support to partition the systems CPUs into a number of sched domains such that it only load balances within each sched domain. Each sched domain covers some subset of the CPUs in the system; no two sched domains overlap; some CPUs might not be in any sched domain and hence won't be load balanced.

Put simply, it costs less to balance between two smaller sched domains than one big one, but doing so means that overloads in one of the two domains won't be load balanced to the other one.

By default, there is one sched domain covering all CPUs, including those marked isolated using the kernel boot time `"isolcpus="` argument. However, the isolated CPUs will not participate in load balancing, and will not have tasks running on them unless explicitly assigned.

This default load balancing across all CPUs is not well suited for the following two situations:

- 1) On large systems, load balancing across many CPUs is expensive. If the system is managed using cpusets to place independent jobs on separate sets of CPUs, full load balancing is unnecessary.

- 2) Systems supporting realtime on some CPUs need to minimize system overhead on those CPUs, including avoiding task load balancing if that is not needed.

When the per-cpuset flag “cpuset.sched\_load\_balance” is enabled (the default setting), it requests that all the CPUs in that cpuset allowed ‘cpuset.cpus’ be contained in a single sched domain, ensuring that load balancing can move a task (not otherwise pinned, as by sched\_setaffinity) from any CPU in that cpuset to any other.

When the per-cpuset flag “cpuset.sched\_load\_balance” is disabled, then the scheduler will avoid load balancing across the CPUs in that cpuset, -except- in so far as is necessary because some overlapping cpuset has “sched\_load\_balance” enabled.

So, for example, if the top cpuset has the flag “cpuset.sched\_load\_balance” enabled, then the scheduler will have one sched domain covering all CPUs, and the setting of the “cpuset.sched\_load\_balance” flag in any other cpusets won’ t matter, as we’re already fully load balancing.

Therefore in the above two situations, the top cpuset flag “cpuset.sched\_load\_balance” should be disabled, and only some of the smaller, child cpusets have this flag enabled.

When doing this, you don’ t usually want to leave any unpinned tasks in the top cpuset that might use non-trivial amounts of CPU, as such tasks may be artificially constrained to some subset of CPUs, depending on the particulars of this flag setting in descendant cpusets. Even if such a task could use spare CPU cycles in some other CPUs, the kernel scheduler might not consider the possibility of load balancing that task to that underused CPU.

Of course, tasks pinned to a particular CPU can be left in a cpuset that disables “cpuset.sched\_load\_balance” as those tasks aren’ t going anywhere else anyway.

There is an impedance mismatch here, between cpusets and sched domains. Cpusets are hierarchical and nest. Sched domains are flat; they don’ t overlap and each CPU is in at most one sched domain.

It is necessary for sched domains to be flat because load balancing across partially overlapping sets of CPUs would risk unstable dynamics that would be beyond our understanding. So if each of two partially overlapping cpusets enables the flag ‘cpuset.sched\_load\_balance’, then we form a single sched domain that is a superset of both. We won’ t move a task to a CPU outside its cpuset, but the scheduler load balancing code might waste some compute cycles considering that possibility.

This mismatch is why there is not a simple one-to-one relation between which cpusets have the flag “cpuset.sched\_load\_balance” enabled, and the sched domain configuration. If a cpuset enables the flag, it will get balancing across all its CPUs, but if it disables the flag, it will only be assured of no load balancing if no other overlapping cpuset enables the flag.

If two cpusets have partially overlapping ‘cpuset.cpus’ allowed, and only one of them has this flag enabled, then the other may find its tasks only partially load balanced, just on the overlapping CPUs. This is just the general case of the top\_cpuset example given a few paragraphs above. In the general case, as in the top\_cpuset case, don’ t leave tasks that might use non-trivial amounts of CPU in such partially

load balanced cpusets, as they may be artificially constrained to some subset of the CPUs allowed to them, for lack of load balancing to the other CPUs.

CPUs in “cpuset.isolcpus” were excluded from load balancing by the `isolcpus=` kernel boot option, and will never be load balanced regardless of the value of “cpuset.sched\_load\_balance” in any cpuset.

### **1.7.1 sched\_load\_balance implementation details.**

The per-cpuset flag ‘cpuset.sched\_load\_balance’ defaults to enabled (contrary to most cpuset flags.) When enabled for a cpuset, the kernel will ensure that it can load balance across all the CPUs in that cpuset (makes sure that all the CPUs in the `cpus_allowed` of that cpuset are in the same sched domain.)

If two overlapping cpusets both have ‘cpuset.sched\_load\_balance’ enabled, then they will be (must be) both in the same sched domain.

If, as is the default, the top cpuset has ‘cpuset.sched\_load\_balance’ enabled, then by the above that means there is a single sched domain covering the whole system, regardless of any other cpuset settings.

The kernel commits to user space that it will avoid load balancing where it can. It will pick as fine a granularity partition of sched domains as it can while still providing load balancing for any set of CPUs allowed to a cpuset having ‘cpuset.sched\_load\_balance’ enabled.

The internal kernel cpuset to scheduler interface passes from the cpuset code to the scheduler code a partition of the load balanced CPUs in the system. This partition is a set of subsets (represented as an array of struct `cpumask`) of CPUs, pairwise disjoint, that cover all the CPUs that must be load balanced.

The cpuset code builds a new such partition and passes it to the scheduler `sched_domain_setup` code, to have the sched domains rebuilt as necessary, whenever:

- the ‘cpuset.sched\_load\_balance’ flag of a cpuset with non-empty CPUs changes,
- or CPUs come or go from a cpuset with this flag enabled,
- or ‘cpuset.sched\_relax\_domain\_level’ value of a cpuset with non-empty CPUs and with this flag enabled changes,
- or a cpuset with non-empty CPUs and with this flag enabled is removed,
- or a cpu is offlined/onlineed.

This partition exactly defines what sched domains the scheduler should setup - one sched domain for each element (struct `cpumask`) in the partition.

The scheduler remembers the currently active sched domain partitions. When the scheduler routine `partition_sched_domains()` is invoked from the cpuset code to update these sched domains, it compares the new partition requested with the current, and updates its sched domains, removing the old and adding the new, for each change.

### 1.8 What is sched\_relax\_domain\_level ?

In sched domain, the scheduler migrates tasks in 2 ways; periodic load balance on tick, and at time of some schedule events.

When a task is woken up, scheduler try to move the task on idle CPU. For example, if a task A running on CPU X activates another task B on the same CPU X, and if CPU Y is X' s sibling and performing idle, then scheduler migrate task B to CPU Y so that task B can start on CPU Y without waiting task A on CPU X.

And if a CPU run out of tasks in its runqueue, the CPU try to pull extra tasks from other busy CPUs to help them before it is going to be idle.

Of course it takes some searching cost to find movable tasks and/or idle CPUs, the scheduler might not search all CPUs in the domain every time. In fact, in some architectures, the searching ranges on events are limited in the same socket or node where the CPU locates, while the load balance on tick searches all.

For example, assume CPU Z is relatively far from CPU X. Even if CPU Z is idle while CPU X and the siblings are busy, scheduler can' t migrate woken task B from X to Z since it is out of its searching range. As the result, task B on CPU X need to wait task A or wait load balance on the next tick. For some applications in special situation, waiting 1 tick may be too long.

The 'cpuset.sched\_relax\_domain\_level' file allows you to request changing this searching range as you like. This file takes int value which indicates size of searching range in levels ideally as follows, otherwise initial value -1 that indicates the cpuset has no request.

-1	no request. use system default or follow request of others.
0	no search.
1	search siblings (hyperthreads in a core).
2	search cores in a package.
3	search cpus in a node [= system wide on non-NUMA system]
4	search nodes in a chunk of node [on NUMA system]
5	search system wide [on NUMA system]

The system default is architecture dependent. The system default can be changed using the relax\_domain\_level= boot parameter.

This file is per-cpuset and affect the sched domain where the cpuset belongs to. Therefore if the flag 'cpuset.sched\_load\_balance' of a cpuset is disabled, then 'cpuset.sched\_relax\_domain\_level' have no effect since there is no sched domain belonging the cpuset.

If multiple cpusets are overlapping and hence they form a single sched domain, the largest value among those is used. Be careful, if one requests 0 and others are -1 then 0 is used.

Note that modifying this file will have both good and bad effects, and whether it is acceptable or not depends on your situation. Don' t modify this file if you are not sure.

If your situation is:

- The migration costs between each cpu can be assumed considerably small(for you) due to your special application' s behavior or special hardware support for CPU cache etc.
- The searching cost doesn' t have impact(for you) or you can make the searching cost enough small by managing cpuset to compact etc.
- The latency is required even it sacrifices cache hit rate etc. then increasing 'sched\_relax\_domain\_level' would benefit you.

### 1.9 How do I use cpusets ?

In order to minimize the impact of cpusets on critical kernel code, such as the scheduler, and due to the fact that the kernel does not support one task updating the memory placement of another task directly, the impact on a task of changing its cpuset CPU or Memory Node placement, or of changing to which cpuset a task is attached, is subtle.

If a cpuset has its Memory Nodes modified, then for each task attached to that cpuset, the next time that the kernel attempts to allocate a page of memory for that task, the kernel will notice the change in the task' s cpuset, and update its per-task memory placement to remain within the new cpusets memory placement. If the task was using mempolicy MPOL\_BIND, and the nodes to which it was bound overlap with its new cpuset, then the task will continue to use whatever subset of MPOL\_BIND nodes are still allowed in the new cpuset. If the task was using MPOL\_BIND and now none of its MPOL\_BIND nodes are allowed in the new cpuset, then the task will be essentially treated as if it was MPOL\_BIND bound to the new cpuset (even though its NUMA placement, as queried by `get_mempolicy()`, doesn' t change). If a task is moved from one cpuset to another, then the kernel will adjust the task' s memory placement, as above, the next time that the kernel attempts to allocate a page of memory for that task.

If a cpuset has its 'cpuset.cpus' modified, then each task in that cpuset will have its allowed CPU placement changed immediately. Similarly, if a task' s pid is written to another cpuset' s 'tasks' file, then its allowed CPU placement is changed immediately. If such a task had been bound to some subset of its cpuset using the `sched_setaffinity()` call, the task will be allowed to run on any CPU allowed in its new cpuset, negating the effect of the prior `sched_setaffinity()` call.

In summary, the memory placement of a task whose cpuset is changed is updated by the kernel, on the next allocation of a page for that task, and the processor placement is updated immediately.

Normally, once a page is allocated (given a physical page of main memory) then that page stays on whatever node it was allocated, so long as it remains allocated, even if the cpusets memory placement policy 'cpuset.mems' subsequently changes. If the cpuset flag file 'cpuset.memory\_migrate' is set true, then when tasks are attached to that cpuset, any pages that task had allocated to it on nodes in its previous cpuset are migrated to the task' s new cpuset. The relative placement of the page within the cpuset is preserved during these migration operations if possible. For example if the page was on the second valid node of the prior cpuset then the page will be placed on the second valid node of the new cpuset.

Also if 'cpuset.memory\_migrate' is set true, then if that cpuset' s 'cpuset.mems'

file is modified, pages allocated to tasks in that cpuset, that were on nodes in the previous setting of 'cpuset.mems', will be moved to nodes in the new setting of 'mems.' Pages that were not in the task's prior cpuset, or in the cpuset's prior 'cpuset.mems' setting, will not be moved.

There is an exception to the above. If hotplug functionality is used to remove all the CPUs that are currently assigned to a cpuset, then all the tasks in that cpuset will be moved to the nearest ancestor with non-empty cpus. But the moving of some (or all) tasks might fail if cpuset is bound with another cgroup subsystem which has some restrictions on task attaching. In this failing case, those tasks will stay in the original cpuset, and the kernel will automatically update their cpus\_allowed to allow all online CPUs. When memory hotplug functionality for removing Memory Nodes is available, a similar exception is expected to apply there as well. In general, the kernel prefers to violate cpuset placement, over starving a task that has had all its allowed CPUs or Memory Nodes taken offline.

There is a second exception to the above. GFP\_ATOMIC requests are kernel internal allocations that must be satisfied, immediately. The kernel may drop some request, in rare cases even panic, if a GFP\_ATOMIC alloc fails. If the request cannot be satisfied within the current task's cpuset, then we relax the cpuset, and look for memory anywhere we can find it. It's better to violate the cpuset than stress the kernel.

To start a new job that is to be contained within a cpuset, the steps are:

- 1) mkdir /sys/fs/cgroup/cpuset
- 2) mount -t cgroup -ocpuset cpuset /sys/fs/cgroup/cpuset
- 3) Create the new cpuset by doing mkdir's and write's (or echo's) in the /sys/fs/cgroup/cpuset virtual file system.
- 4) Start a task that will be the "founding father" of the new job.
- 5) Attach that task to the new cpuset by writing its pid to the /sys/fs/cgroup/cpuset tasks file for that cpuset.
- 6) fork, exec or clone the job tasks from this founding father task.

For example, the following sequence of commands will setup a cpuset named "Charlie", containing just CPUs 2 and 3, and Memory Node 1, and then start a subshell 'sh' in that cpuset:

```
mount -t cgroup -ocpuset cpuset /sys/fs/cgroup/cpuset
cd /sys/fs/cgroup/cpuset
mkdir Charlie
cd Charlie
/bin/echo 2-3 > cpuset.cpus
/bin/echo 1 > cpuset.mems
/bin/echo $$ > tasks
sh
# The subshell 'sh' is now running in cpuset Charlie
# The next line should display '/Charlie'
cat /proc/self/cpuset
```

There are ways to query or modify cpusets:

- via the cpuset file system directly, using the various cd, mkdir, echo, cat, rmdir commands from the shell, or their equivalent from C.
- via the C library libcpuset.
- via the C library libcgroup. (<http://sourceforge.net/projects/libcg/>)
- via the python application cset. (<http://code.google.com/p/cpuset/>)

The sched\_setaffinity calls can also be done at the shell prompt using SGI' s runon or Robert Love' s taskset. The mbind and set\_mempolicy calls can be done at the shell prompt using the numactl command (part of Andi Kleen' s numa package).

## 27.4.2 2. Usage Examples and Syntax

### 2.1 Basic Usage

Creating, modifying, using the cpusets can be done through the cpuset virtual filesystem.

To mount it, type: `# mount -t cgroup -o cpuset cpuset /sys/fs/cgroup/cpuset`

Then under `/sys/fs/cgroup/cpuset` you can find a tree that corresponds to the tree of the cpusets in the system. For instance, `/sys/fs/cgroup/cpuset` is the cpuset that holds the whole system.

If you want to create a new cpuset under `/sys/fs/cgroup/cpuset`:

```
# cd /sys/fs/cgroup/cpuset
# mkdir my_cpuset
```

Now you want to do something with this cpuset:

```
# cd my_cpuset
```

In this directory you can find several files:

```
# ls
cgroup.clone_children  cpuset.memory_pressure
cgroup.event_control  cpuset.memory_spread_page
cgroup.procs          cpuset.memory_spread_slab
cpuset.cpu_exclusive  cpuset.mems
cpuset.cpus           cpuset.sched_load_balance
cpuset.mem_exclusive  cpuset.sched_relax_domain_level
cpuset.mem_hardwall   notify_on_release
cpuset.memory_migrate tasks
```

Reading them will give you information about the state of this cpuset: the CPUs and Memory Nodes it can use, the processes that are using it, its properties. By writing to these files you can manipulate the cpuset.

Set some flags:

```
# /bin/echo 1 > cpuset.cpu_exclusive
```

Add some cpus:

```
# /bin/echo 0-7 > cpuset.cpus
```

Add some mems:

```
# /bin/echo 0-7 > cpuset.mems
```

Now attach your shell to this cpuset:

```
# /bin/echo $$ > tasks
```

You can also create cpubsets inside your cpuset by using mkdir in this directory:

```
# mkdir my_sub_cs
```

To remove a cpuset, just use rmdir:

```
# rmdir my_sub_cs
```

This will fail if the cpuset is in use (has cpubsets inside, or has processes attached).

Note that for legacy reasons, the “cpuset” filesystem exists as a wrapper around the cgroup filesystem.

The command:

```
mount -t cpuset X /sys/fs/cgroup/cpuset
```

is equivalent to:

```
mount -t cgroup -ocpuset,noprefix X /sys/fs/cgroup/cpuset
echo "/sbin/cpuset_release_agent" > /sys/fs/cgroup/cpuset/release_agent
```

## 2.2 Adding/removing cpus

This is the syntax to use when writing in the cpus or mems files in cpuset directories:

```
# /bin/echo 1-4 > cpuset.cpus      -> set cpus list to cpus 1,2,3,4
# /bin/echo 1,2,3,4 > cpuset.cpus  -> set cpus list to cpus 1,2,3,4
```

To add a CPU to a cpuset, write the new list of CPUs including the CPU to be added.  
To add 6 to the above cpuset:

```
# /bin/echo 1-4,6 > cpuset.cpus    -> set cpus list to cpus 1,2,3,4,6
```

Similarly to remove a CPU from a cpuset, write the new list of CPUs without the CPU to be removed.

To remove all the CPUs:

```
# /bin/echo "" > cpuset.cpus      -> clear cpus list
```

## 2.3 Setting flags

The syntax is very simple:

```
# /bin/echo 1 > cpuset.cpu_exclusive -> set flag 'cpuset.cpu_exclusive'
# /bin/echo 0 > cpuset.cpu_exclusive -> unset flag 'cpuset.cpu_exclusive'
```

## 2.4 Attaching processes

```
# /bin/echo PID > tasks
```

Note that it is PID, not PIDs. You can only attach ONE task at a time. If you have several tasks to attach, you have to do it one after another:

```
# /bin/echo PID1 > tasks
# /bin/echo PID2 > tasks
...
# /bin/echo PIDn > tasks
```

### 27.4.3 3. Questions

**Q:** what' s up with this `/bin/echo` ?

**A:** bash' s builtin `echo` command does not check calls to `write()` against errors. If you use it in the `cpuset` file system, you won' t be able to tell whether a command succeeded or failed.

**Q:** When I attach processes, only the first of the line gets really attached !

**A:** We can only return one error code per call to `write()`. So you should also put only ONE pid.

### 27.4.4 4. Contact

Web: <http://www.bullopen-source.org/cpuset>

## 27.5 Device Whitelist Controller

### 27.5.1 1. Description

Implement a cgroup to track and enforce open and `mknod` restrictions on device files. A device cgroup associates a device access whitelist with each cgroup. A whitelist entry has 4 fields. `type` is a (all), c (char), or b (block). `all` means it applies to all types and all major and minor numbers. Major and minor are either an integer or \* for all. Access is a composition of r (read), w (write), and m (`mknod`).

The root device cgroup starts with `rwm` to `all` . A child device cgroup gets a copy of the parent. Administrators can then remove devices from the whitelist or add new entries. A child cgroup can never receive a device access which is denied by its parent.

### 27.5.2 2. User Interface

An entry is added using `devices.allow`, and removed using `devices.deny`. For instance:

```
echo 'c 1:3 mr' > /sys/fs/cgroup/1/devices.allow
```

allows cgroup 1 to read and mknod the device usually known as `/dev/null`. Doing:

```
echo a > /sys/fs/cgroup/1/devices.deny
```

will remove the default `'a : rwm'` entry. Doing:

```
echo a > /sys/fs/cgroup/1/devices.allow
```

will add the `'a : rwm'` entry to the whitelist.

### 27.5.3 3. Security

Any task can move itself between cgroups. This clearly won't suffice, but we can decide the best way to adequately restrict movement as people get some experience with this. We may just want to require `CAP_SYS_ADMIN`, which at least is a separate bit from `CAP_MKNOD`. We may want to just refuse moving to a cgroup which isn't a descendant of the current one. Or we may want to use `CAP_MAC_ADMIN`, since we really are trying to lock down root.

`CAP_SYS_ADMIN` is needed to modify the whitelist or move another task to a new cgroup. (Again we'll probably want to change that).

A cgroup may not be granted more permissions than the cgroup's parent has.

### 27.5.4 4. Hierarchy

device cgroups maintain hierarchy by making sure a cgroup never has more access permissions than its parent. Every time an entry is written to a cgroup's `devices.deny` file, all its children will have that entry removed from their whitelist and all the locally set whitelist entries will be re-evaluated. In case one of the locally set whitelist entries would provide more access than the cgroup's parent, it'll be removed from the whitelist.

Example:

<pre>A  / \   B</pre>		
group	behavior	exceptions
A	allow	"b 8:* rwm", "c 116:1 rw"
B	deny	"c 1:3 rwm", "c 116:2 rwm", "b 3:* rwm"

If a device is denied in group A:

```
# echo "c 116:* r" > A/devices.deny
```

it'll propagate down and after revalidating B's entries, the whitelist entry "c 116:2 rwm" will be removed:

group	whitelist entries	denied devices
A	all →116:* rw"	"b 8:* rwm", "c 116:2 rwm"
B	"c 1:3 rwm", "b 3:* rwm"	all the rest

In case parent's exceptions change and local exceptions are not allowed anymore, they'll be deleted.

Notice that new whitelist entries will not be propagated:

group	whitelist entries	denied devices
A	"c 1:3 rwm", "c 1:5 r"	all the rest
B	"c 1:3 rwm", "c 1:5 r"	all the rest

when adding c \*:3 rwm:

```
# echo "c *:3 rwm" >A/devices.allow
```

the result:

group	whitelist entries	denied devices
A	"c *:3 rwm", "c 1:5 r"	all the rest
B	"c 1:3 rwm", "c 1:5 r"	all the rest

but now it'll be possible to add new entries to B:

```
# echo "c 2:3 rwm" >B/devices.allow
# echo "c 50:3 r" >B/devices.allow
```

or even:

```
# echo "c *:3 rwm" >B/devices.allow
```

Allowing or denying all by writing 'a' to devices.allow or devices.deny will not be possible once the device cgroups has children.

#### 4.1 Hierarchy (internal implementation)

device cgroups is implemented internally using a behavior (ALLOW, DENY) and a list of exceptions. The internal state is controlled using the same user interface to preserve compatibility with the previous whitelist-only implementation. Removal or addition of exceptions that will reduce the access to devices will be propagated down the hierarchy. For every propagated exception, the effective rules will be re-evaluated based on current parent's access rules.

## 27.6 Cgroup Freezer

The cgroup freezer is useful to batch job management system which start and stop sets of tasks in order to schedule the resources of a machine according to the desires of a system administrator. This sort of program is often used on HPC clusters to schedule access to the cluster as a whole. The cgroup freezer uses cgroups to describe the set of tasks to be started/stopped by the batch job management system. It also provides a means to start and stop the tasks composing the job.

The cgroup freezer will also be useful for checkpointing running groups of tasks. The freezer allows the checkpoint code to obtain a consistent image of the tasks by attempting to force the tasks in a cgroup into a quiescent state. Once the tasks are quiescent another task can walk /proc or invoke a kernel interface to gather information about the quiesced tasks. Checkpointed tasks can be restarted later should a recoverable error occur. This also allows the checkpointed tasks to be migrated between nodes in a cluster by copying the gathered information to another node and restarting the tasks there.

Sequences of SIGSTOP and SIGCONT are not always sufficient for stopping and resuming tasks in userspace. Both of these signals are observable from within the tasks we wish to freeze. While SIGSTOP cannot be caught, blocked, or ignored it can be seen by waiting or ptracing parent tasks. SIGCONT is especially unsuitable since it can be caught by the task. Any programs designed to watch for SIGSTOP and SIGCONT could be broken by attempting to use SIGSTOP and SIGCONT to stop and resume tasks. We can demonstrate this problem using nested bash shells:

```
$ echo $$
16644
$ bash
$ echo $$
16690

From a second, unrelated bash shell:
$ kill -SIGSTOP 16690
$ kill -SIGCONT 16690

<at this point 16690 exits and causes 16644 to exit too>
```

This happens because bash can observe both signals and choose how it responds to them.

Another example of a program which catches and responds to these signals is gdb. In fact any program designed to use ptrace is likely to have a problem with this method of stopping and resuming tasks.

In contrast, the cgroup freezer uses the kernel freezer code to prevent the freeze/unfreeze cycle from becoming visible to the tasks being frozen. This allows the bash example above and gdb to run as expected.

The cgroup freezer is hierarchical. Freezing a cgroup freezes all tasks belonging to the cgroup and all its descendant cgroups. Each cgroup has its own state (self-state) and the state inherited from the parent (parent-state). Iff both states are THAWED, the cgroup is THAWED.

The following cgroupfs files are created by cgroup freezer.

- freezer.state: Read-write.

When read, returns the effective state of the cgroup - “THAWED” , “FREEZING” or “FROZEN” . This is the combined self and parent-states. If any is freezing, the cgroup is freezing (FREEZING or FROZEN).

FREEZING cgroup transitions into FROZEN state when all tasks belonging to the cgroup and its descendants become frozen. Note that a cgroup reverts to FREEZING from FROZEN after a new task is added to the cgroup or one of its descendant cgroups until the new task is frozen.

When written, sets the self-state of the cgroup. Two values are allowed - “FROZEN” and “THAWED” . If FROZEN is written, the cgroup, if not already freezing, enters FREEZING state along with all its descendant cgroups.

If THAWED is written, the self-state of the cgroup is changed to THAWED. Note that the effective state may not change to THAWED if the parent-state is still freezing. If a cgroup’ s effective state becomes THAWED, all its descendants which are freezing because of the cgroup also leave the freezing state.

- freezer.self\_freezing: Read only.

Shows the self-state. 0 if the self-state is THAWED; otherwise, 1. This value is 1 iff the last write to freezer.state was “FROZEN” .

- freezer.parent\_freezing: Read only.

Shows the parent-state. 0 if none of the cgroup’ s ancestors is frozen; otherwise, 1.

The root cgroup is non-freezable and the above interface files don’ t exist.

- Examples of usage:

```
# mkdir /sys/fs/cgroup/freezer
# mount -t cgroup -ofreezer freezer /sys/fs/cgroup/freezer
# mkdir /sys/fs/cgroup/freezer/0
# echo $some_pid > /sys/fs/cgroup/freezer/0/tasks
```

to get status of the freezer subsystem:

```
# cat /sys/fs/cgroup/freezer/0/freezer.state
THAWED
```

to freeze all tasks in the container:

```
# echo FROZEN > /sys/fs/cgroup/freezer/0/freezer.state
# cat /sys/fs/cgroup/freezer/0/freezer.state
FREEZING
# cat /sys/fs/cgroup/freezer/0/freezer.state
FROZEN
```

to unfreeze all tasks in the container:

```
# echo THAWED > /sys/fs/cgroup/freezer/0/freezer.state
# cat /sys/fs/cgroup/freezer/0/freezer.state
THAWED
```

This is the basic mechanism which should do the right thing for user space task in a simple scenario.

## 27.7 HugeTLB Controller

HugeTLB controller can be created by first mounting the cgroup filesystem.

```
# mount -t cgroup -o hugetlb none /sys/fs/cgroup
```

With the above step, the initial or the parent HugeTLB group becomes visible at `/sys/fs/cgroup`. At bootup, this group includes all the tasks in the system. `/sys/fs/cgroup/tasks` lists the tasks in this cgroup.

New groups can be created under the parent group `/sys/fs/cgroup`:

```
# cd /sys/fs/cgroup
# mkdir g1
# echo $$ > g1/tasks
```

The above steps create a new group `g1` and move the current shell process (`bash`) into it.

Brief summary of control files:

```
hugetlb.<hugepagesize>.rsvd.limit_in_bytes          # set/show limit of
↳ "hugepagesize" hugetlb reservations
hugetlb.<hugepagesize>.rsvd.max_usage_in_bytes      # show max
↳ "hugepagesize" hugetlb reservations and no-reserve faults
hugetlb.<hugepagesize>.rsvd.usage_in_bytes          # show current
↳ reservations and no-reserve faults for "hugepagesize" hugetlb
hugetlb.<hugepagesize>.rsvd.failcnt                 # show the number of
↳ allocation failure due to HugeTLB reservation limit
hugetlb.<hugepagesize>.limit_in_bytes              # set/show limit of
↳ "hugepagesize" hugetlb faults
hugetlb.<hugepagesize>.max_usage_in_bytes          # show max
↳ "hugepagesize" hugetlb usage recorded
hugetlb.<hugepagesize>.usage_in_bytes              # show current usage
↳ for "hugepagesize" hugetlb
hugetlb.<hugepagesize>.failcnt                     # show the number of
↳ allocation failure due to HugeTLB usage limit
```

For a system supporting three hugepage sizes (64k, 32M and 1G), the control files include:

```
hugetlb.1GB.limit_in_bytes
hugetlb.1GB.max_usage_in_bytes
hugetlb.1GB.usage_in_bytes
hugetlb.1GB.failcnt
hugetlb.1GB.rsvd.limit_in_bytes
hugetlb.1GB.rsvd.max_usage_in_bytes
hugetlb.1GB.rsvd.usage_in_bytes
hugetlb.1GB.rsvd.failcnt
hugetlb.64KB.limit_in_bytes
hugetlb.64KB.max_usage_in_bytes
hugetlb.64KB.usage_in_bytes
```

(continues on next page)

(continued from previous page)

```

hugetlb.64KB.failcnt
hugetlb.64KB.rsvd.limit_in_bytes
hugetlb.64KB.rsvd.max_usage_in_bytes
hugetlb.64KB.rsvd.usage_in_bytes
hugetlb.64KB.rsvd.failcnt
hugetlb.32MB.limit_in_bytes
hugetlb.32MB.max_usage_in_bytes
hugetlb.32MB.usage_in_bytes
hugetlb.32MB.failcnt
hugetlb.32MB.rsvd.limit_in_bytes
hugetlb.32MB.rsvd.max_usage_in_bytes
hugetlb.32MB.rsvd.usage_in_bytes
hugetlb.32MB.rsvd.failcnt

```

### 1. Page fault accounting

```

hugetlb.<hugepagesize>.limit_in_bytes hugetlb.<hugepagesize>.max_usage_in_bytes
hugetlb.<hugepagesize>.usage_in_bytes hugetlb.<hugepagesize>.failcnt

```

The HugeTLB controller allows users to limit the HugeTLB usage (page fault) per control group and enforces the limit during page fault. Since HugeTLB doesn't support page reclaim, enforcing the limit at page fault time implies that, the application will get SIGBUS signal if it tries to fault in HugeTLB pages beyond its limit. Therefore the application needs to know exactly how many HugeTLB pages it uses before hand, and the sysadmin needs to make sure that there are enough available on the machine for all the users to avoid processes getting SIGBUS.

### 2. Reservation accounting

```

hugetlb.<hugepagesize>.rsvd.limit_in_bytes hugetlb.<hugepagesize>.rsvd.max_usage_in_bytes
hugetlb.<hugepagesize>.rsvd.usage_in_bytes hugetlb.<hugepagesize>.rsvd.failcnt

```

The HugeTLB controller allows to limit the HugeTLB reservations per control group and enforces the controller limit at reservation time and at the fault of HugeTLB memory for which no reservation exists. Since reservation limits are enforced at reservation time (on `mmap` or `shget`), reservation limits never causes the application to get SIGBUS signal if the memory was reserved before hand. For `MAP_NORESERVE` allocations, the reservation limit behaves the same as the fault limit, enforcing memory usage at fault time and causing the application to receive a SIGBUS if it's crossing its limit.

Reservation limits are superior to page fault limits described above, since reservation limits are enforced at reservation time (on `mmap` or `shget`), and never causes the application to get SIGBUS signal if the memory was reserved before hand. This allows for easier fallback to alternatives such as non-HugeTLB memory for example. In the case of page fault accounting, it's very hard to avoid processes getting SIGBUS since the sysadmin needs precisely know the HugeTLB usage of all the tasks in the system and make sure there is enough pages to satisfy all requests. Avoiding tasks getting SIGBUS on overcommitted systems is practically impossible with page fault accounting.

### 3. Caveats with shared memory

For shared HugeTLB memory, both HugeTLB reservation and page faults are charged to the first task that causes the memory to be reserved or faulted, and

all subsequent uses of this reserved or faulted memory is done without charging. Shared HugeTLB memory is only uncharged when it is unreserved or deallocated. This is usually when the HugeTLB file is deleted, and not when the task that caused the reservation or fault has exited.

#### 4. Caveats with HugeTLB cgroup offline.

When a HugeTLB cgroup goes offline with some reservations or faults still charged to it, the behavior is as follows:

- The fault charges are charged to the parent HugeTLB cgroup (reparented),
- the reservation charges remain on the offline HugeTLB cgroup.

This means that if a HugeTLB cgroup gets offlined while there is still HugeTLB reservations charged to it, that cgroup persists as a zombie until all HugeTLB reservations are uncharged. HugeTLB reservations behave in this manner to match the memory controller whose cgroups also persist as zombie until all charged memory is uncharged. Also, the tracking of HugeTLB reservations is a bit more complex compared to the tracking of HugeTLB faults, so it is significantly harder to reparent reservations at offline time.

## 27.8 Memory Resource Controller(Memcg) Implementation Memo

Last Updated: 2010/2

Base Kernel Version: based on 2.6.33-rc7-mm(candidate for 34).

Because VM is getting complex (one of reasons is memcg...), memcg' s behavior is complex. This is a document for memcg' s internal behavior. Please note that implementation details can be changed.

(\*) Topics on API should be in Documentation/admin-guide/cgroup-v1/memory.rst)

### 27.8.1 0. How to record usage ?

2 objects are used.

page\_cgroup ...an object per page.

Allocated at boot or memory hotplug. Freed at memory hot removal.

swap\_cgroup ...an entry per swp\_entry.

Allocated at swapon(). Freed at swapoff().

The page\_cgroup has USED bit and double count against a page\_cgroup never occurs. swap\_cgroup is used only when a charged page is swapped-out.

### 27.8.2 1. Charge

a page/swp\_entry may be charged (usage += PAGE\_SIZE) at  
`mem_cgroup_try_charge()`

### 27.8.3 2. Uncharge

a page/swp\_entry may be uncharged (usage -= PAGE\_SIZE) by

**mem\_cgroup\_uncharge()** Called when a page's refcount goes down to 0.

**mem\_cgroup\_uncharge\_swap()** Called when swp\_entry's refcnt goes down to 0. A charge against swap disappears.

### 27.8.4 3. charge-commit-cancel

Memcg pages are charged in two steps:

- `mem_cgroup_try_charge()`
- `mem_cgroup_commit_charge()` or `mem_cgroup_cancel_charge()`

At `try_charge()`, there are no flags to say "this page is charged" . at this point, `usage += PAGE_SIZE`.

At `commit()`, the page is associated with the memcg.

At `cancel()`, simply `usage -= PAGE_SIZE`.

Under below explanation, we assume `CONFIG_MEM_RES_CTRL_SWAP=y`.

### 27.8.5 4. Anonymous

**Anonymous page is newly allocated at**

- page fault into `MAP_ANONYMOUS` mapping.
- Copy-On-Write.

4.1 Swap-in. At swap-in, the page is taken from swap-cache. There are 2 cases.

- (a) If the SwapCache is newly allocated and read, it has no charges.
- (b) If the SwapCache has been mapped by processes, it has been charged already.

4.2 Swap-out. At swap-out, typical state transition is below.

- (a) add to swap cache. (marked as SwapCache) `swp_entry'` s `refcnt += 1`.
- (b) fully unmapped. `swp_entry'` s `refcnt += # of ptes`.
- (c) write back to swap.

(d) delete from swap cache. (remove from SwapCache) `swp_entry'` s `refcnt -= 1`.

Finally, at task exit, (e) `zap_pte()` is called and `swp_entry'` s `refcnt -=1 -> 0`.

### 27.8.6 5. Page Cache

Page Cache is charged at - `add_to_page_cache_locked()`.

The logic is very clear. (About migration, see below)

**Note:** `__remove_from_page_cache()` is called by `remove_from_page_cache()` and `__remove_mapping()`.

### 27.8.7 6. Shmem(tmpfs) Page Cache

The best way to understand `shmem'` s page state transition is to read `mm/shmem.c`.

But brief explanation of the behavior of `memcg` around `shmem` will be helpful to understand the logic.

`Shmem'` s page (just leaf page, not direct/indirect block) can be on

- radix-tree of `shmem'` s inode.
- SwapCache.
- Both on radix-tree and SwapCache. This happens at swap-in and swap-out,

It' s charged when...

- A new page is added to `shmem'` s radix-tree.
- A `swp` page is read. (move a charge from `swap_cgroup` to `page_cgroup`)

### 27.8.8 7. Page Migration

`mem_cgroup_migrate()`

### 27.8.9 8. LRU

Each `memcg` has its own private LRU. Now, its handling is under global VM' s control (means that it' s handled under global `pgdat->lru_lock`). Almost all routines around `memcg'` s LRU is called by global LRU' s list management functions under `pgdat->lru_lock`.

A special function is `mem_cgroup_isolate_pages()`. This scans `memcg'` s private LRU and call `__isolate_lru_page()` to extract a page from LRU.

(By `__isolate_lru_page()`, the page is removed from both of global and private LRU.)

## 27.8.10 9. Typical Tests.

Tests for racy cases.

### 9.1 Small limit to memcg.

When you do test to do racy case, it' s good test to set memcg' s limit to be very small rather than GB. Many races found in the test under xKB or xxMB limits.

(Memory behavior under GB and Memory behavior under MB shows very different situation.)

### 9.2 Shmem

Historically, memcg' s shmem handling was poor and we saw some amount of troubles here. This is because shmem is page-cache but can be SwapCache. Test with shmem/tmpfs is always good test.

### 9.3 Migration

For NUMA, migration is an another special case. To do easy test, cpuset is useful. Following is a sample script to do migration:

```
mount -t cgroup -o cpuset none /opt/cpuset

mkdir /opt/cpuset/01
echo 1 > /opt/cpuset/01/cpuset.cpus
echo 0 > /opt/cpuset/01/cpuset.mems
echo 1 > /opt/cpuset/01/cpuset.memory_migrate
mkdir /opt/cpuset/02
echo 1 > /opt/cpuset/02/cpuset.cpus
echo 1 > /opt/cpuset/02/cpuset.mems
echo 1 > /opt/cpuset/02/cpuset.memory_migrate
```

In above set, when you moves a task from 01 to 02, page migration to node 0 to node 1 will occur. Following is a script to migrate all under cpuset.:

```
--
move_task()
{
for pid in $1
do
    /bin/echo $pid >$2/tasks 2>/dev/null
    echo -n $pid
    echo -n " "
done
echo END
}

G1_TASK=`cat ${G1}/tasks`
```

(continues on next page)

(continued from previous page)

```
G2_TASK=`cat ${G2}/tasks`  
move_task "${G1_TASK}" ${G2} &  
--
```

## 9.4 Memory hotplug

memory hotplug test is one of good test.

to offline memory, do following:

```
# echo offline > /sys/devices/system/memory/memoryXXX/state
```

(XXX is the place of memory)

This is an easy way to test page migration, too.

## 9.5 mkdir/rmdir

When using hierarchy, mkdir/rmdir test should be done. Use tests like the following:

```
echo 1 >/opt/cgroup/01/memory/use_hierarchy  
mkdir /opt/cgroup/01/child_a  
mkdir /opt/cgroup/01/child_b  
  
set limit to 01.  
add limit to 01/child_b  
run jobs under child_a and child_b
```

create/delete following groups at random while jobs are running:

```
/opt/cgroup/01/child_a/child_aa  
/opt/cgroup/01/child_b/child_bb  
/opt/cgroup/01/child_c
```

running new jobs in new group is also good.

## 9.6 Mount with other subsystems

Mounting with other subsystems is a good test because there is a race and lock dependency with other cgroup subsystems.

example:

```
# mount -t cgroup none /cgroup -o cpuset,memory,cpu,devices
```

and do task move, mkdir, rmdir etc...under this.

## 9.7 swapoff

Besides management of swap is one of complicated parts of memcg, call path of swap-in at swapoff is not same as usual swap-in path.. It's worth to be tested explicitly.

For example, test like following is good:

(Shell-A):

```
# mount -t cgroup none /cgroup -o memory
# mkdir /cgroup/test
# echo 40M > /cgroup/test/memory.limit_in_bytes
# echo 0 > /cgroup/test/tasks
```

Run malloc(100M) program under this. You'll see 60M of swaps.

(Shell-B):

```
# move all tasks in /cgroup/test to /cgroup
# /sbin/swapoff -a
# rmdir /cgroup/test
# kill malloc task.
```

Of course, tmpfs v.s. swapoff test should be tested, too.

## 9.8 OOM-Killer

Out-of-memory caused by memcg's limit will kill tasks under the memcg. When hierarchy is used, a task under hierarchy will be killed by the kernel.

In this case, panic\_on\_oom shouldn't be invoked and tasks in other groups shouldn't be killed.

It's not difficult to cause OOM under memcg as following.

Case A) when you can swapoff:

```
#swapoff -a
#echo 50M > /memory.limit_in_bytes
```

run 51M of malloc

Case B) when you use mem+swap limitation:

```
#echo 50M > memory.limit_in_bytes
#echo 50M > memory.memsw.limit_in_bytes
```

run 51M of malloc

### 9.9 Move charges at task migration

Charges associated with a task can be moved along with task migration.

(Shell-A):

```
#mkdir /cgroup/A
#echo $$ >/cgroup/A/tasks
```

run some programs which uses some amount of memory in /cgroup/A.

(Shell-B):

```
#mkdir /cgroup/B
#echo 1 >/cgroup/B/memory.move_charge_at_immigrate
#echo "pid of the program running in group A" >/cgroup/B/tasks
```

You can see charges have been moved by reading `*.usage_in_bytes` or `memory.stat` of both A and B.

See 8.2 of `Documentation/admin-guide/cgroup-v1/memory.rst` to see what value should be written to `move_charge_at_immigrate`.

### 9.10 Memory thresholds

Memory controller implements memory thresholds using cgroups notification API. You can use `tools/cgroup/cgroup_event_listener.c` to test it.

(Shell-A) Create cgroup and run event listener:

```
# mkdir /cgroup/A
# ./cgroup_event_listener /cgroup/A/memory.usage_in_bytes 5M
```

(Shell-B) Add task to cgroup and try to allocate and free memory:

```
# echo $$ >/cgroup/A/tasks
# a="$(dd if=/dev/zero bs=1M count=10)"
# a=
```

You will see message from `cgroup_event_listener` every time you cross the thresholds.

Use `/cgroup/A/memory.memsw.usage_in_bytes` to test memsw thresholds.

It's good idea to test root cgroup as well.

## 27.9 Memory Resource Controller

**NOTE:** This document is hopelessly outdated and it asks for a complete rewrite. It still contains a useful information so we are keeping it here but make sure to check the current code if you need a deeper understanding.

**NOTE:** The Memory Resource Controller has generically been referred to as the memory controller in this document. Do not confuse memory controller used here with the memory controller that is used in hardware.

**(For editors) In this document:** When we mention a cgroup (cgroupfs' s directory) with memory controller, we call it “memory cgroup” . When you see git-log and source code, you' ll see patch' s title and function names tend to use “memcg” . In this document, we avoid using it.

### 27.9.1 Benefits and Purpose of the memory controller

The memory controller isolates the memory behaviour of a group of tasks from the rest of the system. The article on LWN [12] mentions some probable uses of the memory controller. The memory controller can be used to

- a. Isolate an application or a group of applications Memory-hungry applications can be isolated and limited to a smaller amount of memory.
- b. Create a cgroup with a limited amount of memory; this can be used as a good alternative to booting with mem=XXXX.
- c. Virtualization solutions can control the amount of memory they want to assign to a virtual machine instance.
- d. A CD/DVD burner could control the amount of memory used by the rest of the system to ensure that burning does not fail due to lack of available memory.
- e. There are several other use cases; find one or use the controller just for fun (to learn and hack on the VM subsystem).

Current Status: linux-2.6.34-mmotm(development version of 2010/April)

Features:

- accounting anonymous pages, file caches, swap caches usage and limiting them.
- pages are linked to per-memcg LRU exclusively, and there is no global LRU.
- optionally, memory+swap usage can be accounted and limited.
- hierarchical accounting
- soft limit
- moving (recharging) account at moving a task is selectable.
- usage threshold notifier
- memory pressure notifier
- oom-killer disable knob and oom-notifier

- Root cgroup has no limit controls.

Kernel memory support is a work in progress, and the current version provides basically functionality. (See Section 2.7)

Brief summary of control files.

tasks	attach a task(thread) and show list of threads
cgroup.procs	show list of processes
cgroup.event_control	kernel interface for event_fd()
memory.usage_in_bytes	show current usage for memory (See 5.5 for details)
memory.memsw.usage_in_bytes	show current usage for memory+Swap (See 5.5 for details)
memory.limit_in_bytes	set/show limit of memory usage
memory.memsw.limit_in_bytes	set/show limit of memory+Swap usage
memory.failcnt	show the number of memory usage hits limits
memory.memsw.failcnt	show the number of memory+Swap hits limits
memory.max_usage_in_bytes	show max memory usage recorded
memory.memsw.max_usage_in_bytes	show max memory+Swap usage recorded
memory.soft_limit_in_bytes	set/show soft limit of memory usage
memory.stat	show various statistics
memory.use_hierarchy	set/show hierarchical account enabled
memory.force_empty	trigger forced page reclaim
memory.pressure_level	set memory pressure notifications
memory.swappiness	set/show swappiness parameter of vmscan (See sysctl' s vm.swappiness)
memory.move_charge_at_immigrate	set/show controls of moving charges
memory.oom_control	set/show oom controls.
memory.numa_stat	show the number of memory usage per numa node
memory.kmem.limit_in_bytes	set/show hard limit for kernel memory This knob is deprecated and shouldn' t be used. It is planned that this be removed in the foreseeable future.
memory.kmem.usage_in_bytes	show current kernel memory allocation
memory.kmem.failcnt	show the number of kernel memory usage hits limits
memory.kmem.max_usage_in_bytes	show max kernel memory usage recorded
memory.kmem.tcp.limit_in_bytes	set/show hard limit for tcp buf memory
memory.kmem.tcp.usage_in_bytes	show current tcp buf memory allocation
memory.kmem.tcp.failcnt	show the number of tcp buf memory usage hits limits
memory.kmem.tcp.max_usage_in_bytes	show max tcp buf memory usage recorded

### 27.9.2 1. History

The memory controller has a long history. A request for comments for the memory controller was posted by Balbir Singh [1]. At the time the RFC was posted there were several implementations for memory control. The goal of the RFC was to build consensus and agreement for the minimal features required for memory control. The first RSS controller was posted by Balbir Singh[2] in Feb 2007. Pavel Emelianov [3][4][5] has since posted three versions of the RSS controller. At OLS, at the resource management BoF, everyone suggested that we handle both page cache and RSS together. Another request was raised to allow user space handling of OOM. The current memory controller is at version 6; it combines both mapped (RSS) and unmapped Page Cache Control [11].

### 27.9.3 2. Memory Control

Memory is a unique resource in the sense that it is present in a limited amount. If a task requires a lot of CPU processing, the task can spread its processing over a period of hours, days, months or years, but with memory, the same physical memory needs to be reused to accomplish the task.

The memory controller implementation has been divided into phases. These are:

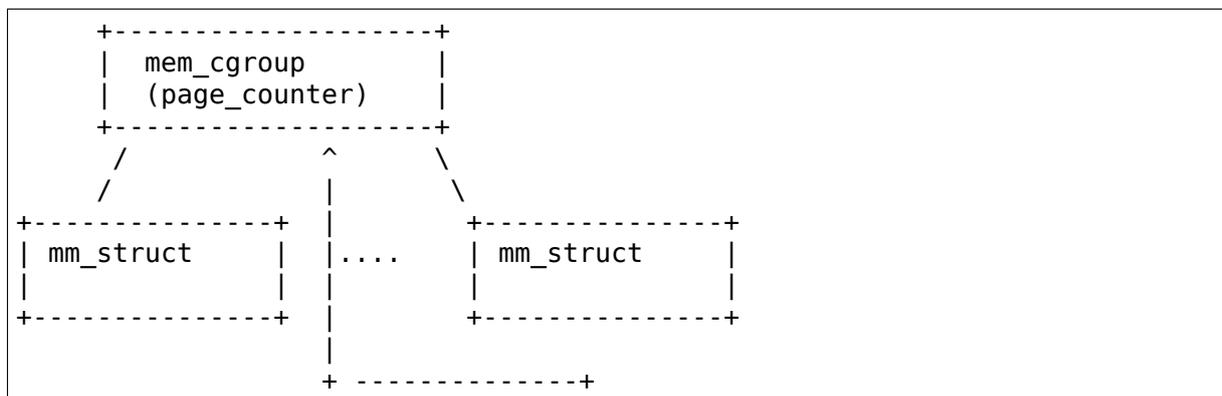
1. Memory controller
2. mlock(2) controller
3. Kernel user memory accounting and slab control
4. user mappings length controller

The memory controller is the first controller developed.

#### 2.1. Design

The core of the design is a counter called the page\_counter. The page\_counter tracks the current memory usage and limit of the group of processes associated with the controller. Each cgroup has a memory controller specific data structure (mem\_cgroup) associated with it.

#### 2.2. Accounting



(continues on next page)

(continued from previous page)

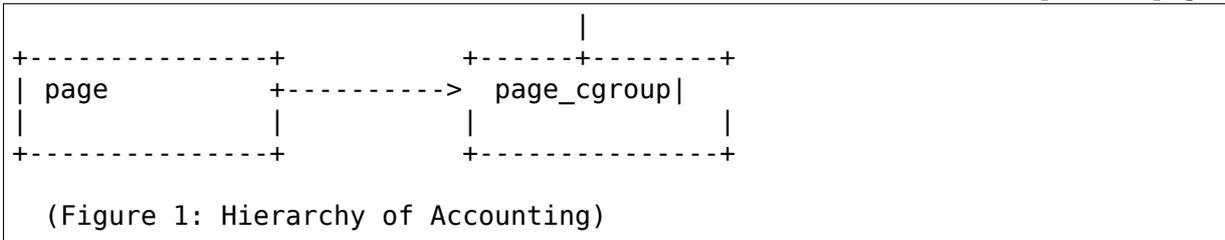


Figure 1 shows the important aspects of the controller

1. Accounting happens per cgroup
2. Each `mm_struct` knows about which cgroup it belongs to
3. Each page has a pointer to the `page_cgroup`, which in turn knows the cgroup it belongs to

The accounting is done as follows: `mem_cgroup_charge_common()` is invoked to set up the necessary data structures and check if the cgroup that is being charged is over its limit. If it is, then reclaim is invoked on the cgroup. More details can be found in the reclaim section of this document. If everything goes well, a page meta-data-structure called `page_cgroup` is updated. `page_cgroup` has its own LRU on cgroup. (\*) `page_cgroup` structure is allocated at boot/memory-hotplug time.

### 2.2.1 Accounting details

All mapped anon pages (RSS) and cache pages (Page Cache) are accounted. Some pages which are never reclaimable and will not be on the LRU are not accounted. We just account pages under usual VM management.

RSS pages are accounted at `page_fault` unless they've already been accounted for earlier. A file page will be accounted for as Page Cache when it's inserted into inode (radix-tree). While it's mapped into the page tables of processes, duplicate accounting is carefully avoided.

An RSS page is unaccounted when it's fully unmapped. A PageCache page is unaccounted when it's removed from radix-tree. Even if RSS pages are fully unmapped (by `kswapd`), they may exist as SwapCache in the system until they are really freed. Such SwapCaches are also accounted. A swapped-in page is accounted after adding into swapcache.

Note: The kernel does `swpin-readahead` and reads multiple swaps at once. Since `page's memcg` recorded into swap whatever `memsw` enabled, the page will be accounted after `swpin`.

At page migration, accounting information is kept.

Note: we just account pages-on-LRU because our purpose is to control amount of used pages; not-on-LRU pages tend to be out-of-control from VM view.

### 2.3 Shared Page Accounting

Shared pages are accounted on the basis of the first touch approach. The cgroup that first touches a page is accounted for the page. The principle behind this approach is that a cgroup that aggressively uses a shared page will eventually get charged for it (once it is uncharged from the cgroup that brought it in - this will happen on memory pressure).

But see section 8.2: when moving a task to another cgroup, its pages may be recharged to the new cgroup, if `move_charge_at_immigrate` has been chosen.

### 2.4 Swap Extension

Swap usage is always recorded for each of cgroup. Swap Extension allows you to read and limit it.

When `CONFIG_SWAP` is enabled, following files are added.

- `memory.memsw.usage_in_bytes`.
- `memory.memsw.limit_in_bytes`.

`memsw` means `memory+swap`. Usage of `memory+swap` is limited by `memsw.limit_in_bytes`.

Example: Assume a system with 4G of swap. A task which allocates 6G of memory (by mistake) under 2G memory limitation will use all swap. In this case, setting `memsw.limit_in_bytes=3G` will prevent bad use of swap. By using the `memsw` limit, you can avoid system OOM which can be caused by swap shortage.

#### why 'memory+swap' rather than swap

The global LRU(`kswapd`) can swap out arbitrary pages. Swap-out means to move account from memory to swap...there is no change in usage of `memory+swap`. In other words, when we want to limit the usage of swap without affecting global LRU, `memory+swap` limit is better than just limiting swap from an OS point of view.

#### What happens when a cgroup hits `memory.memsw.limit_in_bytes`

When a cgroup hits `memory.memsw.limit_in_bytes`, it's useless to do swap-out in this cgroup. Then, swap-out will not be done by cgroup routine and file caches are dropped. But as mentioned above, global LRU can do swapout memory from it for sanity of the system's memory management state. You can't forbid it by cgroup.

### 2.5 Reclaim

Each cgroup maintains a per cgroup LRU which has the same structure as global VM. When a cgroup goes over its limit, we first try to reclaim memory from the cgroup so as to make space for the new pages that the cgroup has touched. If the reclaim is unsuccessful, an OOM routine is invoked to select and kill the bulkiest task in the cgroup. (See 10. OOM Control below.)

The reclaim algorithm has not been modified for cgroups, except that pages that are selected for reclaiming come from the per-cgroup LRU list.

**NOTE:** Reclaim does not work for the root cgroup, since we cannot set any limits on the root cgroup.

**Note2:** When `panic_on_oom` is set to “2”, the whole system will panic.

When oom event notifier is registered, event will be delivered. (See `oom_control` section)

## 2.6 Locking

`lock_page_cgroup()/unlock_page_cgroup()` should not be called under the `i_pages` lock.

Other lock order is following:

**PG\_locked.**

**mm->page\_table\_lock**

**pgdat->lru\_lock** `lock_page_cgroup`.

In many cases, just `lock_page_cgroup()` is called.

per-zone-per-cgroup LRU (cgroup’s private LRU) is just guarded by `pgdat->lru_lock`, it has no lock of its own.

## 2.7 Kernel Memory Extension (CONFIG\_MEMCG\_KMEM)

With the Kernel memory extension, the Memory Controller is able to limit the amount of kernel memory used by the system. Kernel memory is fundamentally different than user memory, since it can’t be swapped out, which makes it possible to DoS the system by consuming too much of this precious resource.

Kernel memory accounting is enabled for all memory cgroups by default. But it can be disabled system-wide by passing `cgroup.memory=nokmem` to the kernel at boot time. In this case, kernel memory will not be accounted at all.

Kernel memory limits are not imposed for the root cgroup. Usage for the root cgroup may or may not be accounted. The memory used is accumulated into `memory.kmem.usage_in_bytes`, or in a separate counter when it makes sense. (currently only for tcp).

The main “kmem” counter is fed into the main counter, so kmem charges will also be visible from the user counter.

Currently no soft limit is implemented for kernel memory. It is future work to trigger slab reclaim when those limits are reached.

### 2.7.1 Current Kernel Memory resources accounted

**stack pages:** every process consumes some stack pages. By accounting into kernel memory, we prevent new processes from being created when the kernel memory usage is too high.

**slab pages:** pages allocated by the SLAB or SLUB allocator are tracked. A copy of each `kmem_cache` is created every time the cache is touched by the first time from inside the memcg. The creation is done lazily, so some objects can still be skipped while the cache is being created. All objects in a slab page should belong to the same memcg. This only fails to hold when a task is migrated to a different memcg during the page allocation by the cache.

**sockets memory pressure:** some sockets protocols have memory pressure thresholds. The Memory Controller allows them to be controlled individually per cgroup, instead of globally.

**tcp memory pressure:** sockets memory pressure for the tcp protocol.

### 2.7.2 Common use cases

Because the “kmem” counter is fed to the main user counter, kernel memory can never be limited completely independently of user memory. Say “U” is the user limit, and “K” the kernel limit. There are three possible ways limits can be set:

**U != 0, K = unlimited:** This is the standard memcg limitation mechanism already present before kmem accounting. Kernel memory is completely ignored.

**U != 0, K < U:** Kernel memory is a subset of the user memory. This setup is useful in deployments where the total amount of memory per-cgroup is overcommitted. Overcommitting kernel memory limits is definitely not recommended, since the box can still run out of non-reclaimable memory. In this case, the admin could set up K so that the sum of all groups is never greater than the total memory, and freely set U at the cost of his QoS.

**WARNING:** In the current implementation, memory reclaim will NOT be triggered for a cgroup when it hits K while staying below U, which makes this setup impractical.

**U != 0, K >= U:** Since kmem charges will also be fed to the user counter and reclaim will be triggered for the cgroup for both kinds of memory. This setup gives the admin a unified view of memory, and it is also useful for people who just want to track kernel memory usage.

## 27.9.4 3. User Interface

### 3.0. Configuration

- a. Enable CONFIG\_CGROUPS
- b. Enable CONFIG\_MEMCG
- c. Enable CONFIG\_MEMCG\_SWAP (to use swap extension)
- d. Enable CONFIG\_MEMCG\_KMEM (to use kmem extension)

### 3.1. Prepare the cgroups (see cgroups.txt, Why are cgroups needed?)

```
# mount -t tmpfs none /sys/fs/cgroup
# mkdir /sys/fs/cgroup/memory
# mount -t cgroup none /sys/fs/cgroup/memory -o memory
```

3.2. Make the new group and move bash into it:

```
# mkdir /sys/fs/cgroup/memory/0
# echo $$ > /sys/fs/cgroup/memory/0/tasks
```

Since now we' re in the 0 cgroup, we can alter the memory limit:

```
# echo 4M > /sys/fs/cgroup/memory/0/memory.limit_in_bytes
```

**NOTE:** We can use a suffix (k, K, m, M, g or G) to indicate values in kilo, mega or gigabytes. (Here, Kilo, Mega, Giga are Kibibytes, Mebibytes, Gibibytes.)

**NOTE:** We can write “-1” to reset the \*.limit\_in\_bytes(unlimited).

**NOTE:** We cannot set limits on the root cgroup any more.

```
# cat /sys/fs/cgroup/memory/0/memory.limit_in_bytes
4194304
```

We can check the usage:

```
# cat /sys/fs/cgroup/memory/0/memory.usage_in_bytes
1216512
```

A successful write to this file does not guarantee a successful setting of this limit to the value written into the file. This can be due to a number of factors, such as rounding up to page boundaries or the total availability of memory on the system. The user is required to re-read this file after a write to guarantee the value committed by the kernel:

```
# echo 1 > memory.limit_in_bytes
# cat memory.limit_in_bytes
4096
```

The memory.failcnt field gives the number of times that the cgroup limit was exceeded.

The memory.stat file gives accounting information. Now, the number of caches, RSS and Active pages/Inactive pages are shown.

### 27.9.5 4. Testing

For testing features and implementation, see memcg\_test.txt.

Performance test is also important. To see pure memory controller' s overhead, testing on tmpfs will give you good numbers of small overheads. Example: do kernel make on tmpfs.

Page-fault scalability is also important. At measuring parallel page fault test, multi-process test may be better than multi-thread test because it has noise of shared objects/status.

But the above two are testing extreme situations. Trying usual test under memory controller is always helpful.

### 4.1 Troubleshooting

Sometimes a user might find that the application under a cgroup is terminated by the OOM killer. There are several causes for this:

1. The cgroup limit is too low (just too low to do anything useful)
2. The user is using anonymous memory and swap is turned off or too low

A sync followed by `echo 1 > /proc/sys/vm/drop_caches` will help get rid of some of the pages cached in the cgroup (page cache pages).

To know what happens, disabling OOM\_Kill as per “10. OOM Control” (below) and seeing what happens will be helpful.

### 4.2 Task migration

When a task migrates from one cgroup to another, its charge is not carried forward by default. The pages allocated from the original cgroup still remain charged to it, the charge is dropped when the page is freed or reclaimed.

You can move charges of a task along with task migration. See 8. “Move charges at task migration”

### 4.3 Removing a cgroup

A cgroup can be removed by `rmdir`, but as discussed in sections 4.1 and 4.2, a cgroup might have some charge associated with it, even though all tasks have migrated away from it. (because we charge against pages, not against tasks.)

We move the stats to root (if `use_hierarchy==0`) or parent (if `use_hierarchy==1`), and no change on the charge except uncharging from the child.

Charges recorded in swap information is not updated at removal of cgroup. Recorded information is discarded and a cgroup which uses swap (swapcache) will be charged as a new owner of it.

About `use_hierarchy`, see Section 6.

## 27.9.6 5. Misc. interfaces

### 5.1 `force_empty`

`memory.force_empty` interface is provided to make cgroup' s memory usage empty. When writing anything to this:

```
# echo 0 > memory.force_empty
```

the cgroup will be reclaimed and as many pages reclaimed as possible.

The typical use case for this interface is before calling `rmdir()`. Though `rmdir()` offlines `memcg`, but the `memcg` may still stay there due to charged file caches. Some out-of-use page caches may keep charged until memory pressure happens. If you want to avoid that, `force_empty` will be useful.

Also, note that when `memory.kmem.limit_in_bytes` is set the charges due to kernel pages will still be seen. This is not considered a failure and the write will still return success. In this case, it is expected that `memory.kmem.usage_in_bytes == memory.usage_in_bytes`.

About `use_hierarchy`, see Section 6.

### 5.2 `stat` file

`memory.stat` file includes following statistics

## per-memory cgroup local status

cache	# of bytes of page cache memory.
rss	# of bytes of anonymous and swap cache memory (includes transparent hugepages).
rss_huge	# of bytes of anonymous transparent hugepages.
mapped_file	# of bytes of mapped file (includes tmpfs/shmem)
pgp-gin	# of charging events to the memory cgroup. The charging event happens each time a page is accounted as either mapped anon page(RSS) or cache page(Page Cache) to the cgroup.
pgp-gout	# of uncharging events to the memory cgroup. The uncharging event happens each time a page is unaccounted from the cgroup.
swap	# of bytes of swap usage
dirty	# of bytes that are waiting to get written back to the disk.
write-back	# of bytes of file/anon cache that are queued for syncing to disk.
inactive_anon	# of bytes of anonymous and swap cache memory on inactive LRU list.
active_anon	# of bytes of anonymous and swap cache memory on active LRU list.
inactive_file	# of bytes of file-backed memory on inactive LRU list.
active_file	# of bytes of file-backed memory on active LRU list.
unevictable	# of bytes of memory that cannot be reclaimed (mlocked etc).

## status considering hierarchy (see memory.use\_hierarchy settings)

hierarchical_memory_limit	# of bytes of memory limit with regard to hierarchy under which the memory cgroup is
hierarchical_memsw_limit	# of bytes of memory+swap limit with regard to hierarchy under which memory cgroup is.
total_<counter>	# hierarchical version of <counter>, which in addition to the cgroup' s own value includes the sum of all hierarchical children' s values of <counter>, i.e. total_cache

## The following additional stats are dependent on CONFIG\_DEBUG\_VM

recent_rotated_anon	VM internal parameter. (see mm/vmscan.c)
recent_rotated_file	VM internal parameter. (see mm/vmscan.c)
recent_scanned_anon	VM internal parameter. (see mm/vmscan.c)
recent_scanned_file	VM internal parameter. (see mm/vmscan.c)

**Memo:** recent\_rotated means recent frequency of LRU rotation. recent\_scanned means recent # of scans to LRU. showing for better debug please see the

code for meanings.

**Note:** Only anonymous and swap cache memory is listed as part of ‘rss’ stat. This should not be confused with the true ‘resident set size’ or the amount of physical memory used by the cgroup.

‘rss + mapped\_file’ will give you resident set size of cgroup.

(Note: file and shmem may be shared among other cgroups. In that case, mapped\_file is accounted only when the memory cgroup is owner of page cache.)

### 5.3 swappiness

Overrides /proc/sys/vm/swappiness for the particular group. The tunable in the root cgroup corresponds to the global swappiness setting.

Please note that unlike during the global reclaim, limit reclaim enforces that 0 swappiness really prevents from any swapping even if there is a swap storage available. This might lead to memcg OOM killer if there are no file pages to reclaim.

### 5.4 failcnt

A memory cgroup provides memory.failcnt and memory.memsw.failcnt files. This failcnt(== failure count) shows the number of times that a usage counter hit its limit. When a memory cgroup hits a limit, failcnt increases and memory under it will be reclaimed.

You can reset failcnt by writing 0 to failcnt file:

```
# echo 0 > ../memory.failcnt
```

### 5.5 usage\_in\_bytes

For efficiency, as other kernel components, memory cgroup uses some optimization to avoid unnecessary cacheline false sharing. usage\_in\_bytes is affected by the method and doesn’t show ‘exact’ value of memory (and swap) usage, it’s a fuzz value for efficient access. (Of course, when necessary, it’s synchronized.) If you want to know more exact memory usage, you should use RSS+CACHE(+SWAP) value in memory.stat(see 5.2).

### 5.6 numa\_stat

This is similar to `numa_maps` but operates on a per-memcg basis. This is useful for providing visibility into the numa locality information within an memcg since the pages are allowed to be allocated from any physical node. One of the use cases is evaluating application performance by combining this information with the application's CPU allocation.

Each memcg's `numa_stat` file includes "total", "file", "anon" and "unevictable" per-node page counts including "hierarchical\_<counter>" which sums up all hierarchical children's values in addition to the memcg's own value.

The output format of `memory.numa_stat` is:

```
total=<total pages> N0=<node 0 pages> N1=<node 1 pages> ...
file=<total file pages> N0=<node 0 pages> N1=<node 1 pages> ...
anon=<total anon pages> N0=<node 0 pages> N1=<node 1 pages> ...
unevictable=<total anon pages> N0=<node 0 pages> N1=<node 1 pages> ...
hierarchical_<counter>=<counter pages> N0=<node 0 pages> N1=<node 1 pages>
↳ ...
```

The "total" count is sum of file + anon + unevictable.

### 27.9.7 6. Hierarchy support

The memory controller supports a deep hierarchy and hierarchical accounting. The hierarchy is created by creating the appropriate cgroups in the cgroup filesystem. Consider for example, the following cgroup filesystem hierarchy:



In the diagram above, with hierarchical accounting enabled, all memory usage of `e`, is accounted to its ancestors up until the root (i.e. `c` and `root`), that has `memory.use_hierarchy` enabled. If one of the ancestors goes over its limit, the reclaim algorithm reclaims from the tasks in the ancestor and the children of the ancestor.

#### 6.1 Enabling hierarchical accounting and reclaim

A memory cgroup by default disables the hierarchy feature. Support can be enabled by writing 1 to `memory.use_hierarchy` file of the root cgroup:

```
# echo 1 > memory.use_hierarchy
```

The feature can be disabled by:

```
# echo 0 > memory.use_hierarchy
```

**NOTE1:** Enabling/disabling will fail if either the cgroup already has other cgroups created below it, or if the parent cgroup has `use_hierarchy` enabled.

**NOTE2:** When `panic_on_oom` is set to “2”, the whole system will panic in case of an OOM event in any cgroup.

### 27.9.8 7. Soft limits

Soft limits allow for greater sharing of memory. The idea behind soft limits is to allow control groups to use as much of the memory as needed, provided

- a. There is no memory contention
- b. They do not exceed their hard limit

When the system detects memory contention or low memory, control groups are pushed back to their soft limits. If the soft limit of each control group is very high, they are pushed back as much as possible to make sure that one control group does not starve the others of memory.

Please note that soft limits is a best-effort feature; it comes with no guarantees, but it does its best to make sure that when memory is heavily contended for, memory is allocated based on the soft limit hints/setup. Currently soft limit based reclaim is set up such that it gets invoked from `balance_pgdat` (`kswapd`).

#### 7.1 Interface

Soft limits can be setup by using the following commands (in this example we assume a soft limit of 256 MiB):

```
# echo 256M > memory.soft_limit_in_bytes
```

If we want to change this to 1G, we can at any time use:

```
# echo 1G > memory.soft_limit_in_bytes
```

**NOTE1:** Soft limits take effect over a long period of time, since they involve reclaiming memory for balancing between memory cgroups

**NOTE2:** It is recommended to set the soft limit always below the hard limit, otherwise the hard limit will take precedence.

### 27.9.9 8. Move charges at task migration

Users can move charges associated with a task along with task migration, that is, uncharge task’s pages from the old cgroup and charge them to the new cgroup. This feature is not supported in `!CONFIG_MMU` environments because of lack of page tables.

### 8.1 Interface

This feature is disabled by default. It can be enabled (and disabled again) by writing to `memory.move_charge_at_immigrate` of the destination cgroup.

If you want to enable it:

```
# echo (some positive value) > memory.move_charge_at_immigrate
```

**Note:** Each bits of `move_charge_at_immigrate` has its own meaning about what type of charges should be moved. See 8.2 for details.

**Note:** Charges are moved only when you move `mm->owner`, in other words, a leader of a thread group.

**Note:** If we cannot find enough space for the task in the destination cgroup, we try to make space by reclaiming memory. Task migration may fail if we cannot make enough space.

**Note:** It can take several seconds if you move charges much.

And if you want disable it again:

```
# echo 0 > memory.move_charge_at_immigrate
```

### 8.2 Type of charges which can be moved

Each bit in `move_charge_at_immigrate` has its own meaning about what type of charges should be moved. But in any case, it must be noted that an account of a page or a swap can be moved only when it is charged to the task's current (old) memory cgroup.

bit	what type of charges would be moved ?
0	A charge of an anonymous page (or swap of it) used by the target task. You must enable Swap Extension (see 2.4) to enable move of swap charges.
1	A charge of file pages (normal file, tmpfs file (e.g. ipc shared memory) and swaps of tmpfs file) mmaped by the target task. Unlike the case of anonymous pages, file pages (and swaps) in the range mmaped by the task will be moved even if the task hasn't done page fault, i.e. they might not be the task's "RSS", but other task's "RSS" that maps the same file. And mapcount of the page is ignored (the page can be moved even if <code>page_mapcount(page) &gt; 1</code> ). You must enable Swap Extension (see 2.4) to enable move of swap charges.

### 8.3 TODO

- All of moving charge operations are done under `cgroup_mutex`. It's not good behavior to hold the mutex too long, so we may need some trick.

#### 27.9.10 9. Memory thresholds

Memory cgroup implements memory thresholds using the cgroups notification API (see `cgroups.txt`). It allows to register multiple memory and memsw thresholds and gets notifications when it crosses.

To register a threshold, an application must:

- create an eventfd using `eventfd(2)`;
- open `memory.usage_in_bytes` or `memory.memsw.usage_in_bytes`;
- write string like “`<event_fd> <fd of memory.usage_in_bytes> <threshold>`” to `cgroup.event_control`.

Application will be notified through eventfd when memory usage crosses threshold in any direction.

It's applicable for root and non-root cgroup.

#### 27.9.11 10. OOM Control

`memory.oom_control` file is for OOM notification and other controls.

Memory cgroup implements OOM notifier using the cgroup notification API (See `cgroups.txt`). It allows to register multiple OOM notification delivery and gets notification when OOM happens.

To register a notifier, an application must:

- create an eventfd using `eventfd(2)`
- open `memory.oom_control` file
- write string like “`<event_fd> <fd of memory.oom_control>`” to `cgroup.event_control`

The application will be notified through eventfd when OOM happens. OOM notification doesn't work for the root cgroup.

You can disable the OOM-killer by writing “1” to `memory.oom_control` file, as:

```
#echo 1 > memory.oom_control
```

If OOM-killer is disabled, tasks under cgroup will hang/sleep in memory cgroup's OOM-waitqueue when they request accountable memory.

For running them, you have to relax the memory cgroup's OOM status by

- enlarge limit or reduce usage.

To reduce usage,

- kill some tasks.

- move some tasks to other group with account migration.
- remove some files (on tmpfs?)

Then, stopped tasks will work again.

At reading, current status of OOM is shown.

- `oom_kill_disable` 0 or 1 (if 1, oom-killer is disabled)
- `under_oom` 0 or 1 (if 1, the memory cgroup is under OOM, tasks may be stopped.)

### 27.9.12 11. Memory Pressure

The pressure level notifications can be used to monitor the memory allocation cost; based on the pressure, applications can implement different strategies of managing their memory resources. The pressure levels are defined as following:

The “low” level means that the system is reclaiming memory for new allocations. Monitoring this reclaiming activity might be useful for maintaining cache level. Upon notification, the program (typically “Activity Manager”) might analyze `vmstat` and act in advance (i.e. prematurely shutdown unimportant services).

The “medium” level means that the system is experiencing medium memory pressure, the system might be making swap, paging out active file caches, etc. Upon this event applications may decide to further analyze `vmstat/zoneinfo/memcg` or internal memory usage statistics and free any resources that can be easily reconstructed or re-read from a disk.

The “critical” level means that the system is actively thrashing, it is about to out of memory (OOM) or even the in-kernel OOM killer is on its way to trigger. Applications should do whatever they can to help the system. It might be too late to consult with `vmstat` or any other statistics, so it’s advisable to take an immediate action.

By default, events are propagated upward until the event is handled, i.e. the events are not pass-through. For example, you have three cgroups: `A->B->C`. Now you set up an event listener on cgroups A, B and C, and suppose group C experiences some pressure. In this situation, only group C will receive the notification, i.e. groups A and B will not receive it. This is done to avoid excessive “broadcasting” of messages, which disturbs the system and which is especially bad if we are low on memory or thrashing. Group B, will receive notification only if there are no event listeners for group C.

There are three optional modes that specify different propagation behavior:

- “default” : this is the default behavior specified above. This mode is the same as omitting the optional mode parameter, preserved by backwards compatibility.
- “hierarchy” : events always propagate up to the root, similar to the default behavior, except that propagation continues regardless of whether there are event listeners at each level, with the “hierarchy” mode. In the above example, groups A, B, and C will receive notification of memory pressure.

- “local” : events are pass-through, i.e. they only receive notifications when memory pressure is experienced in the memcg for which the notification is registered. In the above example, group C will receive notification if registered for “local” notification and the group experiences memory pressure. However, group B will never receive notification, regardless if there is an event listener for group C or not, if group B is registered for local notification.

The level and event notification mode ( “hierarchy” or “local” , if necessary) are specified by a comma-delimited string, i.e. “low,hierarchy” specifies hierarchical, pass-through, notification for all ancestor memcgs. Notification that is the default, non pass-through behavior, does not specify a mode. “medium,local” specifies pass-through notification for the medium level.

The file `memory.pressure_level` is only used to setup an eventfd. To register a notification, an application must:

- create an eventfd using `eventfd(2)`;
- open `memory.pressure_level`;
- write string as “<event\_fd> <fd of memory.pressure\_level> <level[,mode]>” to `cgroup.event_control`.

Application will be notified through eventfd when memory pressure is at the specific level (or higher). Read/write operations to `memory.pressure_level` are not implemented.

Test:

Here is a small script example that makes a new cgroup, sets up a memory limit, sets up a notification in the cgroup and then makes child cgroup experience a critical pressure:

```
# cd /sys/fs/cgroup/memory/
# mkdir foo
# cd foo
# cgroup_event_listener memory.pressure_level low,hierarchy &
# echo 8000000 > memory.limit_in_bytes
# echo 8000000 > memory.memsw.limit_in_bytes
# echo $$ > tasks
# dd if=/dev/zero | read x
```

(Expect a bunch of notifications, and eventually, the oom-killer will trigger.)

### 27.9.13 12. TODO

1. Make per-cgroup scanner reclaim not-shared pages first
2. Teach controller to account for shared-pages
3. Start reclamation in the background when the limit is not yet hit but the usage is getting closer

### 27.9.14 Summary

Overall, the memory controller has been a stable controller and has been commented and discussed quite extensively in the community.

### 27.9.15 References

1. Singh, Balbir. RFC: Memory Controller, <http://lwn.net/Articles/206697/>
2. Singh, Balbir. Memory Controller (RSS Control), <http://lwn.net/Articles/222762/>
3. Emelianov, Pavel. Resource controllers based on process cgroups <http://lkml.org/lkml/2007/3/6/198>
4. Emelianov, Pavel. RSS controller based on process cgroups (v2) <http://lkml.org/lkml/2007/4/9/78>
5. Emelianov, Pavel. RSS controller based on process cgroups (v3) <http://lkml.org/lkml/2007/5/30/244>
6. Menage, Paul. Control Groups v10, <http://lwn.net/Articles/236032/>
7. Vaidyanathan, Srinivasan, Control Groups: Pagecache accounting and control subsystem (v3), <http://lwn.net/Articles/235534/>
8. Singh, Balbir. RSS controller v2 test results (lmbench), <http://lkml.org/lkml/2007/5/17/232>
9. Singh, Balbir. RSS controller v2 AIM9 results <http://lkml.org/lkml/2007/5/18/1>
10. Singh, Balbir. Memory controller v6 test results, <http://lkml.org/lkml/2007/8/19/36>
11. Singh, Balbir. Memory controller introduction (v6), <http://lkml.org/lkml/2007/8/17/69>
12. Corbet, Jonathan, Controlling memory use in cgroups, <http://lwn.net/Articles/243795/>

## 27.10 Network classifier cgroup

The Network classifier cgroup provides an interface to tag network packets with a class identifier (classid).

The Traffic Controller (tc) can be used to assign different priorities to packets from different cgroups. Also, Netfilter (iptables) can use this tag to perform actions on such packets.

Creating a `net_cls` cgroups instance creates a `net_cls.classid` file. This `net_cls.classid` value is initialized to 0.

You can write hexadecimal values to `net_cls.classid`; the format for these values is `0xAAAABBBB`; AAAA is the major handle number and BBBB is the minor handle number. Reading `net_cls.classid` yields a decimal result.

Example:

```
mkdir /sys/fs/cgroup/net_cls
mount -t cgroup -onet_cls net_cls /sys/fs/cgroup/net_cls
mkdir /sys/fs/cgroup/net_cls/0
echo 0x100001 > /sys/fs/cgroup/net_cls/0/net_cls.classid
```

- setting a 10:1 handle:

```
cat /sys/fs/cgroup/net_cls/0/net_cls.classid
1048577
```

- configuring tc:

```
tc qdisc add dev eth0 root handle 10: htb
tc class add dev eth0 parent 10: classid 10:1 htb rate 40mbit
```

- creating traffic class 10:1:

```
tc filter add dev eth0 parent 10: protocol ip prio 10 handle 1: cgroup
```

configuring iptables, basic example:

```
iptables -A OUTPUT -m cgroup ! --cgroup 0x100001 -j DROP
```

## 27.11 Network priority cgroup

The Network priority cgroup provides an interface to allow an administrator to dynamically set the priority of network traffic generated by various applications

Nominally, an application would set the priority of its traffic via the `SO_PRIORITY` socket option. This however, is not always possible because:

- 1) The application may not have been coded to set this value
- 2) The priority of application traffic is often a site-specific administrative decision rather than an application defined one.

This cgroup allows an administrator to assign a process to a group which defines the priority of egress traffic on a given interface. Network priority groups can be created by first mounting the cgroup filesystem:

```
# mount -t cgroup -onet_prio none /sys/fs/cgroup/net_prio
```

With the above step, the initial group acting as the parent accounting group becomes visible at `/sys/fs/cgroup/net_prio`. This group includes all tasks in the system. `/sys/fs/cgroup/net_prio/tasks` lists the tasks in this cgroup.

Each `net_prio` cgroup contains two files that are subsystem specific

**net\_prio.prioidx** This file is read-only, and is simply informative. It contains a unique integer value that the kernel uses as an internal representation of this cgroup.

**net\_prio.ifpriomap** This file contains a map of the priorities assigned to traffic originating from processes in this group and egressing the system on various

interfaces. It contains a list of tuples in the form <ifname priority>. Contents of this file can be modified by echoing a string into the file using the same tuple format. For example:

```
echo "eth0 5" > /sys/fs/cgroups/net_prio/iscsi/net_prio.ifpriomap
```

This command would force any traffic originating from processes belonging to the `iscsi net_prio` cgroup and egressing on interface `eth0` to have the priority of said traffic set to the value 5. The parent accounting group also has a writeable `'net_prio.ifpriomap'` file that can be used to set a system default priority.

Priorities are set immediately prior to queueing a frame to the device queueing discipline (qdisc) so priorities will be assigned prior to the hardware queue selection being made.

One usage for the `net_prio` cgroup is with `mqprio` qdisc allowing application traffic to be steered to hardware/driver based traffic classes. These mappings can then be managed by administrators or other networking protocols such as DCBX.

A new `net_prio` cgroup inherits the parent's configuration.

## 27.12 Process Number Controller

### 27.12.1 Abstract

The process number controller is used to allow a cgroup hierarchy to stop any new tasks from being fork()'d or clone()'d after a certain limit is reached.

Since it is trivial to hit the task limit without hitting any `kmemcg` limits in place, PIDs are a fundamental resource. As such, PID exhaustion must be preventable in the scope of a cgroup hierarchy by allowing resource limiting of the number of tasks in a cgroup.

### 27.12.2 Usage

In order to use the `pids` controller, set the maximum number of tasks in `pids.max` (this is not available in the root cgroup for obvious reasons). The number of processes currently in the cgroup is given by `pids.current`.

Organisational operations are not blocked by cgroup policies, so it is possible to have `pids.current > pids.max`. This can be done by either setting the limit to be smaller than `pids.current`, or attaching enough processes to the cgroup such that `pids.current > pids.max`. However, it is not possible to violate a cgroup policy through `fork()` or `clone()`. `fork()` and `clone()` will return `-EAGAIN` if the creation of a new process would cause a cgroup policy to be violated.

To set a cgroup to have no limit, set `pids.max` to "max". This is the default for all new cgroups (N.B. that PID limits are hierarchical, so the most stringent limit in the hierarchy is followed).

`pids.current` tracks all child cgroup hierarchies, so `parent/pids.current` is a superset of `parent/child/pids.current`.

The pids.events file contains event counters:

- max: Number of times fork failed because limit was hit.

### 27.12.3 Example

First, we mount the pids controller:

```
# mkdir -p /sys/fs/cgroup/pids
# mount -t cgroup -o pids none /sys/fs/cgroup/pids
```

Then we create a hierarchy, set limits and attach processes to it:

```
# mkdir -p /sys/fs/cgroup/pids/parent/child
# echo 2 > /sys/fs/cgroup/pids/parent/pids.max
# echo $$ > /sys/fs/cgroup/pids/parent/cgroup.procs
# cat /sys/fs/cgroup/pids/parent/pids.current
2
#
```

It should be noted that attempts to overcome the set limit (2 in this case) will fail:

```
# cat /sys/fs/cgroup/pids/parent/pids.current
2
# ( /bin/echo "Here's some processes for you." | cat )
sh: fork: Resource temporary unavailable
#
```

Even if we migrate to a child cgroup (which doesn't have a set limit), we will not be able to overcome the most stringent limit in the hierarchy (in this case, parent's):

```
# echo $$ > /sys/fs/cgroup/pids/parent/child/cgroup.procs
# cat /sys/fs/cgroup/pids/parent/pids.current
2
# cat /sys/fs/cgroup/pids/parent/child/pids.current
2
# cat /sys/fs/cgroup/pids/parent/child/pids.max
max
# ( /bin/echo "Here's some processes for you." | cat )
sh: fork: Resource temporary unavailable
#
```

We can set a limit that is smaller than pids.current, which will stop any new processes from being forked at all (note that the shell itself counts towards pids.current):

```
# echo 1 > /sys/fs/cgroup/pids/parent/pids.max
# /bin/echo "We can't even spawn a single process now."
sh: fork: Resource temporary unavailable
# echo 0 > /sys/fs/cgroup/pids/parent/pids.max
# /bin/echo "We can't even spawn a single process now."
sh: fork: Resource temporary unavailable
#
```

## **27.13 RDMA Controller**

### **27.13.1 1. Overview**

#### **1-1. What is RDMA controller?**

RDMA controller allows user to limit RDMA/IB specific resources that a given set of processes can use. These processes are grouped using RDMA controller.

RDMA controller defines two resources which can be limited for processes of a cgroup.

#### **1-2. Why RDMA controller needed?**

Currently user space applications can easily take away all the rdma verb specific resources such as AH, CQ, QP, MR etc. Due to which other applications in other cgroup or kernel space ULPs may not even get chance to allocate any rdma resources. This can lead to service unavailability.

Therefore RDMA controller is needed through which resource consumption of processes can be limited. Through this controller different rdma resources can be accounted.

#### **1-3. How is RDMA controller implemented?**

RDMA cgroup allows limit configuration of resources. Rdma cgroup maintains resource accounting per cgroup, per device using resource pool structure. Each such resource pool is limited up to 64 resources in given resource pool by rdma cgroup, which can be extended later if required.

This resource pool object is linked to the cgroup css. Typically there are 0 to 4 resource pool instances per cgroup, per device in most use cases. But nothing limits to have it more. At present hundreds of RDMA devices per single cgroup may not be handled optimally, however there is no known use case or requirement for such configuration either.

Since RDMA resources can be allocated from any process and can be freed by any of the child processes which shares the address space, rdma resources are always owned by the creator cgroup css. This allows process migration from one to other cgroup without major complexity of transferring resource ownership; because such ownership is not really present due to shared nature of rdma resources. Linking resources around css also ensures that cgroups can be deleted after processes migrated. This allow progress migration as well with active resources, even though that is not a primary use case.

Whenever RDMA resource charging occurs, owner rdma cgroup is returned to the caller. Same rdma cgroup should be passed while uncharging the resource. This also allows process migrated with active RDMA resource to charge to new owner cgroup for new resource. It also allows to uncharge resource of a process from previously charged cgroup which is migrated to new cgroup, even though that is not a primary use case.

Resource pool object is created in following situations. (a) User sets the limit and no previous resource pool exist for the device of interest for the cgroup. (b) No resource limits were configured, but IB/RDMA stack tries to charge the resource. So that it correctly uncharge them when applications are running without limits and later on when limits are enforced during uncharging, otherwise usage count will drop to negative.

Resource pool is destroyed if all the resource limits are set to max and it is the last resource getting deallocated.

User should set all the limit to max value if it intents to remove/unconfigure the resource pool for a particular device.

IB stack honors limits enforced by the rdma controller. When application query about maximum resource limits of IB device, it returns minimum of what is configured by user for a given cgroup and what is supported by IB device.

Following resources can be accounted by rdma controller.

hca_handle	Maximum number of HCA Handles
hca_object	Maximum number of HCA Objects

### 27.13.2 2. Usage Examples

(a) Configure resource limit:

```
echo mlx4_0 hca_handle=2 hca_object=2000 > /sys/fs/cgroup/rdma/1/rdma.
↪max
echo ocrdma1 hca_handle=3 > /sys/fs/cgroup/rdma/2/rdma.max
```

(b) Query resource limit:

```
cat /sys/fs/cgroup/rdma/2/rdma.max
#Output:
mlx4_0 hca_handle=2 hca_object=2000
ocrdma1 hca_handle=3 hca_object=max
```

(c) Query current usage:

```
cat /sys/fs/cgroup/rdma/2/rdma.current
#Output:
mlx4_0 hca_handle=1 hca_object=20
ocrdma1 hca_handle=1 hca_object=23
```

(d) Delete resource limit:

```
echo echo mlx4_0 hca_handle=max hca_object=max > /sys/fs/cgroup/rdma/
↪1/rdma.max
```



## **CONTROL GROUP V2**

**Date** October, 2015

**Author** Tejun Heo <[tj@kernel.org](mailto:tj@kernel.org)>

This is the authoritative documentation on the design, interface and conventions of cgroup v2. It describes all userland-visible aspects of cgroup including core and specific controller behaviors. All future changes must be reflected in this document. Documentation for v1 is available under `Documentation/admin-guide/cgroup-v1/index.rst`.

## **28.1 Introduction**

### **28.1.1 Terminology**

“cgroup” stands for “control group” and is never capitalized. The singular form is used to designate the whole feature and also as a qualifier as in “cgroup controllers”. When explicitly referring to multiple individual control groups, the plural form “cgroups” is used.

### **28.1.2 What is cgroup?**

cgroup is a mechanism to organize processes hierarchically and distribute system resources along the hierarchy in a controlled and configurable manner.

cgroup is largely composed of two parts - the core and controllers. cgroup core is primarily responsible for hierarchically organizing processes. A cgroup controller is usually responsible for distributing a specific type of system resource along the hierarchy although there are utility controllers which serve purposes other than resource distribution.

cgroups form a tree structure and every process in the system belongs to one and only one cgroup. All threads of a process belong to the same cgroup. On creation, all processes are put in the cgroup that the parent process belongs to at the time. A process can be migrated to another cgroup. Migration of a process doesn't affect already existing descendant processes.

Following certain structural constraints, controllers may be enabled or disabled selectively on a cgroup. All controller behaviors are hierarchical - if a controller is enabled on a cgroup, it affects all processes which belong to the cgroups consisting the inclusive sub-hierarchy of the cgroup. When a controller is enabled on a nested

cgroup, it always restricts the resource distribution further. The restrictions set closer to the root in the hierarchy can not be overridden from further away.

## 28.2 Basic Operations

### 28.2.1 Mounting

Unlike v1, cgroup v2 has only single hierarchy. The cgroup v2 hierarchy can be mounted with the following mount command:

```
# mount -t cgroup2 none $MOUNT_POINT
```

cgroup2 filesystem has the magic number 0x63677270 ( “cgrp” ). All controllers which support v2 and are not bound to a v1 hierarchy are automatically bound to the v2 hierarchy and show up at the root. Controllers which are not in active use in the v2 hierarchy can be bound to other hierarchies. This allows mixing v2 hierarchy with the legacy v1 multiple hierarchies in a fully backward compatible way.

A controller can be moved across hierarchies only after the controller is no longer referenced in its current hierarchy. Because per-cgroup controller states are destroyed asynchronously and controllers may have lingering references, a controller may not show up immediately on the v2 hierarchy after the final unmount of the previous hierarchy. Similarly, a controller should be fully disabled to be moved out of the unified hierarchy and it may take some time for the disabled controller to become available for other hierarchies; furthermore, due to inter-controller dependencies, other controllers may need to be disabled too.

While useful for development and manual configurations, moving controllers dynamically between the v2 and other hierarchies is strongly discouraged for production use. It is recommended to decide the hierarchies and controller associations before starting using the controllers after system boot.

During transition to v2, system management software might still automount the v1 cgroup filesystem and so hijack all controllers during boot, before manual intervention is possible. To make testing and experimenting easier, the kernel parameter `cgroup_no_v1=` allows disabling controllers in v1 and make them always available in v2.

cgroup v2 currently supports the following mount options.

#### nsdelegate

Consider cgroup namespaces as delegation boundaries. This option is system wide and can only be set on mount or modified through remount from the init namespace. The mount option is ignored on non-init namespace mounts. Please refer to the Delegation section for details.

#### memory\_localevents

Only populate memory.events with data for the current cgroup, and not any subtrees. This is legacy behaviour, the default behaviour without this option is to include subtree counts. This

option is system wide and can only be set on mount or modified through remount from the init namespace. The mount option is ignored on non-init namespace mounts.

### memory\_recursiveprot

Recursively apply memory.min and memory.low protection to entire subtrees, without requiring explicit downward propagation into leaf cgroups. This allows protecting entire subtrees from one another, while retaining free competition within those subtrees. This should have been the default behavior but is a mount-option to avoid regressing setups relying on the original semantics (e.g. specifying bogusly high ‘bypass’ protection values at higher tree levels).

## 28.2.2 Organizing Processes and Threads

### Processes

Initially, only the root cgroup exists to which all processes belong. A child cgroup can be created by creating a sub-directory:

```
# mkdir $CGROUP_NAME
```

A given cgroup may have multiple child cgroups forming a tree structure. Each cgroup has a read-writable interface file “cgroup.procs”. When read, it lists the PIDs of all processes which belong to the cgroup one-per-line. The PIDs are not ordered and the same PID may show up more than once if the process got moved to another cgroup and then back or the PID got recycled while reading.

A process can be migrated into a cgroup by writing its PID to the target cgroup’s “cgroup.procs” file. Only one process can be migrated on a single write(2) call. If a process is composed of multiple threads, writing the PID of any thread migrates all threads of the process.

When a process forks a child process, the new process is born into the cgroup that the forking process belongs to at the time of the operation. After exit, a process stays associated with the cgroup that it belonged to at the time of exit until it’s reaped; however, a zombie process does not appear in “cgroup.procs” and thus can’t be moved to another cgroup.

A cgroup which doesn’t have any children or live processes can be destroyed by removing the directory. Note that a cgroup which doesn’t have any children and is associated only with zombie processes is considered empty and can be removed:

```
# rmdir $CGROUP_NAME
```

“/proc/\$PID/cgroup” lists a process’s cgroup membership. If legacy cgroup is in use in the system, this file may contain multiple lines, one for each hierarchy. The entry for cgroup v2 is always in the format “0::\$PATH” :

```
# cat /proc/842/cgroup
...
0::/test-cgroup/test-cgroup-nested
```

If the process becomes a zombie and the cgroup it was associated with is removed subsequently, ” (deleted)” is appended to the path:

```
# cat /proc/842/cgroup
...
0::/test-cgroup/test-cgroup-nested (deleted)
```

### Threads

cgroup v2 supports thread granularity for a subset of controllers to support use cases requiring hierarchical resource distribution across the threads of a group of processes. By default, all threads of a process belong to the same cgroup, which also serves as the resource domain to host resource consumptions which are not specific to a process or thread. The thread mode allows threads to be spread across a subtree while still maintaining the common resource domain for them.

Controllers which support thread mode are called threaded controllers. The ones which don't are called domain controllers.

Marking a cgroup threaded makes it join the resource domain of its parent as a threaded cgroup. The parent may be another threaded cgroup whose resource domain is further up in the hierarchy. The root of a threaded subtree, that is, the nearest ancestor which is not threaded, is called threaded domain or thread root interchangeably and serves as the resource domain for the entire subtree.

Inside a threaded subtree, threads of a process can be put in different cgroups and are not subject to the no internal process constraint - threaded controllers can be enabled on non-leaf cgroups whether they have threads in them or not.

As the threaded domain cgroup hosts all the domain resource consumptions of the subtree, it is considered to have internal resource consumptions whether there are processes in it or not and can't have populated child cgroups which aren't threaded. Because the root cgroup is not subject to no internal process constraint, it can serve both as a threaded domain and a parent to domain cgroups.

The current operation mode or type of the cgroup is shown in the “cgroup.type” file which indicates whether the cgroup is a normal domain, a domain which is serving as the domain of a threaded subtree, or a threaded cgroup.

On creation, a cgroup is always a domain cgroup and can be made threaded by writing “threaded” to the “cgroup.type” file. The operation is single direction:

```
# echo threaded > cgroup.type
```

Once threaded, the cgroup can't be made a domain again. To enable the thread mode, the following conditions must be met.

- As the cgroup will join the parent's resource domain. The parent must either be a valid (threaded) domain or a threaded cgroup.
- When the parent is an unthreaded domain, it must not have any domain controllers enabled or populated domain children. The root is exempt from this requirement.

Topology-wise, a cgroup can be in an invalid state. Please consider the following topology:

```
A (threaded domain) - B (threaded) - C (domain, just created)
```

C is created as a domain but isn't connected to a parent which can host child domains. C can't be used until it is turned into a threaded cgroup. "cgroup.type" file will report "domain (invalid)" in these cases. Operations which fail due to invalid topology use EOPNOTSUPP as the errno.

A domain cgroup is turned into a threaded domain when one of its child cgroup becomes threaded or threaded controllers are enabled in the "cgroup.subtree\_control" file while there are processes in the cgroup. A threaded domain reverts to a normal domain when the conditions clear.

When read, "cgroup.threads" contains the list of the thread IDs of all threads in the cgroup. Except that the operations are per-thread instead of per-process, "cgroup.threads" has the same format and behaves the same way as "cgroup.procs". While "cgroup.threads" can be written to in any cgroup, as it can only move threads inside the same threaded domain, its operations are confined inside each threaded subtree.

The threaded domain cgroup serves as the resource domain for the whole subtree, and, while the threads can be scattered across the subtree, all the processes are considered to be in the threaded domain cgroup. "cgroup.procs" in a threaded domain cgroup contains the PIDs of all processes in the subtree and is not readable in the subtree proper. However, "cgroup.procs" can be written to from anywhere in the subtree to migrate all threads of the matching process to the cgroup.

Only threaded controllers can be enabled in a threaded subtree. When a threaded controller is enabled inside a threaded subtree, it only accounts for and controls resource consumptions associated with the threads in the cgroup and its descendants. All consumptions which aren't tied to a specific thread belong to the threaded domain cgroup.

Because a threaded subtree is exempt from no internal process constraint, a threaded controller must be able to handle competition between threads in a non-leaf cgroup and its child cgroups. Each threaded controller defines how such competitions are handled.

### 28.2.3 [Un]populated Notification

Each non-root cgroup has a "cgroup.events" file which contains "populated" field indicating whether the cgroup's sub-hierarchy has live processes in it. Its value is 0 if there is no live process in the cgroup and its descendants; otherwise, 1. poll and [id]notify events are triggered when the value changes. This can be used, for example, to start a clean-up operation after all processes of a given sub-hierarchy have exited. The populated state updates and notifications are recursive. Consider the following sub-hierarchy where the numbers in the parentheses represent the numbers of processes in each cgroup:

```
A(4) - B(0) - C(1)
      \ D(0)
```

A, B and C's "populated" fields would be 1 while D's 0. After the one process in C exits, B and C's "populated" fields would flip to "0" and file modified events

will be generated on the “cgroup.events” files of both cgroups.

### 28.2.4 Controlling Controllers

#### Enabling and Disabling

Each cgroup has a “cgroup.controllers” file which lists all controllers available for the cgroup to enable:

```
# cat cgroup.controllers
cpu io memory
```

No controller is enabled by default. Controllers can be enabled and disabled by writing to the “cgroup.subtree\_control” file:

```
# echo "+cpu +memory -io" > cgroup.subtree_control
```

Only controllers which are listed in “cgroup.controllers” can be enabled. When multiple operations are specified as above, either they all succeed or fail. If multiple operations on the same controller are specified, the last one is effective.

Enabling a controller in a cgroup indicates that the distribution of the target resource across its immediate children will be controlled. Consider the following sub-hierarchy. The enabled controllers are listed in parentheses:

```
A(cpu,memory) - B(memory) - C()
                        \ D()
```

As A has “cpu” and “memory” enabled, A will control the distribution of CPU cycles and memory to its children, in this case, B. As B has “memory” enabled but not “CPU”, C and D will compete freely on CPU cycles but their division of memory available to B will be controlled.

As a controller regulates the distribution of the target resource to the cgroup’s children, enabling it creates the controller’s interface files in the child cgroups. In the above example, enabling “cpu” on B would create the “cpu.” prefixed controller interface files in C and D. Likewise, disabling “memory” from B would remove the “memory.” prefixed controller interface files from C and D. This means that the controller interface files - anything which doesn’t start with “cgroup.” are owned by the parent rather than the cgroup itself.

#### Top-down Constraint

Resources are distributed top-down and a cgroup can further distribute a resource only if the resource has been distributed to it from the parent. This means that all non-root “cgroup.subtree\_control” files can only contain controllers which are enabled in the parent’s “cgroup.subtree\_control” file. A controller can be enabled only if the parent has the controller enabled and a controller can’t be disabled if one or more children have it enabled.

## **No Internal Process Constraint**

Non-root cgroups can distribute domain resources to their children only when they don't have any processes of their own. In other words, only domain cgroups which don't contain any processes can have domain controllers enabled in their "cgroup.subtree\_control" files.

This guarantees that, when a domain controller is looking at the part of the hierarchy which has it enabled, processes are always only on the leaves. This rules out situations where child cgroups compete against internal processes of the parent.

The root cgroup is exempt from this restriction. Root contains processes and anonymous resource consumption which can't be associated with any other cgroups and requires special treatment from most controllers. How resource consumption in the root cgroup is governed is up to each controller (for more information on this topic please refer to the Non-normative information section in the Controllers chapter).

Note that the restriction doesn't get in the way if there is no enabled controller in the cgroup's "cgroup.subtree\_control". This is important as otherwise it wouldn't be possible to create children of a populated cgroup. To control resource distribution of a cgroup, the cgroup must create children and transfer all its processes to the children before enabling controllers in its "cgroup.subtree\_control" file.

## **28.2.5 Delegation**

### **Model of Delegation**

A cgroup can be delegated in two ways. First, to a less privileged user by granting write access of the directory and its "cgroup.procs", "cgroup.threads" and "cgroup.subtree\_control" files to the user. Second, if the "nsdelegate" mount option is set, automatically to a cgroup namespace on namespace creation.

Because the resource control interface files in a given directory control the distribution of the parent's resources, the delegatee shouldn't be allowed to write to them. For the first method, this is achieved by not granting access to these files. For the second, the kernel rejects writes to all files other than "cgroup.procs" and "cgroup.subtree\_control" on a namespace root from inside the namespace.

The end results are equivalent for both delegation types. Once delegated, the user can build sub-hierarchy under the directory, organize processes inside it as it sees fit and further distribute the resources it received from the parent. The limits and other settings of all resource controllers are hierarchical and regardless of what happens in the delegated sub-hierarchy, nothing can escape the resource restrictions imposed by the parent.

Currently, cgroup doesn't impose any restrictions on the number of cgroups in or nesting depth of a delegated sub-hierarchy; however, this may be limited explicitly in the future.

### Delegation Containment

A delegated sub-hierarchy is contained in the sense that processes can't be moved into or out of the sub-hierarchy by the delegatee.

For delegations to a less privileged user, this is achieved by requiring the following conditions for a process with a non-root euid to migrate a target process into a cgroup by writing its PID to the "cgroup.procs" file.

- The writer must have write access to the "cgroup.procs" file.
- The writer must have write access to the "cgroup.procs" file of the common ancestor of the source and destination cgroups.

The above two constraints ensure that while a delegatee may migrate processes around freely in the delegated sub-hierarchy it can't pull in from or push out to outside the sub-hierarchy.

For an example, let's assume cgroups C0 and C1 have been delegated to user U0 who created C00, C01 under C0 and C10 under C1 as follows and all processes under C0 and C1 belong to U0:

```
~~~~~ - C0 - C00
~ cgroup ~ \ C01
~ hierarchy ~
~~~~~ - C1 - C10
```

Let's also say U0 wants to write the PID of a process which is currently in C10 into "C00/cgroup.procs". U0 has write access to the file; however, the common ancestor of the source cgroup C10 and the destination cgroup C00 is above the points of delegation and U0 would not have write access to its "cgroup.procs" files and thus the write will be denied with -EACCES.

For delegations to namespaces, containment is achieved by requiring that both the source and destination cgroups are reachable from the namespace of the process which is attempting the migration. If either is not reachable, the migration is rejected with -ENOENT.

### 28.2.6 Guidelines

#### Organize Once and Control

Migrating a process across cgroups is a relatively expensive operation and stateful resources such as memory are not moved together with the process. This is an explicit design decision as there often exist inherent trade-offs between migration and various hot paths in terms of synchronization cost.

As such, migrating processes across cgroups frequently as a means to apply different resource restrictions is discouraged. A workload should be assigned to a cgroup according to the system's logical and resource structure once on start-up. Dynamic adjustments to resource distribution can be made by changing controller configuration through the interface files.

## Avoid Name Collisions

Interface files for a cgroup and its children cgroups occupy the same directory and it is possible to create children cgroups which collide with interface files.

All cgroup core interface files are prefixed with “cgroup.” and each controller’s interface files are prefixed with the controller name and a dot. A controller’s name is composed of lower case alphabets and ‘\_’ s but never begins with an ‘\_’ so it can be used as the prefix character for collision avoidance. Also, interface file names won’t start or end with terms which are often used in categorizing workloads such as job, service, slice, unit or workload.

cgroup doesn’t do anything to prevent name collisions and it’s the user’s responsibility to avoid them.

## 28.3 Resource Distribution Models

cgroup controllers implement several resource distribution schemes depending on the resource type and expected use cases. This section describes major schemes in use along with their expected behaviors.

### 28.3.1 Weights

A parent’s resource is distributed by adding up the weights of all active children and giving each the fraction matching the ratio of its weight against the sum. As only children which can make use of the resource at the moment participate in the distribution, this is work-conserving. Due to the dynamic nature, this model is usually used for stateless resources.

All weights are in the range [1, 10000] with the default at 100. This allows symmetric multiplicative biases in both directions at fine enough granularity while staying in the intuitive range.

As long as the weight is in range, all configuration combinations are valid and there is no reason to reject configuration changes or process migrations.

“cpu.weight” proportionally distributes CPU cycles to active children and is an example of this type.

### 28.3.2 Limits

A child can only consume upto the configured amount of the resource. Limits can be over-committed - the sum of the limits of children can exceed the amount of resource available to the parent.

Limits are in the range [0, max] and defaults to “max” , which is noop.

As limits can be over-committed, all configuration combinations are valid and there is no reason to reject configuration changes or process migrations.

“io.max” limits the maximum BPS and/or IOPS that a cgroup can consume on an IO device and is an example of this type.

### 28.3.3 Protections

A cgroup is protected upto the configured amount of the resource as long as the usages of all its ancestors are under their protected levels. Protections can be hard guarantees or best effort soft boundaries. Protections can also be over-committed in which case only upto the amount available to the parent is protected among children.

Protections are in the range [0, max] and defaults to 0, which is noop.

As protections can be over-committed, all configuration combinations are valid and there is no reason to reject configuration changes or process migrations.

“memory.low” implements best-effort memory protection and is an example of this type.

### 28.3.4 Allocations

A cgroup is exclusively allocated a certain amount of a finite resource. Allocations can’ t be over-committed - the sum of the allocations of children can not exceed the amount of resource available to the parent.

Allocations are in the range [0, max] and defaults to 0, which is no resource.

As allocations can’ t be over-committed, some configuration combinations are invalid and should be rejected. Also, if the resource is mandatory for execution of processes, process migrations may be rejected.

“cpu.rt.max” hard-allocates realtime slices and is an example of this type.

## 28.4 Interface Files

### 28.4.1 Format

All interface files should be in one of the following formats whenever possible:

```
New-line separated values
(when only one value can be written at once)

    VAL0\n
    VAL1\n
    ...

Space separated values
(when read-only or multiple values can be written at once)

    VAL0 VAL1 ...\n

Flat keyed

    KEY0 VAL0\n
    KEY1 VAL1\n
    ...
```

(continues on next page)

(continued from previous page)

Nested keyed

```
KEY0 SUB_KEY0=VAL00 SUB_KEY1=VAL01...
KEY1 SUB_KEY0=VAL10 SUB_KEY1=VAL11...
...
```

For a writable file, the format for writing should generally match reading; however, controllers may allow omitting later fields or implement restricted shortcuts for most common use cases.

For both flat and nested keyed files, only the values for a single key can be written at a time. For nested keyed files, the sub key pairs may be specified in any order and not all pairs have to be specified.

### 28.4.2 Conventions

- Settings for a single feature should be contained in a single file.
- The root cgroup should be exempt from resource control and thus shouldn't have resource control interface files.
- The default time unit is microseconds. If a different unit is ever used, an explicit unit suffix must be present.
- A parts-per quantity should use a percentage decimal with at least two digit fractional part - e.g. 13.40.
- If a controller implements weight based resource distribution, its interface file should be named "weight" and have the range [1, 10000] with 100 as the default. The values are chosen to allow enough and symmetric bias in both directions while keeping it intuitive (the default is 100%).
- If a controller implements an absolute resource guarantee and/or limit, the interface files should be named "min" and "max" respectively. If a controller implements best effort resource guarantee and/or limit, the interface files should be named "low" and "high" respectively.

In the above four control files, the special token "max" should be used to represent upward infinity for both reading and writing.

- If a setting has a configurable default value and keyed specific overrides, the default entry should be keyed with "default" and appear as the first entry in the file.

The default value can be updated by writing either "default \$VAL" or "\$VAL".

When writing to update a specific override, "default" can be used as the value to indicate removal of the override. Override entries with "default" as the value must not appear when read.

For example, a setting which is keyed by major:minor device numbers with integer values may look like the following:

```
# cat cgroup-example-interface-file
default 150
8:0 300
```

The default value can be updated by:

```
# echo 125 > cgroup-example-interface-file
```

or:

```
# echo "default 125" > cgroup-example-interface-file
```

An override can be set by:

```
# echo "8:16 170" > cgroup-example-interface-file
```

and cleared by:

```
# echo "8:0 default" > cgroup-example-interface-file
# cat cgroup-example-interface-file
default 125
8:16 170
```

- For events which are not very high frequency, an interface file “events” should be created which lists event key value pairs. Whenever a notifiable event happens, file modified event should be generated on the file.

### 28.4.3 Core Interface Files

All cgroup core files are prefixed with “cgroup.”

**cgroup.type**

A read-write single value file which exists on non-root cgroups.

When read, it indicates the current type of the cgroup, which can be one of the following values.

- “domain” : A normal valid domain cgroup.
- “domain threaded” : A threaded domain cgroup which is serving as the root of a threaded subtree.
- “domain invalid” : A cgroup which is in an invalid state. It can’t be populated or have controllers enabled. It may be allowed to become a threaded cgroup.
- “threaded” : A threaded cgroup which is a member of a threaded subtree.

A cgroup can be turned into a threaded cgroup by writing “threaded” to this file.

**cgroup.procs** A read-write new-line separated values file which exists on all cgroups.

When read, it lists the PIDs of all processes which belong to the cgroup one-per-line. The PIDs are not ordered and the same PID may show up more than once if the process got moved to another cgroup and then back or the PID got recycled while reading.

A PID can be written to migrate the process associated with the PID to the cgroup. The writer should match all of the following conditions.

- It must have write access to the “cgroup.procs” file.
- It must have write access to the “cgroup.procs” file of the common ancestor of the source and destination cgroups.

When delegating a sub-hierarchy, write access to this file should be granted along with the containing directory.

In a threaded cgroup, reading this file fails with EOPNOTSUPP as all the processes belong to the thread root. Writing is supported and moves every thread of the process to the cgroup.

**cgroup.threads** A read-write new-line separated values file which exists on all cgroups.

When read, it lists the TIDs of all threads which belong to the cgroup one-per-line. The TIDs are not ordered and the same TID may show up more than once if the thread got moved to another cgroup and then back or the TID got recycled while reading.

A TID can be written to migrate the thread associated with the TID to the cgroup. The writer should match all of the following conditions.

- It must have write access to the “cgroup.threads” file.
- The cgroup that the thread is currently in must be in the same resource domain as the destination cgroup.
- It must have write access to the “cgroup.procs” file of the common ancestor of the source and destination cgroups.

When delegating a sub-hierarchy, write access to this file should be granted along with the containing directory.

**cgroup.controllers** A read-only space separated values file which exists on all cgroups.

It shows space separated list of all controllers available to the cgroup. The controllers are not ordered.

**cgroup.subtree\_control** A read-write space separated values file which exists on all cgroups. Starts out empty.

When read, it shows space separated list of the controllers which are enabled to control resource distribution from the cgroup to its children.

Space separated list of controllers prefixed with ‘+’ or ‘-’ can be written to enable or disable controllers. A controller name prefixed with ‘+’ enables the controller and ‘-’ disables. If a controller appears more than once on the list, the last one is effective. When multiple

enable and disable operations are specified, either all succeed or all fail.

**cgroup.events** A read-only flat-keyed file which exists on non-root cgroups. The following entries are defined. Unless specified otherwise, a value change in this file generates a file modified event.

**populated** 1 if the cgroup or its descendants contains any live processes; otherwise, 0.

**frozen** 1 if the cgroup is frozen; otherwise, 0.

**cgroup.max.descendants** A read-write single value files. The default is “max” .

Maximum allowed number of descent cgroups. If the actual number of descendants is equal or larger, an attempt to create a new cgroup in the hierarchy will fail.

**cgroup.max.depth** A read-write single value files. The default is “max” .

Maximum allowed descent depth below the current cgroup. If the actual descent depth is equal or larger, an attempt to create a new child cgroup will fail.

**cgroup.stat** A read-only flat-keyed file with the following entries:

**nr\_descendants** Total number of visible descendant cgroups.

**nr\_dying\_descendants** Total number of dying descendant cgroups. A cgroup becomes dying after being deleted by a user. The cgroup will remain in dying state for some time undefined time (which can depend on system load) before being completely destroyed.

A process can't enter a dying cgroup under any circumstances, a dying cgroup can't revive.

A dying cgroup can consume system resources not exceeding limits, which were active at the moment of cgroup deletion.

**cgroup.freeze** A read-write single value file which exists on non-root cgroups. Allowed values are “0” and “1” . The default is “0” .

Writing “1” to the file causes freezing of the cgroup and all descendant cgroups. This means that all belonging processes will be stopped and will not run until the cgroup will be explicitly unfrozen. Freezing of the cgroup may take some time; when this action is completed, the “frozen” value in the cgroup.events control file will be updated to “1” and the corresponding notification will be issued.

A cgroup can be frozen either by its own settings, or by settings of any ancestor cgroups. If any of ancestor cgroups is frozen, the cgroup will remain frozen.

Processes in the frozen cgroup can be killed by a fatal signal. They also can enter and leave a frozen cgroup: either by an explicit move by a user, or if freezing of the cgroup races with `fork()`. If a process is moved to a frozen cgroup, it stops. If a process is moved out of a frozen cgroup, it becomes running.

Frozen status of a cgroup doesn't affect any cgroup tree operations: it's possible to delete a frozen (and empty) cgroup, as well as create new sub-cgroups.

## 28.5 Controllers

### 28.5.1 CPU

The “cpu” controllers regulates distribution of CPU cycles. This controller implements weight and absolute bandwidth limit models for normal scheduling policy and absolute bandwidth allocation model for realtime scheduling policy.

In all the above models, cycles distribution is defined only on a temporal base and it does not account for the frequency at which tasks are executed. The (optional) utilization clamping support allows to hint the `schedutil cpufreq` governor about the minimum desired frequency which should always be provided by a CPU, as well as the maximum desired frequency, which should not be exceeded by a CPU.

**WARNING:** cgroup2 doesn't yet support control of realtime processes and the cpu controller can only be enabled when all RT processes are in the root cgroup. Be aware that system management software may already have placed RT processes into nonroot cgroups during the system boot process, and these processes may need to be moved to the root cgroup before the cpu controller can be enabled.

### CPU Interface Files

All time durations are in microseconds.

**cpu.stat** A read-only flat-keyed file. This file exists whether the controller is enabled or not.

It always reports the following three stats:

- `usage_usec`
- `user_usec`
- `system_usec`

and the following three when the controller is enabled:

- `nr_periods`
- `nr_throttled`
- `throttled_usec`

**cpu.weight** A read-write single value file which exists on non-root cgroups. The default is “100” .

The weight in the range [1, 10000].

**cpu.weight.nice** A read-write single value file which exists on non-root cgroups. The default is “0” .

The nice value is in the range [-20, 19].

This interface file is an alternative interface for “cpu.weight” and allows reading and setting weight using the same values used by nice(2). Because the range is smaller and granularity is coarser for the nice values, the read value is the closest approximation of the current weight.

**cpu.max** A read-write two value file which exists on non-root cgroups. The default is “max 100000” .

The maximum bandwidth limit. It’ s in the following format:

\$MAX \$PERIOD
----------------

which indicates that the group may consume upto \$MAX in each \$PERIOD duration. “max” for \$MAX indicates no limit. If only one number is written, \$MAX is updated.

**cpu.pressure** A read-only nested-key file which exists on non-root cgroups.

Shows pressure stall information for CPU. See Documentation/accounting/psi.rst for details.

**cpu.uclamp.min** A read-write single value file which exists on non-root cgroups. The default is “0” , i.e. no utilization boosting.

The requested minimum utilization (protection) as a percentage rational number, e.g. 12.34 for 12.34%.

This interface allows reading and setting minimum utilization clamp values similar to the sched\_setattr(2). This minimum utilization value is used to clamp the task specific minimum utilization clamp.

The requested minimum utilization (protection) is always capped by the current value for the maximum utilization (limit), i.e. cpu.uclamp.max.

**cpu.uclamp.max** A read-write single value file which exists on non-root cgroups. The default is “max” . i.e. no utilization capping

The requested maximum utilization (limit) as a percentage rational number, e.g. 98.76 for 98.76%.

This interface allows reading and setting maximum utilization clamp values similar to the sched\_setattr(2). This maximum utilization value is used to clamp the task specific maximum utilization clamp.

## 28.5.2 Memory

The “memory” controller regulates distribution of memory. Memory is stateful and implements both limit and protection models. Due to the intertwining between memory usage and reclaim pressure and the stateful nature of memory, the distribution model is relatively complex.

While not completely water-tight, all major memory usages by a given cgroup are tracked so that the total memory consumption can be accounted and controlled to a reasonable extent. Currently, the following types of memory usages are tracked.

- Userland memory - page cache and anonymous memory.
- Kernel data structures such as dentries and inodes.
- TCP socket buffers.

The above list may expand in the future for better coverage.

### Memory Interface Files

All memory amounts are in bytes. If a value which is not aligned to `PAGE_SIZE` is written, the value may be rounded up to the closest `PAGE_SIZE` multiple when read back.

**memory.current** A read-only single value file which exists on non-root cgroups.

The total amount of memory currently being used by the cgroup and its descendants.

**memory.min** A read-write single value file which exists on non-root cgroups. The default is “0” .

Hard memory protection. If the memory usage of a cgroup is within its effective min boundary, the cgroup’s memory won’t be reclaimed under any conditions. If there is no unprotected reclaimable memory available, OOM killer is invoked. Above the effective min boundary (or effective low boundary if it is higher), pages are reclaimed proportionally to the overage, reducing reclaim pressure for smaller overages.

Effective min boundary is limited by `memory.min` values of all ancestor cgroups. If there is `memory.min` overcommitment (child cgroup or cgroups are requiring more protected memory than parent will allow), then each child cgroup will get the part of parent’s protection proportional to its actual memory usage below `memory.min`.

Putting more memory than generally available under this protection is discouraged and may lead to constant OOMs.

If a memory cgroup is not populated with processes, its `memory.min` is ignored.

**memory.low** A read-write single value file which exists on non-root cgroups. The default is “0” .

Best-effort memory protection. If the memory usage of a cgroup is within its effective low boundary, the cgroup's memory won't be reclaimed unless there is no reclaimable memory available in unprotected cgroups. Above the effective low boundary (or effective min boundary if it is higher), pages are reclaimed proportionally to the overage, reducing reclaim pressure for smaller overages.

Effective low boundary is limited by `memory.low` values of all ancestor cgroups. If there is `memory.low` overcommitment (child cgroup or cgroups are requiring more protected memory than parent will allow), then each child cgroup will get the part of parent's protection proportional to its actual memory usage below `memory.low`.

Putting more memory than generally available under this protection is discouraged.

**memory.high** A read-write single value file which exists on non-root cgroups. The default is "max".

Memory usage throttle limit. This is the main mechanism to control memory usage of a cgroup. If a cgroup's usage goes over the high boundary, the processes of the cgroup are throttled and put under heavy reclaim pressure.

Going over the high limit never invokes the OOM killer and under extreme conditions the limit may be breached.

**memory.max** A read-write single value file which exists on non-root cgroups. The default is "max".

Memory usage hard limit. This is the final protection mechanism. If a cgroup's memory usage reaches this limit and can't be reduced, the OOM killer is invoked in the cgroup. Under certain circumstances, the usage may go over the limit temporarily.

In default configuration regular 0-order allocations always succeed unless OOM killer chooses current task as a victim.

Some kinds of allocations don't invoke the OOM killer. Caller could retry them differently, return into userspace as `-ENOMEM` or silently ignore in cases like disk readahead.

This is the ultimate protection mechanism. As long as the high limit is used and monitored properly, this limit's utility is limited to providing the final safety net.

**memory.oom.group** A read-write single value file which exists on non-root cgroups. The default value is "0".

Determines whether the cgroup should be treated as an indivisible workload by the OOM killer. If set, all tasks belonging to the cgroup or to its descendants (if the memory cgroup is not a leaf cgroup) are killed together or not at all. This can be used to avoid partial kills to guarantee workload integrity.

Tasks with the OOM protection (`oom_score_adj` set to `-1000`) are treated as an exception and are never killed.

If the OOM killer is invoked in a cgroup, it's not going to kill any tasks outside of this cgroup, regardless `memory.oom.group` values of ancestor cgroups.

**memory.events** A read-only flat-keyed file which exists on non-root cgroups. The following entries are defined. Unless specified otherwise, a value change in this file generates a file modified event.

Note that all fields in this file are hierarchical and the file modified event can be generated due to an event down the hierarchy. For the local events at the cgroup level see `memory.events.local`.

**low** The number of times the cgroup is reclaimed due to high memory pressure even though its usage is under the low boundary. This usually indicates that the low boundary is over-committed.

**high** The number of times processes of the cgroup are throttled and routed to perform direct memory reclaim because the high memory boundary was exceeded. For a cgroup whose memory usage is capped by the high limit rather than global memory pressure, this event's occurrences are expected.

**max** The number of times the cgroup's memory usage was about to go over the max boundary. If direct reclaim fails to bring it down, the cgroup goes to OOM state.

**oom** The number of time the cgroup's memory usage was reached the limit and allocation was about to fail.

This event is not raised if the OOM killer is not considered as an option, e.g. for failed high-order allocations or if caller asked to not retry attempts.

**oom\_kill** The number of processes belonging to this cgroup killed by any kind of OOM killer.

**memory.events.local** Similar to `memory.events` but the fields in the file are local to the cgroup i.e. not hierarchical. The file modified event generated on this file reflects only the local events.

**memory.stat** A read-only flat-keyed file which exists on non-root cgroups.

This breaks down the cgroup's memory footprint into different types of memory, type-specific details, and other information on the state and past events of the memory management system.

All memory amounts are in bytes.

The entries are ordered to be human readable, and new entries can show up in the middle. Don't rely on items remaining in a fixed position; use the keys to look up specific values!

**anon** Amount of memory used in anonymous mappings such as `brk()`, `sbrk()`, and `mmap(MAP_ANONYMOUS)`

**file** Amount of memory used to cache filesystem data, including tmpfs and shared memory.

**kernel\_stack** Amount of memory allocated to kernel stacks.

**slab** Amount of memory used for storing in-kernel data structures.

**sock** Amount of memory used in network transmission buffers

**shmem** Amount of cached filesystem data that is swap-backed, such as tmpfs, shm segments, shared anonymous mmap(s)

**file\_mapped** Amount of cached filesystem data mapped with mmap()

**file\_dirty** Amount of cached filesystem data that was modified but not yet written back to disk

**file\_writeback** Amount of cached filesystem data that was modified and is currently being written back to disk

**anon\_thp** Amount of memory used in anonymous mappings backed by transparent hugepages

**inactive\_anon, active\_anon, inactive\_file, active\_file, unevictable** Amount of memory, swap-backed and filesystem-backed, on the internal memory management lists used by the page reclaim algorithm.

As these represent internal list state (eg. shmem pages are on anon memory management lists), inactive\_foo + active\_foo may not be equal to the value for the foo counter, since the foo counter is type-based, not list-based.

**slab\_reclaimable** Part of “slab” that might be reclaimed, such as dentries and inodes.

**slab\_unreclaimable** Part of “slab” that cannot be reclaimed on memory pressure.

**pgfault** Total number of page faults incurred

**pgmajfault** Number of major page faults incurred

**workingset\_refault** Number of refaults of previously evicted pages

**workingset\_activate** Number of refaulted pages that were immediately activated

**workingset\_restore** Number of restored pages which have been detected as an active workingset before they got reclaimed.

**workingset\_nodereclaim** Number of times a shadow node has been reclaimed

**pgrefill** Amount of scanned pages (in an active LRU list)

**pgscan** Amount of scanned pages (in an inactive LRU list)

**pgsteal** Amount of reclaimed pages

**pgactivate** Amount of pages moved to the active LRU list

**pgdeactivate** Amount of pages moved to the inactive LRU list

**pglazyfree** Amount of pages postponed to be freed under memory pressure

**pglazyfreed** Amount of reclaimed lazyfree pages

**thp\_fault\_alloc** Number of transparent hugepages which were allocated to satisfy a page fault. This counter is not present when CONFIG\_TRANSPARENT\_HUGEPAGE is not set.

**thp\_collapse\_alloc** Number of transparent hugepages which were allocated to allow collapsing an existing range of pages. This counter is not present when CONFIG\_TRANSPARENT\_HUGEPAGE is not set.

**memory.swap.current** A read-only single value file which exists on non-root cgroups.

The total amount of swap currently being used by the cgroup and its descendants.

**memory.swap.high** A read-write single value file which exists on non-root cgroups. The default is “max” .

Swap usage throttle limit. If a cgroup’ s swap usage exceeds this limit, all its further allocations will be throttled to allow userspace to implement custom out-of-memory procedures.

This limit marks a point of no return for the cgroup. It is NOT designed to manage the amount of swapping a workload does during regular operation. Compare to `memory.swap.max`, which prohibits swapping past a set amount, but lets the cgroup continue unimpeded as long as other memory can be reclaimed.

Healthy workloads are not expected to reach this limit.

**memory.swap.max** A read-write single value file which exists on non-root cgroups. The default is “max” .

Swap usage hard limit. If a cgroup’ s swap usage reaches this limit, anonymous memory of the cgroup will not be swapped out.

**memory.swap.events** A read-only flat-keyed file which exists on non-root cgroups. The following entries are defined. Unless specified otherwise, a value change in this file generates a file modified event.

**high** The number of times the cgroup’ s swap usage was over the high threshold.

**max** The number of times the cgroup's swap usage was about to go over the max boundary and swap allocation failed.

**fail** The number of times swap allocation failed either because of running out of swap system-wide or max limit.

When reduced under the current usage, the existing swap entries are reclaimed gradually and the swap usage may stay higher than the limit for an extended period of time. This reduces the impact on the workload and memory management.

**memory.pressure** A read-only nested-key file which exists on non-root cgroups.

Shows pressure stall information for memory. See Documentation/accounting/psi.rst for details.

### Usage Guidelines

“memory.high” is the main mechanism to control memory usage. Over-committing on high limit (sum of high limits > available memory) and letting global memory pressure to distribute memory according to usage is a viable strategy.

Because breach of the high limit doesn't trigger the OOM killer but throttles the offending cgroup, a management agent has ample opportunities to monitor and take appropriate actions such as granting more memory or terminating the workload.

Determining whether a cgroup has enough memory is not trivial as memory usage doesn't indicate whether the workload can benefit from more memory. For example, a workload which writes data received from network to a file can use all available memory but can also operate as performant with a small amount of memory. A measure of memory pressure - how much the workload is being impacted due to lack of memory - is necessary to determine whether a workload needs more memory; unfortunately, memory pressure monitoring mechanism isn't implemented yet.

### Memory Ownership

A memory area is charged to the cgroup which instantiated it and stays charged to the cgroup until the area is released. Migrating a process to a different cgroup doesn't move the memory usages that it instantiated while in the previous cgroup to the new cgroup.

A memory area may be used by processes belonging to different cgroups. To which cgroup the area will be charged is in-deterministic; however, over time, the memory area is likely to end up in a cgroup which has enough memory allowance to avoid high reclaim pressure.

If a cgroup sweeps a considerable amount of memory which is expected to be accessed repeatedly by other cgroups, it may make sense to use `POSIX_FADV_DONTNEED` to relinquish the ownership of memory areas belonging to the affected files to ensure correct memory ownership.

### 28.5.3 IO

The “io” controller regulates the distribution of IO resources. This controller implements both weight based and absolute bandwidth or IOPS limit distribution; however, weight based distribution is available only if cfq-iosched is in use and neither scheme is available for blk-mq devices.

#### IO Interface Files

**io.stat** A read-only nested-keyed file which exists on non-root cgroups.

Lines are keyed by \$MAJ:\$MIN device numbers and not ordered. The following nested keys are defined.

rbytes	Bytes read
wbytes	Bytes written
rios	Number of read IOs
wios	Number of write IOs
dbytes	Bytes discarded
dios	Number of discard IOs

An example read output follows:

```
8:16 rbytes=1459200 wbytes=314773504 rios=192 wios=353
↳dbytes=0 dios=0
8:0 rbytes=90430464 wbytes=299008000 rios=8950 wios=1252
↳dbytes=50331648 dios=3021
```

**io.cost.qos** A read-write nested-keyed file with exists only on the root cgroup.

This file configures the Quality of Service of the IO cost model based controller (CONFIG\_BLK\_CGROUP\_IOCOST) which currently implements “io.weight” proportional control. Lines are keyed by \$MAJ:\$MIN device numbers and not ordered. The line for a given device is populated on the first write for the device on “io.cost.qos” or “io.cost.model” . The following nested keys are defined.

enable	Weight-based control enable
ctrl	“auto” or “user”
rpct	Read latency percentile [0, 100]
rlat	Read latency threshold
wpct	Write latency percentile [0, 100]
wlat	Write latency threshold
min	Minimum scaling percentage [1, 10000]
max	Maximum scaling percentage [1, 10000]

The controller is disabled by default and can be enabled by setting “enable” to 1. “rpct” and “wpct” parameters default to zero and the controller uses internal device saturation state to adjust the overall IO rate between “min” and “max” .

When a better control quality is needed, latency QoS parameters can be configured. For example:

```
8:16 enable=1 ctrl=auto rpct=95.00 rlat=75000 wpct=95.00
↪wlat=150000 min=50.00 max=150.0
```

shows that on `sdb`, the controller is enabled, will consider the device saturated if the 95th percentile of read completion latencies is above 75ms or write 150ms, and adjust the overall IO issue rate between 50% and 150% accordingly.

The lower the saturation point, the better the latency QoS at the cost of aggregate bandwidth. The narrower the allowed adjustment range between “min” and “max”, the more conformant to the cost model the IO behavior. Note that the IO issue base rate may be far off from 100% and setting “min” and “max” blindly can lead to a significant loss of device capacity or control quality. “min” and “max” are useful for regulating devices which show wide temporary behavior changes - e.g. a ssd which accepts writes at the line speed for a while and then completely stalls for multiple seconds.

When “ctrl” is “auto”, the parameters are controlled by the kernel and may change automatically. Setting “ctrl” to “user” or setting any of the percentile and latency parameters puts it into “user” mode and disables the automatic changes. The automatic mode can be restored by setting “ctrl” to “auto” .

**io.cost.model** A read-write nested-keyed file with exists only on the root cgroup.

This file configures the cost model of the IO cost model based controller (`CONFIG_BLK_CGROUP_IOCOST`) which currently implements “io.weight” proportional control. Lines are keyed by `$MAJ:$MIN` device numbers and not ordered. The line for a given device is populated on the first write for the device on “io.cost.qos” or “io.cost.model” . The following nested keys are defined.

ctrl	“auto” or “user”
model	The cost model in use - “linear”

When “ctrl” is “auto”, the kernel may change all parameters dynamically. When “ctrl” is set to “user” or any other parameters are written to, “ctrl” become “user” and the automatic changes are disabled.

When “model” is “linear”, the following model parameters are defined.

[r w]bps	The maximum sequential IO throughput
[r w]seqiops	The maximum 4k sequential IOs per second
[r w]randiops	The maximum 4k random IOs per second

From the above, the builtin linear model determines the base costs of a sequential and random IO and the cost coefficient for the IO size. While simple, this model can cover most common device classes acceptably.

The IO cost model isn't expected to be accurate in absolute sense and is scaled to the device behavior dynamically.

If needed, tools/cgroup/iocost\_coef\_gen.py can be used to generate device-specific coefficients.

**io.weight** A read-write flat-keyed file which exists on non-root cgroups. The default is “default 100” .

The first line is the default weight applied to devices without specific override. The rest are overrides keyed by \$MAJ:\$MIN device numbers and not ordered. The weights are in the range [1, 10000] and specifies the relative amount IO time the cgroup can use in relation to its siblings.

The default weight can be updated by writing either “default \$WEIGHT” or simply “\$WEIGHT” . Overrides can be set by writing “\$MAJ:\$MIN \$WEIGHT” and unset by writing “\$MAJ:\$MIN default” .

An example read output follows:

```
default 100
8:16 200
8:0 50
```

**io.max** A read-write nested-keyed file which exists on non-root cgroups.

BPS and IOPS based IO limit. Lines are keyed by \$MAJ:\$MIN device numbers and not ordered. The following nested keys are defined.

rbps	Max read bytes per second
wbps	Max write bytes per second
riops	Max read IO operations per second
wiops	Max write IO operations per second

When writing, any number of nested key-value pairs can be specified in any order. “max” can be specified as the value to remove a specific limit. If the same key is specified multiple times, the outcome is undefined.

BPS and IOPS are measured in each IO direction and IOs are delayed if limit is reached. Temporary bursts are allowed.

Setting read limit at 2M BPS and write at 120 IOPS for 8:16:

```
echo "8:16 rbps=2097152 wiops=120" > io.max
```

Reading returns the following:

```
8:16 rbps=2097152 wbps=max riops=max wiops=120
```

Write IOPS limit can be removed by writing the following:

```
echo "8:16 wiops=max" > io.max
```

Reading now returns the following:

```
8:16 rbps=2097152 wbps=max riops=max wiops=max
```

**io.pressure** A read-only nested-key file which exists on non-root cgroups.

Shows pressure stall information for IO. See Documentation/accounting/psi.rst for details.

### Writeback

Page cache is dirtied through buffered writes and shared mmaps and written asynchronously to the backing filesystem by the writeback mechanism. Writeback sits between the memory and IO domains and regulates the proportion of dirty memory by balancing dirtying and write IOs.

The io controller, in conjunction with the memory controller, implements control of page cache writeback IOs. The memory controller defines the memory domain that dirty memory ratio is calculated and maintained for and the io controller defines the io domain which writes out dirty pages for the memory domain. Both system-wide and per-cgroup dirty memory states are examined and the more restrictive of the two is enforced.

cgroup writeback requires explicit support from the underlying filesystem. Currently, cgroup writeback is implemented on ext2, ext4 and btrfs. On other filesystems, all writeback IOs are attributed to the root cgroup.

There are inherent differences in memory and writeback management which affects how cgroup ownership is tracked. Memory is tracked per page while writeback per inode. For the purpose of writeback, an inode is assigned to a cgroup and all IO requests to write dirty pages from the inode are attributed to that cgroup.

As cgroup ownership for memory is tracked per page, there can be pages which are associated with different cgroups than the one the inode is associated with. These are called foreign pages. The writeback constantly keeps track of foreign pages and, if a particular foreign cgroup becomes the majority over a certain period of time, switches the ownership of the inode to that cgroup.

While this model is enough for most use cases where a given inode is mostly dirtied by a single cgroup even when the main writing cgroup changes over time, use cases where multiple cgroups write to a single inode simultaneously are not supported well. In such circumstances, a significant portion of IOs are likely to be attributed incorrectly. As memory controller assigns page ownership on the first use and doesn't update it until the page is released, even if writeback strictly follows page ownership, multiple cgroups dirtying overlapping areas wouldn't work as expected. It's recommended to avoid such usage patterns.

The `sysctl` knobs which affect writeback behavior are applied to cgroup writeback as follows.

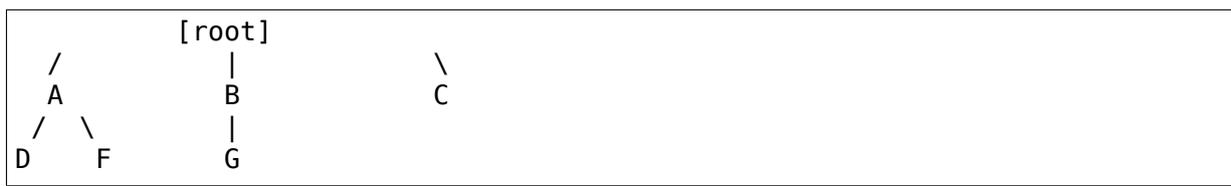
**vm.dirty\_background\_ratio, vm.dirty\_ratio** These ratios apply the same to cgroup writeback with the amount of available memory capped by limits imposed by the memory controller and system-wide clean memory.

**vm.dirty\_background\_bytes, vm.dirty\_bytes** For cgroup writeback, this is calculated into ratio against total available memory and applied the same way as `vm.dirty[_background]_ratio`.

## IO Latency

This is a cgroup v2 controller for IO workload protection. You provide a group with a latency target, and if the average latency exceeds that target the controller will throttle any peers that have a lower latency target than the protected workload.

The limits are only applied at the peer level in the hierarchy. This means that in the diagram below, only groups A, B, and C will influence each other, and groups D and F will influence each other. Group G will influence nobody:



So the ideal way to configure this is to set `io.latency` in groups A, B, and C. Generally you do not want to set a value lower than the latency your device supports. Experiment to find the value that works best for your workload. Start at higher than the expected latency for your device and watch the `avg_lat` value in `io.stat` for your workload group to get an idea of the latency you see during normal operation. Use the `avg_lat` value as a basis for your real setting, setting at 10-15% higher than the value in `io.stat`.

## How IO Latency Throttling Works

`io.latency` is work conserving; so as long as everybody is meeting their latency target the controller doesn't do anything. Once a group starts missing its target it begins throttling any peer group that has a higher target than itself. This throttling takes 2 forms:

- Queue depth throttling. This is the number of outstanding IO's a group is allowed to have. We will clamp down relatively quickly, starting at no limit and going all the way down to 1 IO at a time.
- Artificial delay induction. There are certain types of IO that cannot be throttled without possibly adversely affecting higher priority groups. This includes swapping and metadata IO. These types of IO are allowed to occur normally, however they are "charged" to the originating group. If the originating group is being throttled you will see the `use_delay` and `delay` fields in `io.stat` increase. The delay value is how many microseconds that are being added to

any process that runs in this group. Because this number can grow quite large if there is a lot of swapping or metadata IO occurring we limit the individual delay events to 1 second at a time.

Once the victimized group starts meeting its latency target again it will start unthrottling any peer groups that were throttled previously. If the victimized group simply stops doing IO the global counter will unthrottle appropriately.

### IO Latency Interface Files

**io.latency** This takes a similar format as the other controllers.

“MAJOR:MINOR target=<target time in microseconds”

**io.stat** If the controller is enabled you will see extra stats in io.stat in addition to the normal ones.

**depth** This is the current queue depth for the group.

**avg\_lat** This is an exponential moving average with a decay rate of  $1/\text{exp}$  bound by the sampling interval. The decay rate interval can be calculated by multiplying the win value in io.stat by the corresponding number of samples based on the win value.

**win** The sampling window size in milliseconds. This is the minimum duration of time between evaluation events. Windows only elapse with IO activity. Idle periods extend the most recent window.

### 28.5.4 PID

The process number controller is used to allow a cgroup to stop any new tasks from being fork()'d or clone()'d after a specified limit is reached.

The number of tasks in a cgroup can be exhausted in ways which other controllers cannot prevent, thus warranting its own controller. For example, a fork bomb is likely to exhaust the number of tasks before hitting memory restrictions.

Note that PIDs used in this controller refer to TIDs, process IDs as used by the kernel.

### PID Interface Files

**pids.max** A read-write single value file which exists on non-root cgroups. The default is “max” .

Hard limit of number of processes.

**pids.current** A read-only single value file which exists on all cgroups.

The number of processes currently in the cgroup and its descendants.

Organisational operations are not blocked by cgroup policies, so it is possible to have `pids.current > pids.max`. This can be done by either setting the limit to be smaller than `pids.current`, or attaching enough processes to the cgroup such that `pids.current` is larger than `pids.max`. However, it is not possible to violate a cgroup PID policy through `fork()` or `clone()`. These will return `-EAGAIN` if the creation of a new process would cause a cgroup policy to be violated.

### 28.5.5 Cpuset

The “cpuset” controller provides a mechanism for constraining the CPU and memory node placement of tasks to only the resources specified in the cpuset interface files in a task’s current cgroup. This is especially valuable on large NUMA systems where placing jobs on properly sized subsets of the systems with careful processor and memory placement to reduce cross-node memory access and contention can improve overall system performance.

The “cpuset” controller is hierarchical. That means the controller cannot use CPUs or memory nodes not allowed in its parent.

#### Cpuset Interface Files

**cpuset.cpus** A read-write multiple values file which exists on non-root cpuset-enabled cgroups.

It lists the requested CPUs to be used by tasks within this cgroup. The actual list of CPUs to be granted, however, is subjected to constraints imposed by its parent and can differ from the requested CPUs.

The CPU numbers are comma-separated numbers or ranges. For example:

```
# cat cpuset.cpus
0-4,6,8-10
```

An empty value indicates that the cgroup is using the same setting as the nearest cgroup ancestor with a non-empty “cpuset.cpus” or all the available CPUs if none is found.

The value of “cpuset.cpus” stays constant until the next update and won’t be affected by any CPU hotplug events.

**cpuset.cpus.effective** A read-only multiple values file which exists on all cpuset-enabled cgroups.

It lists the onlined CPUs that are actually granted to this cgroup by its parent. These CPUs are allowed to be used by tasks within the current cgroup.

If “cpuset.cpus” is empty, the “cpuset.cpus.effective” file shows all the CPUs from the parent cgroup that can be available to be used by this cgroup. Otherwise, it should be a subset of “cpuset.cpus” unless none of the CPUs listed in “cpuset.cpus” can be granted. In this case, it will be treated just like an empty “cpuset.cpus” .

Its value will be affected by CPU hotplug events.

**cpuset.mems** A read-write multiple values file which exists on non-root cpuset-enabled cgroups.

It lists the requested memory nodes to be used by tasks within this cgroup. The actual list of memory nodes granted, however, is subjected to constraints imposed by its parent and can differ from the requested memory nodes.

The memory node numbers are comma-separated numbers or ranges. For example:

```
# cat cpuset.mems
0-1,3
```

An empty value indicates that the cgroup is using the same setting as the nearest cgroup ancestor with a non-empty “cpuset.mems” or all the available memory nodes if none is found.

The value of “cpuset.mems” stays constant until the next update and won't be affected by any memory nodes hotplug events.

**cpuset.mems.effective** A read-only multiple values file which exists on all cpuset-enabled cgroups.

It lists the onlined memory nodes that are actually granted to this cgroup by its parent. These memory nodes are allowed to be used by tasks within the current cgroup.

If “cpuset.mems” is empty, it shows all the memory nodes from the parent cgroup that will be available to be used by this cgroup. Otherwise, it should be a subset of “cpuset.mems” unless none of the memory nodes listed in “cpuset.mems” can be granted. In this case, it will be treated just like an empty “cpuset.mems” .

Its value will be affected by memory nodes hotplug events.

**cpuset.cpus.partition** A read-write single value file which exists on non-root cpuset-enabled cgroups. This flag is owned by the parent cgroup and is not delegatable.

It accepts only the following input values when written to.

“root” - a partition root “member” - a non-root member of a partition

When set to be a partition root, the current cgroup is the root of a new partition or scheduling domain that comprises itself and all its descendants except those that are separate partition roots themselves and their descendants. The root cgroup is always a partition root.

There are constraints on where a partition root can be set. It can only be set in a cgroup if all the following conditions are true.

- 1) The “cpuset.cpus” is not empty and the list of CPUs are exclusive, i.e. they are not shared by any of its siblings.
- 2) The parent cgroup is a partition root.

- 3) The “cpuset.cpus” is also a proper subset of the parent’s “cpuset.cpus.effective” .
- 4) There is no child cgroups with cpuset enabled. This is for eliminating corner cases that have to be handled if such a condition is allowed.

Setting it to partition root will take the CPUs away from the effective CPUs of the parent cgroup. Once it is set, this file cannot be reverted back to “member” if there are any child cgroups with cpuset enabled.

A parent partition cannot distribute all its CPUs to its child partitions. There must be at least one cpu left in the parent partition.

Once becoming a partition root, changes to “cpuset.cpus” is generally allowed as long as the first condition above is true, the change will not take away all the CPUs from the parent partition and the new “cpuset.cpus” value is a superset of its children’s “cpuset.cpus” values.

Sometimes, external factors like changes to ancestors’ “cpuset.cpus” or cpu hotplug can cause the state of the partition root to change. On read, the “cpuset.sched.partition” file can show the following values.

“member” Non-root member of a partition “root” Partition root “root invalid” Invalid partition root

It is a partition root if the first 2 partition root conditions above are true and at least one CPU from “cpuset.cpus” is granted by the parent cgroup.

A partition root can become invalid if none of CPUs requested in “cpuset.cpus” can be granted by the parent cgroup or the parent cgroup is no longer a partition root itself. In this case, it is not a real partition even though the restriction of the first partition root condition above will still apply. The cpu affinity of all the tasks in the cgroup will then be associated with CPUs in the nearest ancestor partition.

An invalid partition root can be transitioned back to a real partition root if at least one of the requested CPUs can now be granted by its parent. In this case, the cpu affinity of all the tasks in the formerly invalid partition will be associated to the CPUs of the newly formed partition. Changing the partition state of an invalid partition root to “member” is always allowed even if child cpusets are present.

### 28.5.6 Device controller

Device controller manages access to device files. It includes both creation of new device files (using `mknod`), and access to the existing device files.

Cgroup v2 device controller has no interface files and is implemented on top of cgroup BPF. To control access to device files, a user may create bpf programs of the `BPF_CGROUP_DEVICE` type and attach them to cgroups. On an attempt to access a device file, corresponding BPF programs will be executed, and depending on the return value the attempt will succeed or fail with `-EPERM`.

A `BPF_CGROUP_DEVICE` program takes a pointer to the `bpf_cgroup_dev_ctx` structure, which describes the device access attempt: access type (`mknod/read/write`) and device (type, major and minor numbers). If the program returns 0, the attempt fails with `-EPERM`, otherwise it succeeds.

An example of `BPF_CGROUP_DEVICE` program may be found in the kernel source tree in the `tools/testing/selftests/bpf/dev_cgroup.c` file.

### 28.5.7 RDMA

The “rdma” controller regulates the distribution and accounting of of RDMA resources.

#### RDMA Interface Files

**rdma.max** A readwrite nested-keyed file that exists for all the cgroups except root that describes current configured resource limit for a RDMA/IB device.

Lines are keyed by device name and are not ordered. Each line contains space separated resource name and its configured limit that can be distributed.

The following nested keys are defined.

<code>hca_handle</code>	Maximum number of HCA Handles
<code>hca_object</code>	Maximum number of HCA Objects

An example for `mlx4` and `ocrdma` device follows:

```
mlx4_0 hca_handle=2 hca_object=2000
ocrdma1 hca_handle=3 hca_object=max
```

**rdma.current** A read-only file that describes current resource usage. It exists for all the cgroup except root.

An example for `mlx4` and `ocrdma` device follows:

```
mlx4_0 hca_handle=1 hca_object=20
ocrdma1 hca_handle=1 hca_object=23
```

## 28.5.8 HugeTLB

The HugeTLB controller allows to limit the HugeTLB usage per control group and enforces the controller limit during page fault.

### HugeTLB Interface Files

**hugetlb.<hugepagesize>.current** Show current usage for “hugepagesize” hugetlb. It exists for all the cgroup except root.

**hugetlb.<hugepagesize>.max** Set/show the hard limit of “hugepagesize” hugetlb usage. The default value is “max” . It exists for all the cgroup except root.

**hugetlb.<hugepagesize>.events** A read-only flat-keyed file which exists on non-root cgroups.

**max** The number of allocation failure due to HugeTLB limit

**hugetlb.<hugepagesize>.events.local** Similar to hugetlb.<hugepagesize>.events but the fields in the file are local to the cgroup i.e. not hierarchical. The file modified event generated on this file reflects only the local events.

## 28.5.9 Misc

### perf\_event

perf\_event controller, if not mounted on a legacy hierarchy, is automatically enabled on the v2 hierarchy so that perf events can always be filtered by cgroup v2 path. The controller can still be moved to a legacy hierarchy after v2 hierarchy is populated.

## 28.5.10 Non-normative information

This section contains information that isn't considered to be a part of the stable kernel API and so is subject to change.

### CPU controller root cgroup process behaviour

When distributing CPU cycles in the root cgroup each thread in this cgroup is treated as if it was hosted in a separate child cgroup of the root cgroup. This child cgroup weight is dependent on its thread nice level.

For details of this mapping see sched\_prio\_to\_weight array in kernel/sched/core.c file (values from this array should be scaled appropriately so the neutral - nice 0 - value is 100 instead of 1024).

### IO controller root cgroup process behaviour

Root cgroup processes are hosted in an implicit leaf child node. When distributing IO resources this implicit child node is taken into account as if it was a normal child cgroup of the root cgroup with a weight value of 200.

## 28.6 Namespace

### 28.6.1 Basics

cgroup namespace provides a mechanism to virtualize the view of the “/proc/\$PID/cgroup” file and cgroup mounts. The CLONE\_NEWCGROUP clone flag can be used with clone(2) and unshare(2) to create a new cgroup namespace. The process running inside the cgroup namespace will have its “/proc/\$PID/cgroup” output restricted to cgroupns root. The cgroupns root is the cgroup of the process at the time of creation of the cgroup namespace.

Without cgroup namespace, the “/proc/\$PID/cgroup” file shows the complete path of the cgroup of a process. In a container setup where a set of cgroups and namespaces are intended to isolate processes the “/proc/\$PID/cgroup” file may leak potential system level information to the isolated processes. For Example:

```
# cat /proc/self/cgroup
0::/batchjobs/container_id1
```

The path ‘/batchjobs/container\_id1’ can be considered as system-data and undesirable to expose to the isolated processes. cgroup namespace can be used to restrict visibility of this path. For example, before creating a cgroup namespace, one would see:

```
# ls -l /proc/self/ns/cgroup
lrwxrwxrwx 1 root root 0 2014-07-15 10:37 /proc/self/ns/cgroup ->┘
└─cgroup:[4026531835]
# cat /proc/self/cgroup
0::/batchjobs/container_id1
```

After unsharing a new namespace, the view changes:

```
# ls -l /proc/self/ns/cgroup
lrwxrwxrwx 1 root root 0 2014-07-15 10:35 /proc/self/ns/cgroup ->┘
└─cgroup:[4026532183]
# cat /proc/self/cgroup
0::/
```

When some thread from a multi-threaded process unshares its cgroup namespace, the new cgroupns gets applied to the entire process (all the threads). This is natural for the v2 hierarchy; however, for the legacy hierarchies, this may be unexpected.

A cgroup namespace is alive as long as there are processes inside or mounts pinning it. When the last usage goes away, the cgroup namespace is destroyed. The cgroupns root and the actual cgroups remain.

## 28.6.2 The Root and Views

The ‘cgroupns root’ for a cgroup namespace is the cgroup in which the process calling `unshare(2)` is running. For example, if a process in `/batchjobs/container_id1` cgroup calls `unshare`, cgroup `/batchjobs/container_id1` becomes the cgroupns root. For the `init_cgroup_ns`, this is the real root ( `/` ) cgroup.

The cgroupns root cgroup does not change even if the namespace creator process later moves to a different cgroup:

```
# ~/unshare -c # unshare cgroupns in some cgroup
# cat /proc/self/cgroup
0::/
# mkdir sub_cgrp_1
# echo 0 > sub_cgrp_1/cgroup.procs
# cat /proc/self/cgroup
0:./sub_cgrp_1
```

Each process gets its namespace-specific view of “`/proc/$PID/cgroup`”

Processes running inside the cgroup namespace will be able to see cgroup paths (in `/proc/self/cgroup`) only inside their root cgroup. From within an unshared cgroupns:

```
# sleep 100000 &
[1] 7353
# echo 7353 > sub_cgrp_1/cgroup.procs
# cat /proc/7353/cgroup
0:./sub_cgrp_1
```

From the initial cgroup namespace, the real cgroup path will be visible:

```
$ cat /proc/7353/cgroup
0:./batchjobs/container_id1/sub_cgrp_1
```

From a sibling cgroup namespace (that is, a namespace rooted at a different cgroup), the cgroup path relative to its own cgroup namespace root will be shown. For instance, if PID 7353’s cgroup namespace root is at `/batchjobs/container_id2`, then it will see:

```
# cat /proc/7353/cgroup
0:../container_id2/sub_cgrp_1
```

Note that the relative path always starts with `/` to indicate that its relative to the cgroup namespace root of the caller.

### 28.6.3 Migration and setns(2)

Processes inside a cgroup namespace can move into and out of the namespace root if they have proper access to external cgroups. For example, from inside a namespace with cgroupns root at `/batchjobs/container_id1`, and assuming that the global hierarchy is still accessible inside cgroupns:

```
# cat /proc/7353/cgroup
0::/sub_cgrp_1
# echo 7353 > batchjobs/container_id2/cgroup.procs
# cat /proc/7353/cgroup
0::/../container_id2
```

Note that this kind of setup is not encouraged. A task inside cgroup namespace should only be exposed to its own cgroupns hierarchy.

setns(2) to another cgroup namespace is allowed when:

- (a) the process has `CAP_SYS_ADMIN` against its current user namespace
- (b) the process has `CAP_SYS_ADMIN` against the target cgroup namespace's users

No implicit cgroup changes happen with attaching to another cgroup namespace. It is expected that the someone moves the attaching process under the target cgroup namespace root.

### 28.6.4 Interaction with Other Namespaces

Namespace specific cgroup hierarchy can be mounted by a process running inside a non-init cgroup namespace:

```
# mount -t cgroup2 none $MOUNT_POINT
```

This will mount the unified cgroup hierarchy with cgroupns root as the filesystem root. The process needs `CAP_SYS_ADMIN` against its user and mount namespaces.

The virtualization of `/proc/self/cgroup` file combined with restricting the view of cgroup hierarchy by namespace-private cgroupfs mount provides a properly isolated cgroup view inside the container.

## 28.7 Information on Kernel Programming

This section contains kernel programming information in the areas where interacting with cgroup is necessary. cgroup core and controllers are not covered.

### 28.7.1 Filesystem Support for Writeback

A filesystem can support cgroup writeback by updating `address_space_operations->writepage[s]()` to annotate bio's using the following two functions.

**wbc\_init\_bio(@wbc, @bio)** Should be called for each bio carrying writeback data and associates the bio with the inode's owner cgroup and the corresponding request queue. This must be called after a queue (device) has been associated with the bio and before submission.

**wbc\_account\_cgroup\_owner(@wbc, @page, @bytes)** Should be called for each data segment being written out. While this function doesn't care exactly when it's called during the writeback session, it's the easiest and most natural to call it as data segments are added to a bio.

With writeback bio's annotated, cgroup support can be enabled per `super_block` by setting `SB_I_CGROUPWB` in `->s_iflags`. This allows for selective disabling of cgroup writeback support which is helpful when certain filesystem features, e.g. journaled data mode, are incompatible.

`wbc_init_bio()` binds the specified bio to its cgroup. Depending on the configuration, the bio may be executed at a lower priority and if the writeback session is holding shared resources, e.g. a journal entry, may lead to priority inversion. There is no one easy solution for the problem. Filesystems can try to work around specific problem cases by skipping `wbc_init_bio()` and using `bio_associate_blkcg()` directly.

## 28.8 Deprecated v1 Core Features

- Multiple hierarchies including named ones are not supported.
- All v1 mount options are not supported.
- The "tasks" file is removed and "cgroup.procs" is not sorted.
- "cgroup.clone\_children" is removed.
- `/proc/cgroups` is meaningless for v2. Use "cgroup.controllers" file at the root instead.

## 28.9 Issues with v1 and Rationales for v2

### 28.9.1 Multiple Hierarchies

cgroup v1 allowed an arbitrary number of hierarchies and each hierarchy could host any number of controllers. While this seemed to provide a high level of flexibility, it wasn't useful in practice.

For example, as there is only one instance of each controller, utility type controllers such as freezer which can be useful in all hierarchies could only be used in one. The issue is exacerbated by the fact that controllers couldn't be moved to another

hierarchy once hierarchies were populated. Another issue was that all controllers bound to a hierarchy were forced to have exactly the same view of the hierarchy. It wasn't possible to vary the granularity depending on the specific controller.

In practice, these issues heavily limited which controllers could be put on the same hierarchy and most configurations resorted to putting each controller on its own hierarchy. Only closely related ones, such as the `cpu` and `cpuacct` controllers, made sense to be put on the same hierarchy. This often meant that userland ended up managing multiple similar hierarchies repeating the same steps on each hierarchy whenever a hierarchy management operation was necessary.

Furthermore, support for multiple hierarchies came at a steep cost. It greatly complicated `cgroup` core implementation but more importantly the support for multiple hierarchies restricted how `cgroup` could be used in general and what controllers was able to do.

There was no limit on how many hierarchies there might be, which meant that a thread's `cgroup` membership couldn't be described in finite length. The key might contain any number of entries and was unlimited in length, which made it highly awkward to manipulate and led to addition of controllers which existed only to identify membership, which in turn exacerbated the original problem of proliferating number of hierarchies.

Also, as a controller couldn't have any expectation regarding the topologies of hierarchies other controllers might be on, each controller had to assume that all other controllers were attached to completely orthogonal hierarchies. This made it impossible, or at least very cumbersome, for controllers to cooperate with each other.

In most use cases, putting controllers on hierarchies which are completely orthogonal to each other isn't necessary. What usually is called for is the ability to have differing levels of granularity depending on the specific controller. In other words, hierarchy may be collapsed from leaf towards root when viewed from specific controllers. For example, a given configuration might not care about how memory is distributed beyond a certain level while still wanting to control how CPU cycles are distributed.

### 28.9.2 Thread Granularity

`cgroup v1` allowed threads of a process to belong to different `cgroups`. This didn't make sense for some controllers and those controllers ended up implementing different ways to ignore such situations but much more importantly it blurred the line between API exposed to individual applications and system management interface.

Generally, in-process knowledge is available only to the process itself; thus, unlike service-level organization of processes, categorizing threads of a process requires active participation from the application which owns the target process.

`cgroup v1` had an ambiguously defined delegation model which got abused in combination with thread granularity. `cgroups` were delegated to individual applications so that they can create and manage their own sub-hierarchies and control resource distributions along them. This effectively raised `cgroup` to the status of a `syscall`-like API exposed to lay programs.

First of all, cgroup has a fundamentally inadequate interface to be exposed this way. For a process to access its own knobs, it has to extract the path on the target hierarchy from `/proc/self/cgroup`, construct the path by appending the name of the knob to the path, open and then read and/or write to it. This is not only extremely clunky and unusual but also inherently racy. There is no conventional way to define transaction across the required steps and nothing can guarantee that the process would actually be operating on its own sub-hierarchy.

cgroup controllers implemented a number of knobs which would never be accepted as public APIs because they were just adding control knobs to system-management pseudo filesystem. cgroup ended up with interface knobs which were not properly abstracted or refined and directly revealed kernel internal details. These knobs got exposed to individual applications through the ill-defined delegation mechanism effectively abusing cgroup as a shortcut to implementing public APIs without going through the required scrutiny.

This was painful for both userland and kernel. Userland ended up with misbehaving and poorly abstracted interfaces and kernel exposing and locked into constructs inadvertently.

### **28.9.3 Competition Between Inner Nodes and Threads**

cgroup v1 allowed threads to be in any cgroups which created an interesting problem where threads belonging to a parent cgroup and its children cgroups competed for resources. This was nasty as two different types of entities competed and there was no obvious way to settle it. Different controllers did different things.

The cpu controller considered threads and cgroups as equivalents and mapped nice levels to cgroup weights. This worked for some cases but fell flat when children wanted to be allocated specific ratios of CPU cycles and the number of internal threads fluctuated - the ratios constantly changed as the number of competing entities fluctuated. There also were other issues. The mapping from nice level to weight wasn't obvious or universal, and there were various other knobs which simply weren't available for threads.

The io controller implicitly created a hidden leaf node for each cgroup to host the threads. The hidden leaf had its own copies of all the knobs with `leaf_` prefixed. While this allowed equivalent control over internal threads, it was with serious drawbacks. It always added an extra layer of nesting which wouldn't be necessary otherwise, made the interface messy and significantly complicated the implementation.

The memory controller didn't have a way to control what happened between internal tasks and child cgroups and the behavior was not clearly defined. There were attempts to add ad-hoc behaviors and knobs to tailor the behavior to specific workloads which would have led to problems extremely difficult to resolve in the long term.

Multiple controllers struggled with internal tasks and came up with different ways to deal with it; unfortunately, all the approaches were severely flawed and, furthermore, the widely different behaviors made cgroup as a whole highly inconsistent.

This clearly is a problem which needs to be addressed from cgroup core in a uniform way.

### 28.9.4 Other Interface Issues

cgroup v1 grew without oversight and developed a large number of idiosyncrasies and inconsistencies. One issue on the cgroup core side was how an empty cgroup was notified - a userland helper binary was forked and executed for each event. The event delivery wasn't recursive or delegatable. The limitations of the mechanism also led to in-kernel event delivery filtering mechanism further complicating the interface.

Controller interfaces were problematic too. An extreme example is controllers completely ignoring hierarchical organization and treating all cgroups as if they were all located directly under the root cgroup. Some controllers exposed a large amount of inconsistent implementation details to userland.

There also was no consistency across controllers. When a new cgroup was created, some controllers defaulted to not imposing extra restrictions while others disallowed any resource usage until explicitly configured. Configuration knobs for the same type of control used widely differing naming schemes and formats. Statistics and information knobs were named arbitrarily and used different formats and units even in the same controller.

cgroup v2 establishes common conventions where appropriate and updates controllers so that they expose minimal and consistent interfaces.

### 28.9.5 Controller Issues and Remedies

#### Memory

The original lower boundary, the soft limit, is defined as a limit that is per default unset. As a result, the set of cgroups that global reclaim prefers is opt-in, rather than opt-out. The costs for optimizing these mostly negative lookups are so high that the implementation, despite its enormous size, does not even provide the basic desirable behavior. First off, the soft limit has no hierarchical meaning. All configured groups are organized in a global rbtree and treated like equal peers, regardless where they are located in the hierarchy. This makes subtree delegation impossible. Second, the soft limit reclaim pass is so aggressive that it not just introduces high allocation latencies into the system, but also impacts system performance due to overreclaim, to the point where the feature becomes self-defeating.

The memory.low boundary on the other hand is a top-down allocated reserve. A cgroup enjoys reclaim protection when it's within its effective low, which makes delegation of subtrees possible. It also enjoys having reclaim pressure proportional to its overage when above its effective low.

The original high boundary, the hard limit, is defined as a strict limit that can not budge, even if the OOM killer has to be called. But this generally goes against the goal of making the most out of the available memory. The memory consumption of workloads varies during runtime, and that requires users to overcommit. But doing that with a strict upper limit requires either a fairly accurate prediction of the working set size or adding slack to the limit. Since working set size estimation is hard and error prone, and getting it wrong results in OOM kills, most users tend to err on the side of a looser limit and end up wasting precious resources.

The `memory.high` boundary on the other hand can be set much more conservatively. When hit, it throttles allocations by forcing them into direct reclaim to work off the excess, but it never invokes the OOM killer. As a result, a high boundary that is chosen too aggressively will not terminate the processes, but instead it will lead to gradual performance degradation. The user can monitor this and make corrections until the minimal memory footprint that still gives acceptable performance is found.

In extreme cases, with many concurrent allocations and a complete breakdown of reclaim progress within the group, the high boundary can be exceeded. But even then it's mostly better to satisfy the allocation from the slack available in other groups or the rest of the system than killing the group. Otherwise, `memory.max` is there to limit this type of spillover and ultimately contain buggy or even malicious applications.

Setting the original `memory.limit_in_bytes` below the current usage was subject to a race condition, where concurrent charges could cause the limit setting to fail. `memory.max` on the other hand will first set the limit to prevent new charges, and then reclaim and OOM kill until the new limit is met - or the task writing to `memory.max` is killed.

The combined `memory+swap` accounting and limiting is replaced by real control over swap space.

The main argument for a combined `memory+swap` facility in the original cgroup design was that global or parental pressure would always be able to swap all anonymous memory of a child group, regardless of the child's own (possibly untrusted) configuration. However, untrusted groups can sabotage swapping by other means - such as referencing its anonymous memory in a tight loop - and an admin can not assume full swappability when overcommitting untrusted jobs.

For trusted jobs, on the other hand, a combined counter is not an intuitive userspace interface, and it flies in the face of the idea that cgroup controllers should account and limit specific physical resources. Swap space is a resource like all others in the system, and that's why unified hierarchy allows distributing it separately.



## 29.1 Introduction

This is the client VFS module for the SMB3 NAS protocol as well as for older dialects such as the Common Internet File System (CIFS) protocol which was the successor to the Server Message Block (SMB) protocol, the native file sharing mechanism for most early PC operating systems. New and improved versions of CIFS are now called SMB2 and SMB3. Use of SMB3 (and later, including SMB3.1.1) is strongly preferred over using older dialects like CIFS due to security reasons. All modern dialects, including the most recent, SMB3.1.1 are supported by the CIFS VFS module. The SMB3 protocol is implemented and supported by all major file servers such as all modern versions of Windows (including Windows 2016 Server), as well as by Samba (which provides excellent CIFS/SMB2/SMB3 server support and tools for Linux and many other operating systems). Apple systems also support SMB3 well, as do most Network Attached Storage vendors, so this network filesystem client can mount to a wide variety of systems. It also supports mounting to the cloud (for example Microsoft Azure), including the necessary security features.

The intent of this module is to provide the most advanced network file system function for SMB3 compliant servers, including advanced security features, excellent parallelized high performance i/o, better POSIX compliance, secure per-user session establishment, encryption, high performance safe distributed caching (leases/oplocks), optional packet signing, large files, Unicode support and other internationalization improvements. Since both Samba server and this filesystem client support the CIFS Unix extensions (and in the future SMB3 POSIX extensions), the combination can provide a reasonable alternative to other network and cluster file systems for fileserving in some Linux to Linux environments, not just in Linux to Windows (or Linux to Mac) environments.

This filesystem has a mount utility (`mount.cifs`) and various user space tools (including `smbinfo` and `setcifsacl`) that can be obtained from

<https://git.samba.org/?p=cifs-utils.git>

or

`git://git.samba.org/cifs-utils.git`

mount.cifs should be installed in the directory with the other mount helpers.

For more information on the module see the project wiki page at

<https://wiki.samba.org/index.php/LinuxCIFS>

and

[https://wiki.samba.org/index.php/LinuxCIFS\\_utils](https://wiki.samba.org/index.php/LinuxCIFS_utils)

## 29.2 Usage

This module supports the SMB3 family of advanced network protocols (as well as older dialects, originally called “CIFS” or SMB1).

The CIFS VFS module for Linux supports many advanced network filesystem features such as hierarchical DFS like namespace, hardlinks, locking and more. It was designed to comply with the SNIA CIFS Technical Reference (which supersedes the 1992 X/Open SMB Standard) as well as to perform best practice practical interoperability with Windows 2000, Windows XP, Samba and equivalent servers. This code was developed in participation with the Protocol Freedom Information Foundation. CIFS and now SMB3 has now become a defacto standard for interop-erating between Macs and Windows and major NAS appliances.

Please see MS-SMB2 (for detailed SMB2/SMB3/SMB3.1.1 protocol specification) <http://protocolfreedom.org/> and <http://samba.org/samba/PFIF/> for more details.

For questions or bug reports please contact:

[smfrench@gmail.com](mailto:smfrench@gmail.com)

See the project page at: [https://wiki.samba.org/index.php/LinuxCIFS\\_utils](https://wiki.samba.org/index.php/LinuxCIFS_utils)

### 29.2.1 Build instructions

For Linux:

- 1) Download the kernel (e.g. from <http://www.kernel.org>) and change directory into the top of the kernel directory tree (e.g. /usr/src/linux-2.5.73)
- 2) make menuconfig (or make xconfig)
- 3) select cifs from within the network filesystem choices
- 4) save and exit
- 5) make

### 29.2.2 Installation instructions

If you have built the CIFS vfs as module (successfully) simply type `make modules_install` (or if you prefer, manually copy the file to the modules directory e.g. `/lib/modules/2.4.10-4GB/kernel/fs/cifs/cifs.ko`).

If you have built the CIFS vfs into the kernel itself, follow the instructions for your distribution on how to install a new kernel (usually you would simply type `make install`).

If you do not have the utility `mount.cifs` (in the Samba 4.x source tree and on the CIFS VFS web site) copy it to the same directory in which mount helpers reside (usually `/sbin`). Although the helper software is not required, `mount.cifs` is recommended. Most distros include a `cifs-utils` package that includes this utility so it is recommended to install this.

Note that running the Winbind pam/nss module (logon service) on all of your Linux clients is useful in mapping Uids and Gids consistently across the domain to the proper network user. The `mount.cifs` mount helper can be found at `cifs-utils.git` on `git.samba.org`

If `cifs` is built as a module, then the size and number of network buffers and maximum number of simultaneous requests to one server can be configured. Changing these from their defaults is not recommended. By executing `modinfo`:

```
modinfo kernel/fs/cifs/cifs.ko
```

on `kernel/fs/cifs/cifs.ko` the list of configuration changes that can be made at module initialization time (by running `insmod cifs.ko`) can be seen.

### 29.2.3 Recommendations

To improve security the SMB2.1 dialect or later (usually will get SMB3) is now the new default. To use old dialects (e.g. to mount Windows XP) use `vers=1.0` on mount (or `vers=2.0` for Windows Vista). Note that the CIFS (`vers=1.0`) is much older and less secure than the default dialect SMB3 which includes many advanced security features such as downgrade attack detection and encrypted shares and stronger signing and authentication algorithms. There are additional mount options that may be helpful for SMB3 to get improved POSIX behavior (NB: can use `vers=3.0` to force only SMB3, never 2.1):

```
    mfsymlinks and cifsacl and idsfromsid
```

### 29.2.4 Allowing User Mounts

To permit users to mount and unmount over directories they own is possible with the `cifs` vfs. A way to enable such mounting is to mark the `mount.cifs` utility as `suid` (e.g. `chmod +s /sbin/mount.cifs`). To enable users to unmount shares they mount requires

- 1) `mount.cifs` version 1.4 or later
- 2) an entry for the share in `/etc/fstab` indicating that a user may unmount it e.g.:

```
//server/usersharename /mnt/username cifs user 0 0
```

Note that when the `mount.cifs` utility is run `suid` (allowing user mounts), in order to reduce risks, the `nosuid` mount flag is passed in on `mount` to disallow execution of an `suid` program mounted on the remote target. When `mount` is executed as `root`, `nosuid` is not passed in by default, and execution of `suid` programs on the remote target would be enabled by default. This can be changed, as with `nfs` and other filesystems, by simply specifying `nosuid` among the mount options. For user mounts though to be able to pass the `suid` flag to `mount` requires rebuilding `mount.cifs` with the following flag: `CIFS_ALLOW_USR_SUID`

There is a corresponding manual page for `cifs` mounting in the Samba 3.0 and later source tree in `docs/manpages/mount.cifs.8`

### 29.2.5 Allowing User Unmounts

To permit users to `umount` directories that they have user mounted (see above), the utility `umount.cifs` may be used. It may be invoked directly, or if `umount.cifs` is placed in `/sbin`, `umount` can invoke the `cifs` `umount` helper (at least for most versions of the `umount` utility) for `umount` of `cifs` mounts, unless `umount` is invoked with `-i` (which will avoid invoking a `umount` helper). As with `mount.cifs`, to enable user unmounts `umount.cifs` must be marked as `suid` (e.g. `chmod +s /sbin/umount.cifs`) or equivalent (some distributions allow adding entries to a file to the `/etc/permissions` file to achieve the equivalent `suid` effect). For this utility to succeed the target path must be a `cifs` mount, and the `uid` of the current user must match the `uid` of the user who mounted the resource.

Also note that the customary way of allowing user mounts and unmounts is (instead of using `mount.cifs` and `umount.cifs` as `suid`) to add a line to the file `/etc/fstab` for each `//server/share` you wish to mount, but this can become unwieldy when potential mount targets include many or unpredictable UNC names.

### 29.2.6 Samba Considerations

Most current servers support SMB2.1 and SMB3 which are more secure, but there are useful protocol extensions for the older less secure CIFS dialect, so to get the maximum benefit if mounting using the older dialect (CIFS/SMB1), we recommend using a server that supports the SNIA CIFS Unix Extensions standard (e.g. almost any version of Samba ie version 2.2.5 or later) but the CIFS `vfs` works fine with a wide variety of CIFS servers. Note that `uid`, `gid` and file permissions will display default values if you do not have a server that supports the Unix extensions for CIFS (such as Samba 2.2.5 or later). To enable the Unix CIFS Extensions in the Samba server, add the line:

```
unix extensions = yes
```

to your `smb.conf` file on the server. Note that the following `smb.conf` settings are also useful (on the Samba server) when the majority of clients are Unix or Linux:

```
case sensitive = yes
delete readonly = yes
ea support = yes
```

Note that server ea support is required for supporting xattrs from the Linux cifs client, and that EA support is present in later versions of Samba (e.g. 3.0.6 and later (also EA support works in all versions of Windows, at least to shares on NTFS filesystems). Extended Attribute (xattr) support is an optional feature of most Linux filesystems which may require enabling via make menuconfig. Client support for extended attributes (user xattr) can be disabled on a per-mount basis by specifying `nouser_xattr` on mount.

The CIFS client can get and set POSIX ACLs (`getfacl`, `setfacl`) to Samba servers version 3.10 and later. Setting POSIX ACLs requires enabling both XATTR and then POSIX support in the CIFS configuration options when building the cifs module. POSIX ACL support can be disabled on a per mount basic by specifying `noacl` on mount.

Some administrators may want to change Samba's `smb.conf` `map archive` and `create mask` parameters from the default. Unless the `create mask` is changed newly created files can end up with an unnecessarily restrictive default mode, which may not be what you want, although if the CIFS Unix extensions are enabled on the server and client, subsequent `setattr` calls (e.g. `chmod`) can fix the mode. Note that creating special devices (`mknod`) remotely may require specifying a `mkdev` function to Samba if you are not using Samba 3.0.6 or later. For more information on these see the manual pages (`man smb.conf`) on the Samba server system. Note that the `cifs vfs`, unlike the `smbfs vfs`, does not read the `smb.conf` on the client system (the few optional settings are passed in on mount via `-o` parameters instead). Note that Samba 2.2.7 or later includes a fix that allows the CIFS VFS to delete open files (required for strict POSIX compliance). Windows Servers already supported this feature. Samba server does not allow symlinks that refer to files outside of the share, so in Samba versions prior to 3.0.6, most symlinks to files with absolute paths (ie beginning with slash) such as:

```
ln -s /mnt/foo bar
```

would be forbidden. Samba 3.0.6 server or later includes the ability to create such symlinks safely by converting unsafe symlinks (ie symlinks to server files that are outside of the share) to a samba specific format on the server that is ignored by local server applications and non-cifs clients and that will not be traversed by the Samba server). This is opaque to the Linux client application using the `cifs vfs`. Absolute symlinks will work to Samba 3.0.5 or later, but only for remote clients using the CIFS Unix extensions, and will be invisible to Windows clients and typically will not affect local applications running on the same server as Samba.

### 29.2.7 Use instructions

Once the CIFS VFS support is built into the kernel or installed as a module (cifs.ko), you can use mount syntax like the following to access Samba or Mac or Windows servers:

```
mount -t cifs //9.53.216.11/e$ /mnt -o username=myname,password=mypassword
```

Before -o the option -v may be specified to make the mount.cifs mount helper display the mount steps more verbosely. After -o the following commonly used cifs vfs specific options are supported:

```
username=<username>  
password=<password>  
domain=<domain name>
```

Other cifs mount options are described below. Use of TCP names (in addition to ip addresses) is available if the mount helper (mount.cifs) is installed. If you do not trust the server to which are mounted, or if you do not have cifs signing enabled (and the physical network is insecure), consider use of the standard mount options noexec and nosuid to reduce the risk of running an altered binary on your local system (downloaded from a hostile server or altered by a hostile router).

Although mounting using format corresponding to the CIFS URL specification is not possible in mount.cifs yet, it is possible to use an alternate format for the server and sharename (which is somewhat similar to NFS style mount syntax) instead of the more widely used UNC format (i.e. \servershare):

```
mount -t cifs tcp_name_of_server:share_name /mnt -o user=myname,  
↪pass=mypasswd
```

When using the mount helper mount.cifs, passwords may be specified via alternate mechanisms, instead of specifying it after -o using the normal pass= syntax on the command line: 1) By including it in a credential file. Specify credentials=filename as one of the mount options. Credential files contain two lines:

```
username=someuser  
password=your_password
```

- 2) By specifying the password in the PASSWD environment variable (similarly the user name can be taken from the USER environment variable).
- 3) By specifying the password in a file by name via PASSWD\_FILE
- 4) By specifying the password in a file by file descriptor via PASSWD\_FD

If no password is provided, mount.cifs will prompt for password entry

### 29.2.8 Restrictions

Servers must support either “pure-TCP” (port 445 TCP/IP CIFS connections) or RFC 1001/1002 support for “Netbios-Over-TCP/IP.” This is not likely to be a problem as most servers support this.

Valid filenames differ between Windows and Linux. Windows typically restricts filenames which contain certain reserved characters (e.g. the character `:` which is used to delimit the beginning of a stream name by Windows), while Linux allows a slightly wider set of valid characters in filenames. Windows servers can remap such characters when an explicit mapping is specified in the Server’s registry. Samba starting with version 3.10 will allow such filenames (ie those which contain valid Linux characters, which normally would be forbidden for Windows/CIFS semantics) as long as the server is configured for Unix Extensions (and the client has not disabled `/proc/fs/cifs/LinuxExtensionsEnabled`). In addition the mount option `mapposix` can be used on CIFS (`vers=1.0`) to force the mapping of illegal Windows/NTFS/SMB characters to a remap range (this mount parm is the default for SMB3). This remap (`mapposix`) range is also compatible with Mac (and “Services for Mac” on some older Windows).

### 29.2.9 CIFS VFS Mount Options

A partial list of the supported mount options follows:

**username** The user name to use when trying to establish the CIFS session.

**password** The user password. If the mount helper is installed, the user will be prompted for password if not supplied.

**ip** The ip address of the target server

**unc** The target server Universal Network Name (export) to mount.

**domain** Set the SMB/CIFS workgroup name prepended to the username during CIFS session establishment

**forceuid** Set the default uid for inodes to the uid passed in on mount. For mounts to servers which do support the CIFS Unix extensions, such as a properly configured Samba server, the server provides the uid, gid and mode so this parameter should not be specified unless the server and clients uid and gid numbering differ. If the server and client are in the same domain (e.g. running winbind or nss\_ldap) and the server supports the Unix Extensions then the uid and gid can be retrieved from the server (and uid and gid would not have to be specified on the mount. For servers which do not support the CIFS Unix extensions, the default uid (and gid) returned on lookup of existing files will be the uid (gid) of the person who executed the mount (root, except when `mount.cifs` is configured `setuid` for user mounts) unless the `uid=` (`gid`) mount option is specified. Also note that permission checks (authorization checks) on accesses to a file occur at the server, but there are cases in which an administrator may want to restrict at the client as well. For those servers which do not report a uid/gid owner (such as Windows), permissions can also

be checked at the client, and a crude form of client side permission checking can be enabled by specifying `file_mode` and `dir_mode` on the client. (default)

**forcegid** (similar to above but for the groupid instead of uid) (default)

**noforceuid** Fill in file owner information (uid) by requesting it from the server if possible. With this option, the value given in the `uid=` option (on mount) will only be used if the server can not support returning uids on inodes.

**noforcegid** (similar to above but for the group owner, gid, instead of uid)

**uid** Set the default uid for inodes, and indicate to the cifs kernel driver which local user mounted. If the server supports the unix extensions the default uid is not used to fill in the owner fields of inodes (files) unless the `forceuid` parameter is specified.

**gid** Set the default gid for inodes (similar to above).

**file\_mode** If CIFS Unix extensions are not supported by the server this overrides the default mode for file inodes.

**fsc** Enable local disk caching using FS-Cache (off by default). This option could be useful to improve performance on a slow link, heavily loaded server and/or network where reading from the disk is faster than reading from the server (over the network). This could also impact scalability positively as the number of calls to the server are reduced. However, local caching is not suitable for all workloads for e.g. read-once type workloads. So, you need to consider carefully your workload/scenario before using this option. Currently, local disk caching is functional for CIFS files opened as read-only.

**dir\_mode** If CIFS Unix extensions are not supported by the server this overrides the default mode for directory inodes.

**port** attempt to contact the server on this tcp port, before trying the usual ports (port 445, then 139).

**iocharset** Codepage used to convert local path names to and from Unicode. Unicode is used by default for network path names if the server supports it. If `iocharset` is not specified then the `nls_default` specified during the local client kernel build will be used. If server does not support Unicode, this parameter is unused.

**rsize** default read size (usually 16K). The client currently can not use `rsize` larger than `CIFSMaxBufSize`. `CIFSMaxBufSize` defaults to 16K and may be changed (from 8K to the maximum `kmalloc` size allowed by your kernel) at module install time for `cifs.ko`. Setting `CIFSMaxBufSize` to a very large value will cause `cifs` to use more memory and may reduce performance in some cases. To use `rsize` greater than 127K (the original `cifs` protocol maximum) also requires that the server support a new Unix Capability flag (for very large read) which some newer servers (e.g. Samba 3.0.26 or later) do. `rsize` can be set from a minimum of 2048 to a maximum of 130048 (127K or `CIFSMaxBufSize`, whichever is smaller)

**wsize** default write size (default 57344) maximum wsize currently allowed by CIFS is 57344 (fourteen 4096 byte pages)

**actimeo=*n*** attribute cache timeout in seconds (default 1 second). After this timeout, the cifs client requests fresh attribute information from the server. This option allows to tune the attribute cache timeout to suit the workload needs. Shorter timeouts mean better the cache coherency, but increased number of calls to the server. Longer timeouts mean reduced number of calls to the server at the expense of less stricter cache coherency checks (i.e. incorrect attribute cache for a short period of time).

**rw** mount the network share read-write (note that the server may still consider the share read-only)

**ro** mount network share read-only

**version** used to distinguish different versions of the mount helper utility (not typically needed)

**sep** if first mount option (after the -o), overrides the comma as the separator between the mount parms. e.g.:

```
-o user=myname,password=mypassword,domain=mydom
```

could be passed instead with period as the separator by:

```
-o sep=.user=myname.password=mypassword.domain=mydom
```

this might be useful when comma is contained within username or password or domain. This option is less important when the cifs mount helper cifs.mount (version 1.1 or later) is used.

**nosuid** Do not allow remote executables with the suid bit program to be executed. This is only meaningful for mounts to servers such as Samba which support the CIFS Unix Extensions. If you do not trust the servers in your network (your mount targets) it is recommended that you specify this option for greater security.

**exec** Permit execution of binaries on the mount.

**noexec** Do not permit execution of binaries on the mount.

**dev** Recognize block devices on the remote mount.

**nodev** Do not recognize devices on the remote mount.

**suid** Allow remote files on this mountpoint with suid enabled to be executed (default for mounts when executed as root, nosuid is default for user mounts).

**credentials** Although ignored by the cifs kernel component, it is used by the mount helper, mount.cifs. When mount.cifs is installed it opens and reads the credential file specified in order to obtain the userid and password arguments which are passed to the cifs vfs.

**guest** Although ignored by the kernel component, the mount.cifs mount helper will not prompt the user for a password if guest is specified

on the mount options. If no password is specified a null password will be used.

**perm** Client does permission checks (vfs\_permission check of uid and gid of the file against the mode and desired operation), Note that this is in addition to the normal ACL check on the target machine done by the server software. Client permission checking is enabled by default.

**noperm** Client does not do permission checks. This can expose files on this mount to access by other users on the local client system. It is typically only needed when the server supports the CIFS Unix Extensions but the UIDs/GIDs on the client and server system do not match closely enough to allow access by the user doing the mount, but it may be useful with non CIFS Unix Extension mounts for cases in which the default mode is specified on the mount but is not to be enforced on the client (e.g. perhaps when MultiUserMount is enabled) Note that this does not affect the normal ACL check on the target machine done by the server software (of the server ACL against the user name provided at mount time).

**serverino** Use server's inode numbers instead of generating automatically incrementing inode numbers on the client. Although this will make it easier to spot hardlinked files (as they will have the same inode numbers) and inode numbers may be persistent, note that the server does not guarantee that the inode numbers are unique if multiple server side mounts are exported under a single share (since inode numbers on the servers might not be unique if multiple filesystems are mounted under the same shared higher level directory). Note that some older (e.g. pre-Windows 2000) do not support returning UniqueIDs or the CIFS Unix Extensions equivalent and for those this mount option will have no effect. Exporting cifs mounts under nfsd requires this mount option on the cifs mount. This is now the default if server supports the required network operation.

**noserverino** Client generates inode numbers (rather than using the actual one from the server). These inode numbers will vary after unmount or reboot which can confuse some applications, but not all server filesystems support unique inode numbers.

**setuids** If the CIFS Unix extensions are negotiated with the server the client will attempt to set the effective uid and gid of the local process on newly created files, directories, and devices (create, mkdir, mknod). If the CIFS Unix Extensions are not negotiated, for newly created files and directories instead of using the default uid and gid specified on the mount, cache the new file's uid and gid locally which means that the uid for the file can change when the inode is reloaded (or the user remounts the share).

**nosetuids** The client will not attempt to set the uid and gid on newly created files, directories, and devices (create, mkdir, mknod) which will result in the server setting the uid and gid to the default (usually the server uid of the user who mounted the share). Letting the server (rather than the client) set the uid and gid is the default. If

the CIFS Unix Extensions are not negotiated then the uid and gid for new files will appear to be the uid (gid) of the mounter or the uid (gid) parameter specified on the mount.

**netbiosname** When mounting to servers via port 139, specifies the RFC1001 source name to use to represent the client netbios machine name when doing the RFC1001 netbios session initialize.

**direct** Do not do inode data caching on files opened on this mount. This precludes mmaping files on this mount. In some cases with fast networks and little or no caching benefits on the client (e.g. when the application is doing large sequential reads bigger than page size without rereading the same data) this can provide better performance than the default behavior which caches reads (readahead) and writes (writebehind) through the local Linux client pagecache if oplock (caching token) is granted and held. Note that direct allows write operations larger than page size to be sent to the server.

**strictcache** Use for switching on strict cache mode. In this mode the client read from the cache all the time it has Oplock Level II, otherwise - read from the server. All written data are stored in the cache, but if the client doesn't have Exclusive Oplock, it writes the data to the server.

**rwpidforward** Forward pid of a process who opened a file to any read or write operation on that file. This prevent applications like WINE from failing on read and write if we use mandatory brlock style.

**acl** Allow setfacl and getfacl to manage posix ACLs if server supports them. (default)

**noacl** Do not allow setfacl and getfacl calls on this mount

**user\_xattr** Allow getting and setting user xattrs (those attributes whose name begins with user. or os2.) as OS/2 EAs (extended attributes) to the server. This allows support of the setfattr and getfattr utilities. (default)

**nouser\_xattr** Do not allow getfattr/setfattr to get/set/list xattrs

**mapchars** Translate six of the seven reserved characters (not backslash):

```
*?<>| :
```

to the remap range (above 0xF000), which also allows the CIFS client to recognize files created with such characters by Windows' s POSIX emulation. This can also be useful when mounting to most versions of Samba (which also forbids creating and opening files whose names contain any of these seven characters). This has no effect if the server does not support Unicode on the wire.

**nomapchars** Do not translate any of these seven characters (default).

**nocase** Request case insensitive path name matching (case sensitive is the default if the server supports it). (mount option ignorecase is identical to nocase)

**posixpaths** If CIFS Unix extensions are supported, attempt to negotiate posix path name support which allows certain characters forbidden in typical CIFS filenames, without requiring remapping. (default)

**noposixpaths** If CIFS Unix extensions are supported, do not request posix path name support (this may cause servers to reject creating-file with certain reserved characters).

**nounix** Disable the CIFS Unix Extensions for this mount (tree connection). This is rarely needed, but it may be useful in order to turn off multiple settings all at once (ie posix acls, posix locks, posix paths, symlink support and retrieving uids/gids/mode from the server) or to work around a bug in server which implement the Unix Extensions.

**nobrl** Do not send byte range lock requests to the server. This is necessary for certain applications that break with cifs style mandatory byte range locks (and most cifs servers do not yet support requesting advisory byte range locks).

**forcemandatorylock** Even if the server supports posix (advisory) byte range locking, send only mandatory lock requests. For some (presumably rare) applications, originally coded for DOS/Windows, which require Windows style mandatory byte range locking, they may be able to take advantage of this option, forcing the cifs client to only send mandatory locks even if the cifs server would support posix advisory locks. `forcemand` is accepted as a shorter form of this mount option.

**nostrictsync** If this mount option is set, when an application does an `fsync` call then the cifs client does not send an SMB Flush to the server (to force the server to write all dirty data for this file immediately to disk), although cifs still sends all dirty (cached) file data to the server and waits for the server to respond to the write. Since SMB Flush can be very slow, and some servers may be reliable enough (to risk delaying slightly flushing the data to disk on the server), turning on this option may be useful to improve performance for applications that `fsync` too much, at a small risk of server crash. If this mount option is not set, by default cifs will send an SMB flush request (and wait for a response) on every `fsync` call.

**nodfs** Disable DFS (global name space support) even if the server claims to support it. This can help work around a problem with parsing of DFS paths with Samba server versions 3.0.24 and 3.0.25.

**remount** remount the share (often used to change from `ro` to `rw` mounts or vice versa)

**cifsacl** Report mode bits (e.g. on `stat`) based on the Windows ACL for the file. (EXPERIMENTAL)

**servern** Specify the server's netbios name (RFC1001 name) to use when attempting to setup a session to the server. This is needed for mounting to some older servers (such as OS/2 or Windows 98 and Windows ME) since they do not support a default server name. A server name can be up to 15 characters long and is usually upper-cased.

**sfu** When the CIFS Unix Extensions are not negotiated, attempt to create device files and fifos in a format compatible with Services for Unix (SFU). In addition retrieve bits 10-12 of the mode via the SETFILEBITS extended attribute (as SFU does). In the future the bottom 9 bits of the mode also will be emulated using queries of the security descriptor (ACL).

**mfsymlinks** Enable support for Minshall+French symlinks (see [http://wiki.samba.org/index.php/UNIX\\_Extensions#Minshall.2BFrench\\_symlinks](http://wiki.samba.org/index.php/UNIX_Extensions#Minshall.2BFrench_symlinks)) This option is ignored when specified together with the 'sfu' option. Minshall+French symlinks are used even if the server supports the CIFS Unix Extensions.

**sign** Must use packet signing (helps avoid unwanted data modification by intermediate systems in the route). Note that signing does not work with lanman or plaintext authentication.

**seal** Must seal (encrypt) all data on this mounted share before sending on the network. Requires support for Unix Extensions. Note that this differs from the sign mount option in that it causes encryption of data sent over this mounted share but other shares mounted to the same server are unaffected.

**locallease** This option is rarely needed. Fcntl F\_SETLEASE is used by some applications such as Samba and NFSv4 server to check to see whether a file is cacheable. CIFS has no way to explicitly request a lease, but can check whether a file is cacheable (oplocked). Unfortunately, even if a file is not oplocked, it could still be cacheable (ie cifs client could grant fcntl leases if no other local processes are using the file) for cases for example such as when the server does not support oplocks and the user is sure that the only updates to the file will be from this client. Specifying this mount option will allow the cifs client to check for leases (only) locally for files which are not oplocked instead of denying leases in that case. (EXPERIMENTAL)

**sec** Security mode. Allowed values are:

**none** attempt to connection as a null user (no name)

**krb5** Use Kerberos version 5 authentication

**krb5i** Use Kerberos authentication and packet signing

**ntlm** Use NTLM password hashing (default)

**ntlmi** Use NTLM password hashing with signing (if /proc/fs/cifs/PacketSigningEnabled on or if server requires signing also can be the default)

**ntlmv2** Use NTLMv2 password hashing

**ntlmv2i** Use NTLMv2 password hashing with packet signing

**lanman** (if configured in kernel config) use older lanman hash

**hard** Retry file operations if server is not responding

**soft** Limit retries to unresponsive servers (usually only one retry) before returning an error. (default)

The mount.cifs mount helper also accepts a few mount options before -o including:

-S	take password from stdin (equivalent to setting the environment variable <code>PASSWD_FD=0</code> )
-V	print mount.cifs version
-?	display simple usage information

With most 2.6 kernel versions of modutils, the version of the cifs kernel module can be displayed via modinfo.

### 29.2.10 Misc /proc/fs/cifs Flags and Debug Info

Informational pseudo-files:

Debug-Data	Displays information about active CIFS sessions and shares, features enabled as well as the cifs.ko version.
Stats	Lists summary resource usage information as well as per share statistics.

Configuration pseudo-files:

<p>SecurityFlags</p>	<p>Flags which control security negotiation and also packet signing. Authentication (may/must) flags (e.g. for NTLM and/or NTLMv2) may be combined with the signing flags. Specifying two different password hashing mechanisms (as “must use” ) on the other hand does not make much sense. Default flags are:</p> <p>0x07007</p> <p>(NTLM, NTLMv2 and packet signing allowed). The maximum allowable flags if you want to allow mounts to servers using weaker password hashes is 0x37037 (lanman, plaintext, ntlm, ntlmv2, signing allowed). Some SecurityFlags require the corresponding menuconfig options to be enabled (lanman and plaintext require CONFIG_CIFS_WEAK_PW_HASH for example). Enabling plaintext authentication currently requires also enabling lanman authentication in the security flags because the cifs module only supports sending plaintext passwords using the older lanman dialect form of the session setup SMB. (e.g. for authentication using plain text passwords, set the SecurityFlags to 0x30030):</p> <pre> may use packet signing           0x00001 ↪                                0x00001 must use packet signing          0x01001 ↪                                0x01001 may use NTLM (most common password hash) 0x00002 ↪                                0x00002 must use NTLM                    0x02002 ↪                                0x02002 may use NTLMv2                   0x00004 ↪                                0x00004 must use NTLMv2                  0x04004 ↪                                0x04004 may use Kerberos security        0x00008 ↪                                0x00008 must use Kerberos                0x08008 ↪                                0x08008 may use lanman (weak) password hash 0x00010 ↪                                0x00010 must use lanman password hash    0x10010 ↪                                0x10010 may use plaintext passwords      0x00020 ↪                                0x00020 must use plaintext passwords     0x20020 ↪                                0x20020 (reserved for future packet encryption) 0x00040 </pre>
<p><b>29.2. Usage</b></p>	<p>↪encryption) 0x00040 <b>649</b></p>
<p>cifsFYI</p>	<p>If set to non-zero value, additional debug information will be logged to the system error log. This field contains</p>

These experimental features and tracing can be enabled by changing flags in `/proc/fs/cifs` (after the `cifs` module has been installed or built into the kernel, e.g. `insmod cifs`). To enable a feature set it to 1 e.g. to enable tracing to the kernel message log type:

```
echo 7 > /proc/fs/cifs/cifsFYI
```

`cifsFYI` functions as a bit mask. Setting it to 1 enables additional kernel logging of various informational messages. 2 enables logging of non-zero SMB return codes while 4 enables logging of requests that take longer than one second to complete (except for byte range lock requests). Setting it to 4 requires `CONFIG_CIFS_STATS2` to be set in kernel configuration (`.config`). Setting it to seven enables all three. Finally, tracing the start of `smb` requests and responses can be enabled via:

```
echo 1 > /proc/fs/cifs/traceSMB
```

Per share (per client mount) statistics are available in `/proc/fs/cifs/Stats`. Additional information is available if `CONFIG_CIFS_STATS2` is enabled in the kernel configuration (`.config`). The statistics returned include counters which represent the number of attempted and failed (ie non-zero return code from the server) SMB3 (or `cifs`) requests grouped by request type (read, write, close etc.). Also recorded is the total bytes read and bytes written to the server for that share. Note that due to client caching effects this can be less than the number of bytes read and written by the application running on the client. Statistics can be reset to zero by `echo 0 > /proc/fs/cifs/Stats` which may be useful if comparing performance of two different scenarios.

Also note that `cat /proc/fs/cifs/DebugData` will display information about the active sessions and the shares that are mounted.

Enabling Kerberos (extended security) works but requires version 1.2 or later of the helper program `cifs.upcall` to be present and to be configured in the `/etc/request-key.conf` file. The `cifs.upcall` helper program is from the Samba project (<http://www.samba.org>). NTLM and NTLMv2 and LANMAN support do not require this helper. Note that NTLMv2 security (which does not require the `cifs.upcall` helper program), instead of using Kerberos, is sufficient for some use cases.

DFS support allows transparent redirection to shares in an MS-DFS name space. In addition, DFS support for target shares which are specified as UNC names which begin with host names (rather than IP addresses) requires a user space helper (such as `cifs.upcall`) to be present in order to translate host names to ip address, and the user space helper must also be configured in the file `/etc/request-key.conf`. Samba, Windows servers and many NAS appliances support DFS as a way of constructing a global name space to ease network configuration and improve reliability.

To use `cifs` Kerberos and DFS support, the Linux `keyutils` package should be installed and something like the following lines should be added to the `/etc/request-key.conf` file:

```
create cifs.spnego * * /usr/local/sbin/cifs.upcall %k
create dns_resolver * * /usr/local/sbin/cifs.upcall %k
```

### 29.2.11 CIFS kernel module parameters

These module parameters can be specified or modified either during the time of module loading or during the runtime by using the interface:

```
/proc/module/cifs/parameters/<param>
```

i.e.:

```
echo "value" > /sys/module/cifs/parameters/<param>
```

1. enable_oplocks	Enable or disable oplocks. Oplocks are enabled by default. [Y/y/1]. To disable use any of [N/n/0].
-------------------	--

## 29.3 TODO

Version 2.14 December 21, 2018

### 29.3.1 A Partial List of Missing Features

Contributions are welcome. There are plenty of opportunities for visible, important contributions to this module. Here is a partial list of the known problems and missing features:

- a) SMB3 (and SMB3.1.1) missing optional features:
  - multichannel (started), integration with RDMA
  - directory leases (improved metadata caching), started (root dir only)
  - T10 copy offload ie “ODX” (copy chunk, and “Duplicate Extents” ioctl currently the only two server side copy mechanisms supported)
- b) improved sparse file support (fiemap and SEEK\_HOLE are implemented but additional features would be supportable by the protocol).
- c) Directory entry caching relies on a 1 second timer, rather than using Directory Leases, currently only the root file handle is cached longer
- d) quota support (needs minor kernel change since quota calls to make it to network filesystems or deviceless filesystems)
- e) Additional use cases can be optimized to use “compounding” (e.g. open/query/close and open/setinfo/close) to reduce the number of roundtrips to the server and improve performance. Various cases (stat, statfs, create, unlink, mkdir) already have been improved by using compounding but more can be done. In addition we could significantly reduce redundant opens by using deferred close (with handle caching leases) and better using reference counters on file handles.
- f) Finish inotify support so kde and gnome file list windows will autorefresh (partially complete by Asser). Needs minor kernel vfs change to support removing D\_NOTIFY on a file.

- g) Add GUI tool to configure `/proc/fs/cifs` settings and for display of the CIFS statistics (started)
- h) implement support for security and trusted categories of xattrs (requires minor protocol extension) to enable better support for SELINUX
- i) Add support for tree connect contexts (see MS-SMB2) a new SMB3.1.1 protocol feature (may be especially useful for virtualization).
- j) Create UID mapping facility so server UIDs can be mapped on a per mount or a per server basis to client UIDs or nobody if no mapping exists. Also better integration with winbind for resolving SID owners
- k) Add tools to take advantage of more smb3 specific ioctls and features (passthrough ioctl/fsctl is now implemented in cifs.ko to allow sending various SMB3 fsctls and query info and set info calls directly from user space) Add tools to make setting various non-POSIX metadata attributes easier from tools (e.g. extending what was done in smb-info tool).
- l) encrypted file support
- m) improved stats gathering tools (perhaps integration with nfsometer?) to extend and make easier to use what is currently in `/proc/fs/cifs/Stats`
- n) Add support for claims based ACLs ( “DAC” )
- o) mount helper GUI (to simplify the various configuration options on mount)
- p) Add support for witness protocol (perhaps ioctl to cifs.ko from user space tool listening on witness protocol RPC) to allow for notification of share move, server failover, and server adapter changes. And also improve other failover scenarios, e.g. when client knows multiple DFS entries point to different servers, and the server we are connected to has gone down.
- q) Allow mount.cifs to be more verbose in reporting errors with dialect or unsupported feature errors.
- r) updating cifs documentation, and user guide.
- s) Addressing bugs found by running a broader set of xfstests in standard file system xfstest suite.
- t) split cifs and smb3 support into separate modules so legacy (and less secure) CIFS dialect can be disabled in environments that don’ t need it and simplify the code.
- v) POSIX Extensions for SMB3.1.1 (started, create and mkdir support added so far).
- w) Add support for additional strong encryption types, and additional spnego authentication mechanisms (see MS-SMB2)
- x) Finish support for SMB3.1.1 compression

### 29.3.2 Known Bugs

See <http://bugzilla.samba.org> - search on product “CifsVFS” for current bug list. Also check <http://bugzilla.kernel.org> (Product = File System, Component = CIFS)

- 1) existing symbolic links (Windows reparse points) are recognized but can not be created remotely. They are implemented for Samba and those that support the CIFS Unix extensions, although earlier versions of Samba overly restrict the pathnames.
- 2) `follow_link` and `readdir` code does not follow dfs junctions but recognizes them

### 29.3.3 Misc testing to do

- 1) check out max path names and max path name components against various server types. Try nested symlinks (8 deep). Return max path name in `stat -f` information
- 2) Improve `xfstest`'s cifs/smb3 enablement and adapt `xfstests` where needed to test cifs/smb3 better
- 3) Additional performance testing and optimization using `iozone` and similar - there are some easy changes that can be done to parallelize sequential writes, and when signing is disabled to request larger read sizes (larger than negotiated size) and send larger write sizes to modern servers.
- 4) More exhaustively test against less common servers
- 5) Continue to extend the smb3 “buildbot” which does automated `xfstesting` against Windows, Samba and Azure currently - to add additional tests and to allow the buildbot to execute the tests faster. The URL for the buildbot is: <http://smb3-test-rhel-75.southcentralus.cloudapp.azure.com>
- 6) Address various coverity warnings (most are not bugs per-se, but the more warnings are addressed, the easier it is to spot real problems that static analyzers will point out in the future).

## 29.4 Changes

See <https://wiki.samba.org/index.php/LinuxCIFSKernel> for summary information (that may be easier to read than parsing the output of “`git log fs/cifs`” ) about fixes/improvements to CIFS/SMB2/SMB3 support (changes to `cifs.ko` module) by kernel version (and `cifs` internal module version).

## 29.5 Authors

### 29.5.1 Original Author

Steve French ([sfrench@samba.org](mailto:sfrench@samba.org))

The author wishes to express his appreciation and thanks to: Andrew Tridgell (Samba team) for his early suggestions about smb/cifs VFS improvements. Thanks to IBM for allowing me time and test resources to pursue this project, to Jim McDonough from IBM (and the Samba Team) for his help, to the IBM Linux JFS team for explaining many esoteric Linux filesystem features. Jeremy Allison of the Samba team has done invaluable work in adding the server side of the original CIFS Unix extensions and reviewing and implementing portions of the newer CIFS POSIX extensions into the Samba 3 file server. Thank Dave Boutcher of IBM Rochester (author of the OS/400 smb/cifs filesystem client) for proving years ago that very good smb/cifs clients could be done on Unix-like operating systems. Volker Lendecke, Andrew Tridgell, Urban Widmark, John Newbigin and others for their work on the Linux smbfs module. Thanks to the other members of the Storage Network Industry Association CIFS Technical Workgroup for their work specifying this highly complex protocol and finally thanks to the Samba team for their technical advice and encouragement.

### 29.5.2 Patch Contributors

- Zwane Mwaikambo
- Andi Kleen
- Amrut Joshi
- Shobhit Dayal
- Sergey Vlasov
- Richard Hughes
- Yury Umanets
- Mark Hamzy (for some of the early cifs IPv6 work)
- Domen Puncer
- Jesper Juhl (in particular for lots of whitespace/formatting cleanup)
- Vince Negri and Dave Stahl (for finding an important caching bug)
- Adrian Bunk (kcalloc cleanups)
- Miklos Szeredi
- Kazeon team for various fixes especially for 2.4 version.
- Asser Ferno (Change Notify support)
- Shaggy (Dave Kleikamp) for innumerable small fs suggestions and some good cleanup
- Gunter Kukuk (testing and suggestions for support of old servers)

- Igor Mammedov (DFS support)
- Jeff Layton (many, many fixes, as well as great work on the cifs Kerberos code)
- Scott Lovenberg
- Pavel Shilovsky (for great work adding SMB2 support, and various SMB3 features)
- Aurelien Aptel (for DFS SMB3 work and some key bug fixes)
- Ronnie Sahlberg (for SMB3 xattr work, bug fixes, and lots of great work on compounding)
- Shirish Pargaonkar (for many ACL patches over the years)
- Sachin Prabhu (many bug fixes, including for reconnect, copy offload and security)
- Paulo Alcantara
- Long Li (some great work on RDMA, SMB Direct)

### **29.5.3 Test case and Bug Report contributors**

Thanks to those in the community who have submitted detailed bug reports and debug of problems they have found: Jochen Dolze, David Blaine, Rene Scharfe, Martin Josefsson, Alexander Wild, Anthony Liguori, Lars Muller, Urban Widmark, Massimiliano Ferrero, Howard Owen, Olaf Kirch, Kieron Briggs, Nick Millington and others. Also special mention to the Stanford Checker (SWAT) which pointed out many minor bugs in error paths. Valuable suggestions also have come from Al Viro and Dave Miller.

And thanks to the IBM LTC and Power test teams and SuSE and Citrix and RedHat testers for finding multiple bugs during excellent stress test runs.



## **CLEARING WARN\_ONCE**

WARN\_ONCE / WARN\_ON\_ONCE / printk\_once only emit a message once.

```
echo 1 > /sys/kernel/debug/clear_warn_once
```

clears the state and allows the warnings to print once again. This can be useful after test suite runs to reproduce problems.



## CPU LOAD

Linux exports various bits of information via `/proc/stat` and `/proc/uptime` that userland tools, such as `top(1)`, use to calculate the average time system spent in a particular state, for example:

```
$ iostat
Linux 2.6.18.3-exp (linmac)    02/20/2007

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           10.01    0.00   2.92   5.44    0.00   81.63

...
```

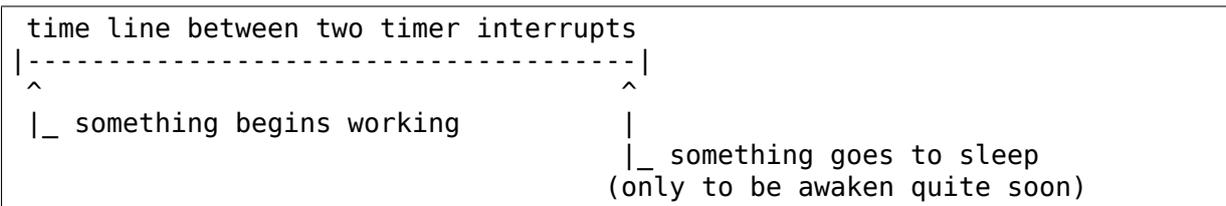
Here the system thinks that over the default sampling period the system spent 10.01% of the time doing work in user space, 2.92% in the kernel, and was overall 81.63% of the time idle.

In most cases the `/proc/stat` information reflects the reality quite closely, however due to the nature of how/when the kernel collects this data sometimes it can not be trusted at all.

So how is this information collected? Whenever timer interrupt is signalled the kernel looks what kind of task was running at this moment and increments the counter that corresponds to this tasks kind/state. The problem with this is that the system could have switched between various states multiple times between two timer interrupts yet the counter is incremented only for the last state.

### 31.1 Example

If we imagine the system with one task that periodically burns cycles in the following manner:



In the above situation the system will be 0% loaded according to the `/proc/stat` (since the timer interrupt will always happen when the system is executing the idle handler), but in reality the load is closer to 99%.

One can imagine many more situations where this behavior of the kernel will lead to quite erratic information inside /proc/stat:

```
/* gcc -o hog smallhog.c */
#include <time.h>
#include <limits.h>
#include <signal.h>
#include <sys/time.h>
#define HIST 10

static volatile sig_atomic_t stop;

static void sighandler (int signr)
{
    (void) signr;
    stop = 1;
}

static unsigned long hog (unsigned long niters)
{
    stop = 0;
    while (!stop && --niters);
    return niters;
}

int main (void)
{
    int i;
    struct itimerval it = { .it_interval = { .tv_sec = 0, .tv_usec = 1 },
                           .it_value = { .tv_sec = 0, .tv_usec = 1 } };

    sigset_t set;
    unsigned long v[HIST];
    double tmp = 0.0;
    unsigned long n;
    signal (SIGALRM, &sighandler);
    setitimer (ITIMER_REAL, &it, NULL);

    hog (ULONG_MAX);
    for (i = 0; i < HIST; ++i) v[i] = ULONG_MAX - hog (ULONG_MAX);
    for (i = 0; i < HIST; ++i) tmp += v[i];
    tmp /= HIST;
    n = tmp - (tmp / 3.0);

    sigemptyset (&set);
    sigaddset (&set, SIGALRM);

    for (;;) {
        hog (n);
        sigwait (&set, &i);
    }
    return 0;
}
```

## 31.2 References

- <http://lkml.org/lkml/2007/2/12/6>
- Documentation/filesystems/proc.rst (1.8)

## 31.3 Thanks

Con Kolivas, Pavel Machek



## **HOW CPU TOPOLOGY INFO IS EXPORTED VIA SYSFS**

Export CPU topology info via sysfs. Items (attributes) are similar to /proc/cpuinfo output of some architectures. They reside in /sys/devices/system/cpu/cpuX/topology/:

**physical\_package\_id:**

physical package id of cpuX. Typically corresponds to a physical socket number, but the actual value is architecture and platform dependent.

**die\_id:**

the CPU die ID of cpuX. Typically it is the hardware platform's identifier (rather than the kernel's). The actual value is architecture and platform dependent.

**core\_id:**

the CPU core ID of cpuX. Typically it is the hardware platform's identifier (rather than the kernel's). The actual value is architecture and platform dependent.

**book\_id:**

the book ID of cpuX. Typically it is the hardware platform's identifier (rather than the kernel's). The actual value is architecture and platform dependent.

**drawer\_id:**

the drawer ID of cpuX. Typically it is the hardware platform's identifier (rather than the kernel's). The actual value is architecture and platform dependent.

**core\_cpus:**

internal kernel map of CPUs within the same core. (deprecated name: "thread\_siblings" )

**core\_cpus\_list:**

human-readable list of CPUs within the same core. (deprecated name: "thread\_siblings\_list" );

**package\_cpus:**

internal kernel map of the CPUs sharing the same physical\_package\_id. (deprecated name: "core\_siblings" )

package\_cpus\_list:

human-readable list of CPUs sharing the same physical\_package\_id.  
(deprecated name: “core\_siblings\_list” )

die\_cpus:

internal kernel map of CPUs within the same die.

die\_cpus\_list:

human-readable list of CPUs within the same die.

book\_siblings:

internal kernel map of cpuX’s hardware threads within the same book\_id.

book\_siblings\_list:

human-readable list of cpuX’ s hardware threads within the same book\_id.

drawer\_siblings:

internal kernel map of cpuX’ s hardware threads within the same drawer\_id.

drawer\_siblings\_list:

human-readable list of cpuX’ s hardware threads within the same drawer\_id.

Architecture-neutral, drivers/base/topology.c, exports these attributes. However, the book and drawer related sysfs files will only be created if CONFIG\_SCHED\_BOOK and CONFIG\_SCHED\_DRAWER are selected, respectively.

CONFIG\_SCHED\_BOOK and CONFIG\_SCHED\_DRAWER are currently only used on s390, where they reflect the cpu and cache hierarchy.

For an architecture to support this feature, it must define some of these macros in include/asm-XXX/topology.h:

```
#define topology_physical_package_id(cpu)
#define topology_die_id(cpu)
#define topology_core_id(cpu)
#define topology_book_id(cpu)
#define topology_drawer_id(cpu)
#define topology_sibling_cpumask(cpu)
#define topology_core_cpumask(cpu)
#define topology_die_cpumask(cpu)
#define topology_book_cpumask(cpu)
#define topology_drawer_cpumask(cpu)
```

The type of \*\_id macros is int. The type of \*\_cpumask macros is (const) struct cpumask \*. The latter correspond with appropriate \*\_siblings sysfs attributes (except for topology\_sibling\_cpumask() which corresponds with thread\_siblings).

To be consistent on all architectures, include/linux/topology.h provides default definitions for any of the above macros that are not defined by include/asm-XXX/topology.h:

- 1) topology\_physical\_package\_id: -1
- 2) topology\_die\_id: -1
- 3) topology\_core\_id: 0
- 4) topology\_sibling\_cpumask: just the given CPU
- 5) topology\_core\_cpumask: just the given CPU
- 6) topology\_die\_cpumask: just the given CPU

For architectures that don't support books (CONFIG\_SCHED\_BOOK) there are no default definitions for topology\_book\_id() and topology\_book\_cpumask(). For architectures that don't support drawers (CONFIG\_SCHED\_DRAWER) there are no default definitions for topology\_drawer\_id() and topology\_drawer\_cpumask().

Additionally, CPU topology information is provided under /sys/devices/system/cpu and includes these files. The internal source for the output is in brackets ( "[ ]" ).

kernel_max:	the maximum CPU index allowed by the kernel configuration. [NR_CPUS-1]
offline:	CPUs that are not online because they have been HOT-PLUGGED off (see cpu-hotplug.txt) or exceed the limit of CPUs allowed by the kernel configuration (kernel_max above). [~cpu_online_mask + cpus >= NR_CPUS]
online:	CPUs that are online and being scheduled [cpu_online_mask]
possible:	CPUs that have been allocated resources and can be brought online if they are present. [cpu_possible_mask]
present:	CPUs that have been identified as being present in the system. [cpu_present_mask]

The format for the above output is compatible with cpulist\_parse() [see <linux/cpumask.h>]. Some examples follow.

In this example, there are 64 CPUs in the system but cpus 32-63 exceed the kernel max which is limited to 0..31 by the NR\_CPUS config option being 32. Note also that CPUs 2 and 4-31 are not online but could be brought online as they are both present and possible:

```
kernel_max: 31
  offline: 2,4-31,32-63
  online: 0-1,3
  possible: 0-31
  present: 0-31
```

In this example, the NR\_CPUS config option is 128, but the kernel was started with possible\_cpus=144. There are 4 CPUs in the system and cpu2 was manually taken offline (and is the only CPU that can be brought online.):

```
kernel_max: 127
  offline: 2,4-127,128-143
```

(continues on next page)

(continued from previous page)

```
online: 0-1,3  
possible: 0-127  
present: 0-3
```

See `cpu-hotplug.txt` for the `possible_cpus=NUM` kernel start parameter as well as more information on the various cpumasks.

## **DELL REMOTE BIOS UPDATE DRIVER (DELL\_RBU)**

### **33.1 Purpose**

Document demonstrating the use of the Dell Remote BIOS Update driver for updating BIOS images on Dell servers and desktops.

### **33.2 Scope**

This document discusses the functionality of the rbu driver only. It does not cover the support needed from applications to enable the BIOS to update itself with the image downloaded in to the memory.

### **33.3 Overview**

This driver works with Dell OpenManage or Dell Update Packages for updating the BIOS on Dell servers (starting from servers sold since 1999), desktops and notebooks (starting from those sold in 2005).

Please go to <http://support.dell.com> register and you can find info on OpenManage and Dell Update packages (DUP).

Libsmbios can also be used to update BIOS on Dell systems go to <http://linux.dell.com/libsmmbios/> for details.

Dell\_RBU driver supports BIOS update using the monolithic image and packetized image methods. In case of monolithic the driver allocates a contiguous chunk of physical pages having the BIOS image. In case of packetized the app using the driver breaks the image in to packets of fixed sizes and the driver would place each packet in contiguous physical memory. The driver also maintains a link list of packets for reading them back.

If the dell\_rbu driver is unloaded all the allocated memory is freed.

The rbu driver needs to have an application (as mentioned above) which will inform the BIOS to enable the update in the next system reboot.

The user should not unload the rbu driver after downloading the BIOS image or updating.

The driver load creates the following directories under the /sys file system:

```
/sys/class/firmware/dell_rbu/loading  
/sys/class/firmware/dell_rbu/data  
/sys/devices/platform/dell_rbu/image_type  
/sys/devices/platform/dell_rbu/data  
/sys/devices/platform/dell_rbu/packet_size
```

The driver supports two types of update mechanism; monolithic and packetized. These update mechanism depends upon the BIOS currently running on the system. Most of the Dell systems support a monolithic update where the BIOS image is copied to a single contiguous block of physical memory.

In case of packet mechanism the single memory can be broken in smaller chunks of contiguous memory and the BIOS image is scattered in these packets.

By default the driver uses monolithic memory for the update type. This can be changed to packets during the driver load time by specifying the load parameter `image_type=packet`. This can also be changed later as below:

```
echo packet > /sys/devices/platform/dell_rbu/image_type
```

In packet update mode the packet size has to be given before any packets can be downloaded. It is done as below:

```
echo XXXX > /sys/devices/platform/dell_rbu/packet_size
```

In the packet update mechanism, the user needs to create a new file having packets of data arranged back to back. It can be done as follows: The user creates packets header, gets the chunk of the BIOS image and places it next to the packetheader; now, the packetheader + BIOS image chunk added together should match the specified `packet_size`. This makes one packet, the user needs to create more such packets out of the entire BIOS image file and then arrange all these packets back to back in to one single file.

This file is then copied to `/sys/class/firmware/dell_rbu/data`. Once this file gets to the driver, the driver extracts `packet_size` data from the file and spreads it across the physical memory in contiguous `packet_size` space.

This method makes sure that all the packets get to the driver in a single operation.

In monolithic update the user simply get the BIOS image (.hdr file) and copies to the data file as is without any change to the BIOS image itself.

Do the steps below to download the BIOS image.

- 1) `echo 1 > /sys/class/firmware/dell_rbu/loading`
- 2) `cp bios_image.hdr /sys/class/firmware/dell_rbu/data`
- 3) `echo 0 > /sys/class/firmware/dell_rbu/loading`

The `/sys/class/firmware/dell_rbu/` entries will remain till the following is done.

```
echo -1 > /sys/class/firmware/dell_rbu/loading
```

Until this step is completed the driver cannot be unloaded.

Also echoing either `mono`, `packet` or `init` in to `image_type` will free up the memory allocated by the driver.

If a user by accident executes steps 1 and 3 above without executing step 2; it will make the `/sys/class/firmware/dell_rbu/` entries disappear.

The entries can be recreated by doing the following:

```
echo init > /sys/devices/platform/dell_rbu/image_type
```

---

**Note:** echoing `init` in `image_type` does not change its original value.

---

Also the driver provides `/sys/devices/platform/dell_rbu/data` readonly file to read back the image downloaded.

---

**Note:** After updating the BIOS image a user mode application needs to execute code which sends the BIOS update request to the BIOS. So on the next reboot the BIOS knows about the new image downloaded and it updates itself. Also don't unload the `rbu` driver if the image has to be updated.

---



## **DEVICE MAPPER**

### **34.1 Guidance for writing policies**

Try to keep transactionality out of it. The core is careful to avoid asking about anything that is migrating. This is a pain, but makes it easier to write the policies.

Mappings are loaded into the policy at construction time.

Every bio that is mapped by the target is referred to the policy. The policy can return a simple HIT or MISS or issue a migration.

Currently there's no way for the policy to issue background work, e.g. to start writing back dirty blocks that are going to be evicted soon.

Because we map bios, rather than requests it's easy for the policy to get fooled by many small bios. For this reason the core target issues periodic ticks to the policy. It's suggested that the policy doesn't update states (eg, hit counts) for a block more than once for each tick. The core ticks by watching bios complete, and so trying to see when the io scheduler has let the ios run.

#### **34.1.1 Overview of supplied cache replacement policies**

##### **multiqueue (mq)**

This policy is now an alias for smq (see below).

The following tunables are accepted, but have no effect:

```
'sequential_threshold <#nr_sequential_ios>'
'random_threshold <#nr_random_ios>'
'read_promote_adjustment <value>'
'write_promote_adjustment <value>'
'discard_promote_adjustment <value>'
```

### Stochastic multiqueue (smq)

This policy is the default.

The stochastic multi-queue (smq) policy addresses some of the problems with the multiqueue (mq) policy.

The smq policy (vs mq) offers the promise of less memory utilization, improved performance and increased adaptability in the face of changing workloads. smq also does not have any cumbersome tuning knobs.

Users may switch from “mq” to “smq” simply by appropriately reloading a DM table that is using the cache target. Doing so will cause all of the mq policy’s hints to be dropped. Also, performance of the cache may degrade slightly until smq recalculates the origin device’s hotspots that should be cached.

### Memory usage

The mq policy used a lot of memory; 88 bytes per cache block on a 64 bit machine.

smq uses 28bit indexes to implement its data structures rather than pointers. It avoids storing an explicit hit count for each block. It has a ‘hotspot’ queue, rather than a pre-cache, which uses a quarter of the entries (each hotspot block covers a larger area than a single cache block).

All this means smq uses ~25bytes per cache block. Still a lot of memory, but a substantial improvement nonetheless.

### Level balancing

mq placed entries in different levels of the multiqueue structures based on their hit count ( $\sim \ln(\text{hit count})$ ). This meant the bottom levels generally had the most entries, and the top ones had very few. Having unbalanced levels like this reduced the efficacy of the multiqueue.

smq does not maintain a hit count, instead it swaps hit entries with the least recently used entry from the level above. The overall ordering being a side effect of this stochastic process. With this scheme we can decide how many entries occupy each multiqueue level, resulting in better promotion/demotion decisions.

Adaptability: The mq policy maintained a hit count for each cache block. For a different block to get promoted to the cache its hit count has to exceed the lowest currently in the cache. This meant it could take a long time for the cache to adapt between varying IO patterns.

smq doesn’t maintain hit counts, so a lot of this problem just goes away. In addition it tracks performance of the hotspot queue, which is used to decide which blocks to promote. If the hotspot queue is performing badly then it starts moving entries more quickly between levels. This lets it adapt to new IO patterns very quickly.

## Performance

Testing smq shows substantially better performance than mq.

### cleaner

The cleaner writes back all dirty blocks in a cache to decommission it.

### 34.1.2 Examples

The syntax for a table is:

```
cache <metadata dev> <cache dev> <origin dev> <block size>
<#feature_args> [<feature arg>]*
<policy> <#policy_args> [<policy arg>]*
```

The syntax to send a message using the dmsetup command is:

```
dmsetup message <mapped device> 0 sequential_threshold 1024
dmsetup message <mapped device> 0 random_threshold 8
```

Using dmsetup:

```
dmsetup create blah --table "0 268435456 cache /dev/sdb /dev/sdc \
    /dev/sdd 512 0 mq 4 sequential_threshold 1024 random_threshold 8"
creates a 128GB large mapped device named 'blah' with the
sequential threshold set to 1024 and the random_threshold set to 8.
```

## 34.2 Cache

### 34.2.1 Introduction

dm-cache is a device mapper target written by Joe Thornber, Heinz Mauelshagen, and Mike Snitzer.

It aims to improve performance of a block device (eg, a spindle) by dynamically migrating some of its data to a faster, smaller device (eg, an SSD).

This device-mapper solution allows us to insert this caching at different levels of the dm stack, for instance above the data device for a thin-provisioning pool. Caching solutions that are integrated more closely with the virtual memory system should give better performance.

The target reuses the metadata library used in the thin-provisioning library.

The decision as to what data to migrate and when is left to a plug-in policy module. Several of these have been written as we experiment, and we hope other people will contribute others for specific io scenarios (eg. a vm image server).

### 34.2.2 Glossary

**Migration** Movement of the primary copy of a logical block from one device to the other.

**Promotion** Migration from slow device to fast device.

**Demotion** Migration from fast device to slow device.

The origin device always contains a copy of the logical block, which may be out of date or kept in sync with the copy on the cache device (depending on policy).

### 34.2.3 Design

#### Sub-devices

The target is constructed by passing three devices to it (along with other parameters detailed later):

1. An origin device - the big, slow one.
2. A cache device - the small, fast one.
3. A small metadata device - records which blocks are in the cache, which are dirty, and extra hints for use by the policy object. This information could be put on the cache device, but having it separate allows the volume manager to configure it differently, e.g. as a mirror for extra robustness. This metadata device may only be used by a single cache device.

#### Fixed block size

The origin is divided up into blocks of a fixed size. This block size is configurable when you first create the cache. Typically we've been using block sizes of 256KB - 1024KB. The block size must be between 64 sectors (32KB) and 2097152 sectors (1GB) and a multiple of 64 sectors (32KB).

Having a fixed block size simplifies the target a lot. But it is something of a compromise. For instance, a small part of a block may be getting hit a lot, yet the whole block will be promoted to the cache. So large block sizes are bad because they waste cache space. And small block sizes are bad because they increase the amount of metadata (both in core and on disk).

#### Cache operating modes

The cache has three operating modes: writeback, writethrough and passthrough.

If writeback, the default, is selected then a write to a block that is cached will go only to the cache and the block will be marked dirty in the metadata.

If writethrough is selected then a write to a cached block will not complete until it has hit both the origin and cache devices. Clean blocks should remain clean.

If passthrough is selected, useful when the cache contents are not known to be coherent with the origin device, then all reads are served from the origin device

(all reads miss the cache) and all writes are forwarded to the origin device; additionally, write hits cause cache block invalidates. To enable passthrough mode the cache must be clean. Passthrough mode allows a cache device to be activated without having to worry about coherency. Coherency that exists is maintained, although the cache will gradually cool as writes take place. If the coherency of the cache can later be verified, or established through use of the “invalidate\_cblocks” message, the cache device can be transitioned to writethrough or writeback mode while still warm. Otherwise, the cache contents can be discarded prior to transitioning to the desired operating mode.

A simple cleaner policy is provided, which will clean (write back) all dirty blocks in a cache. Useful for decommissioning a cache or when shrinking a cache. Shrinking the cache’s fast device requires all cache blocks, in the area of the cache being removed, to be clean. If the area being removed from the cache still contains dirty blocks the resize will fail. Care must be taken to never reduce the volume used for the cache’s fast device until the cache is clean. This is of particular importance if writeback mode is used. Writethrough and passthrough modes already maintain a clean cache. Future support to partially clean the cache, above a specified threshold, will allow for keeping the cache warm and in writeback mode during resize.

### **Migration throttling**

Migrating data between the origin and cache device uses bandwidth. The user can set a throttle to prevent more than a certain amount of migration occurring at any one time. Currently we’re not taking any account of normal io traffic going to the devices. More work needs doing here to avoid migrating during those peak io moments.

For the time being, a message “migration\_threshold <#sectors>” can be used to set the maximum number of sectors being migrated, the default being 2048 sectors (1MB).

### **Updating on-disk metadata**

On-disk metadata is committed every time a FLUSH or FUA bio is written. If no such requests are made then commits will occur every second. This means the cache behaves like a physical disk that has a volatile write cache. If power is lost you may lose some recent writes. The metadata should always be consistent in spite of any crash.

The ‘dirty’ state for a cache block changes far too frequently for us to keep updating it on the fly. So we treat it as a hint. In normal operation it will be written when the dm device is suspended. If the system crashes all cache blocks will be assumed dirty when restarted.

### Per-block policy hints

Policy plug-ins can store a chunk of data per cache block. It's up to the policy how big this chunk is, but it should be kept small. Like the dirty flags this data is lost if there's a crash so a safe fallback value should always be possible.

Policy hints affect performance, not correctness.

### Policy messaging

Policies will have different tunables, specific to each one, so we need a generic way of getting and setting these. Device-mapper messages are used. Refer to `cache-policies.txt`.

### Discard bitset resolution

We can avoid copying data during migration if we know the block has been discarded. A prime example of this is when `mkfs` discards the whole block device. We store a bitset tracking the discard state of blocks. However, we allow this bitset to have a different block size from the cache blocks. This is because we need to track the discard state for all of the origin device (compare with the dirty bitset which is just for the smaller cache device).

## 34.2.4 Target interface

### Constructor

```
cache <metadata dev> <cache dev> <origin dev> <block size>
    <#feature args> [<feature arg>]*
    <policy> <#policy args> [policy args]*
```

meta-data dev	fast device holding the persistent metadata
cache dev	fast device holding cached data blocks
origin dev	slow device holding original data blocks
block size	cache unit size in sectors
#feature args	number of feature arguments passed
feature args	writethrough or passthrough (The default is writeback.)
policy	the replacement policy to use
#policy args	an even number of arguments corresponding to key/value pairs passed to the policy
policy args	key/value pairs passed to the policy E.g. 'sequential_threshold 1024' See cache-policies.txt for details.

Optional feature arguments are:

writethrough	writethrough caching that prohibits cache block content from being different from origin block content. Without this argument, the default behaviour is to write back cache block contents later for performance reasons, so they may differ from the corresponding origin blocks.
passthrough	degraded mode useful for various cache coherency situations (e.g., rolling back snapshots of underlying storage). Reads and writes always go to the origin. If a write goes to a cached origin block, then the cache block is invalidated. To enable passthrough mode the cache must be clean.
meta-data2	use version 2 of the metadata. This stores the dirty bits in a separate btree, which improves speed of shutting down the cache.
no_discard_passing	disables passing down discards from the cache to the origin's data device.

A policy called 'default' is always registered. This is an alias for the policy we currently think is giving best all round performance.

As the default policy could vary between kernels, if you are relying on the characteristics of a specific policy, always request it by name.

### Status

```
<metadata block size> <#used metadata blocks>/<#total metadata blocks>  
<cache block size> <#used cache blocks>/<#total cache blocks>  
<#read hits> <#read misses> <#write hits> <#write misses>  
<#demotions> <#promotions> <#dirty> <#features> <features>*  
<#core args> <core args>* <policy name> <#policy args> <policy args>*  
<cache metadata mode>
```

meta- data block size	Fixed block size for each metadata block in sectors
#used meta- data blocks	Number of metadata blocks used
#to- tal meta- data blocks	Total number of metadata blocks
cache block size	Configurable block size for the cache device in sectors
#used cache blocks	Number of blocks resident in the cache
#to- tal cache blocks	Total number of cache blocks
#read hits	Number of times a READ bio has been mapped to the cache
#read misses	Number of times a READ bio has been mapped to the origin
#write hits	Number of times a WRITE bio has been mapped to the cache
#write misses	Number of times a WRITE bio has been mapped to the origin
#de- mo- tions	Number of times a block has been removed from the cache
#pro- mo- tions	Number of times a block has been moved to the cache
#dirty	Number of blocks in the cache that differ from the origin
#fea- ture args	Number of feature args to follow
fea- ture args	'writethrough' (optional)
#core args	Number of core arguments (must be even)
core args	Key/value pairs for tuning the core e.g. migration_threshold
pol- icy name	Name of the policy
#pol- icy args	Number of policy arguments to follow (must be even)
pol- icy	Key/value pairs e.g. sequential_threshold

### Messages

Policies will have different tunables, specific to each one, so we need a generic way of getting and setting these. Device-mapper messages are used. (A sysfs interface would also be possible.)

The message format is:

```
<key> <value>
```

E.g.:

```
dmsetup message my_cache 0 sequential_threshold 1024
```

Invalidation is removing an entry from the cache without writing it back. Cache blocks can be invalidated via the `invalidate_cblocks` message, which takes an arbitrary number of cblock ranges. Each cblock range's end value is "one past the end", meaning 5-10 expresses a range of values from 5 to 9. Each cblock must be expressed as a decimal value, in the future a variant message that takes cblock ranges expressed in hexadecimal may be needed to better support efficient invalidation of larger caches. The cache must be in passthrough mode when `invalidate_cblocks` is used:

```
invalidate_cblocks [<cblock>|<cblock begin>-<cblock end>]*
```

E.g.:

```
dmsetup message my_cache 0 invalidate_cblocks 2345 3456-4567 5678-6789
```

### 34.2.5 Examples

The test suite can be found here:

<https://github.com/jthornber/device-mapper-test-suite>

```
dmsetup create my_cache --table '0 41943040 cache /dev/mapper/metadata \
/dev/mapper/ssid /dev/mapper/origin 512 1 writeback default 0'
dmsetup create my_cache --table '0 41943040 cache /dev/mapper/metadata \
/dev/mapper/ssid /dev/mapper/origin 1024 1 writeback \
mq 4 sequential_threshold 1024 random_threshold 8'
```

## 34.3 dm-delay

Device-Mapper's "delay" target delays reads and/or writes and maps them to different devices.

Parameters:

```
<device> <offset> <delay> [<write_device> <write_offset> <write_delay>
                           [<flush_device> <flush_offset> <flush_delay>]]
```

With separate write parameters, the first set is only used for reads. Offsets are specified in sectors. Delays are specified in milliseconds.

### 34.3.1 Example scripts

```
#!/bin/sh
# Create device delaying rw operation for 500ms
echo "0 `blockdev --getsz $1` delay $1 0 500" | dmsetup create delayed
```

```
#!/bin/sh
# Create device delaying only write operation for 500ms and
# splitting reads and writes to different devices $1 $2
echo "0 `blockdev --getsz $1` delay $1 0 0 $2 0 500" | dmsetup create_↵
↵delayed
```

## 34.4 dm-clone

### 34.4.1 Introduction

dm-clone is a device mapper target which produces a one-to-one copy of an existing, read-only source device into a writable destination device: It presents a virtual block device which makes all data appear immediately, and redirects reads and writes accordingly.

The main use case of dm-clone is to clone a potentially remote, high-latency, read-only, archival-type block device into a writable, fast, primary-type device for fast, low-latency I/O. The cloned device is visible/mountable immediately and the copy of the source device to the destination device happens in the background, in parallel with user I/O.

For example, one could restore an application backup from a read-only copy, accessible through a network storage protocol (NBD, Fibre Channel, iSCSI, AoE, etc.), into a local SSD or NVMe device, and start using the device immediately, without waiting for the restore to complete.

When the cloning completes, the dm-clone table can be removed altogether and be replaced, e.g., by a linear table, mapping directly to the destination device.

The dm-clone target reuses the metadata library used by the thin-provisioning target.

### 34.4.2 Glossary

**Hydration** The process of filling a region of the destination device with data from the same region of the source device, i.e., copying the region from the source to the destination device.

Once a region gets hydrated we redirect all I/O regarding it to the destination device.

### 34.4.3 Design

#### Sub-devices

The target is constructed by passing three devices to it (along with other parameters detailed later):

1. A source device - the read-only device that gets cloned and source of the hydration.
2. A destination device - the destination of the hydration, which will become a clone of the source device.
3. A small metadata device - it records which regions are already valid in the destination device, i.e., which regions have already been hydrated, or have been written to directly, via user I/O.

The size of the destination device must be at least equal to the size of the source device.

#### Regions

dm-clone divides the source and destination devices in fixed sized regions. Regions are the unit of hydration, i.e., the minimum amount of data copied from the source to the destination device.

The region size is configurable when you first create the dm-clone device. The recommended region size is the same as the file system block size, which usually is 4KB. The region size must be between 8 sectors (4KB) and 2097152 sectors (1GB) and a power of two.

Reads and writes from/to hydrated regions are serviced from the destination device.

A read to a not yet hydrated region is serviced directly from the source device.

A write to a not yet hydrated region will be delayed until the corresponding region has been hydrated and the hydration of the region starts immediately.

Note that a write request with size equal to region size will skip copying of the corresponding region from the source device and overwrite the region of the destination device directly.

#### Discards

dm-clone interprets a discard request to a range that hasn't been hydrated yet as a hint to skip hydration of the regions covered by the request, i.e., it skips copying the region's data from the source to the destination device, and only updates its metadata.

If the destination device supports discards, then by default dm-clone will pass down discard requests to it.

## Background Hydration

dm-clone copies continuously from the source to the destination device, until all of the device has been copied.

Copying data from the source to the destination device uses bandwidth. The user can set a throttle to prevent more than a certain amount of copying occurring at any one time. Moreover, dm-clone takes into account user I/O traffic going to the devices and pauses the background hydration when there is I/O in-flight.

A message `hydration_threshold <#regions>` can be used to set the maximum number of regions being copied, the default being 1 region.

dm-clone employs dm-kcopyd for copying portions of the source device to the destination device. By default, we issue copy requests of size equal to the region size. A message `hydration_batch_size <#regions>` can be used to tune the size of these copy requests. Increasing the hydration batch size results in dm-clone trying to batch together contiguous regions, so we copy the data in batches of this many regions.

When the hydration of the destination device finishes, a dm event will be sent to user space.

## Updating on-disk metadata

On-disk metadata is committed every time a FLUSH or FUA bio is written. If no such requests are made then commits will occur every second. This means the dm-clone device behaves like a physical disk that has a volatile write cache. If power is lost you may lose some recent writes. The metadata should always be consistent in spite of any crash.

### 34.4.4 Target Interface

#### Constructor

```
clone <metadata dev> <destination dev> <source dev>
↳<region size>
    [<#feature args> [<feature arg>]* [<#core args> [
↳<core arg>]*]]
```

metadata dev	Fast device holding the persistent metadata
destination dev	The destination device, where the source will be cloned
source dev	Read only device containing the data that gets cloned
region size	The size of a region in sectors
#feature args	Number of feature arguments passed
feature args	no_hydration or no_discard_passthrough
#core args	An even number of arguments corresponding to key/value pairs passed to dm-clone
core args	Key/value pairs passed to dm-clone, e.g. hydration_threshold 256

Optional feature arguments are:

no_hydration	Create a dm-clone instance with background hydration disabled
no_discard_passthrough	Disable passing down discards to the destination device

Optional core arguments are:

hydration_threshold <#regions>	Maximum number of regions being copied from the source to the destination device at any one time, during background hydration.
hydration_batch_size <#regions>	During background hydration, try to batch together contiguous regions, so we copy data from the source to the destination device in batches of this many regions.

## Status

```
<metadata block size> <#used metadata blocks>/<#total_
↳metadata blocks>
<region size> <#hydrated regions>/<#total regions> <
↳#hydrating regions>
<#feature args> <feature args>* <#core args> <core args>*
<clone metadata mode>
```

meta-data block size	Fixed block size for each metadata block in sectors
#used meta-data blocks	Number of metadata blocks used
#total meta-data blocks	Total number of metadata blocks
region size	Configurable region size for the device in sectors
#hydrated regions	Number of regions that have finished hydrating
#total regions	Total number of regions to hydrate
#hydrating regions	Number of regions currently hydrating
#feature args	Number of feature arguments to follow
feature args	Feature arguments, e.g. no_hydration
#core args	Even number of core arguments to follow
core args	Key/value pairs for tuning the core, e.g. hydration_threshold 256
clone meta-data mode	ro if read-only, rw if read-write In serious cases where even a read-only mode is deemed unsafe no further I/O will be permitted and the status will just contain the string 'Fail' . If the metadata mode changes, a dm event will be sent to user space.

## Messages

**disable\_hydration** Disable the background hydration of the destination device.

**enable\_hydration** Enable the background hydration of the destination device.

**hydration\_threshold <#regions>** Set background hydration threshold.

**hydration\_batch\_size <#regions>** Set background hydration batch size.

### 34.4.5 Examples

#### Clone a device containing a file system

1. Create the dm-clone device.

```
dmsetup create clone --table "0 1048576000 clone $metadata_dev $dest_
↪dev \
    $source_dev 8 1 no_hydration"
```

2. Mount the device and trim the file system. dm-clone interprets the discards sent by the file system and it will not hydrate the unused space.

```
mount /dev/mapper/clone /mnt/cloned-fs
fstrim /mnt/cloned-fs
```

3. Enable background hydration of the destination device.

```
dmsetup message clone 0 enable_hydration
```

4. When the hydration finishes, we can replace the dm-clone table with a linear table.

```
dmsetup suspend clone
dmsetup load clone --table "0 1048576000 linear $dest_dev 0"
dmsetup resume clone
```

The metadata device is no longer needed and can be safely discarded or reused for other purposes.

### 34.4.6 Known issues

1. We redirect reads, to not-yet-hydrated regions, to the source device. If reading the source device has high latency and the user repeatedly reads from the same regions, this behaviour could degrade performance. We should use these reads as hints to hydrate the relevant regions sooner. Currently, we rely on the page cache to cache these regions, so we hopefully don't end up reading them multiple times from the source device.

2. Release in-core resources, i.e., the bitmaps tracking which regions are hydrated, after the hydration has finished.
3. During background hydration, if we fail to read the source or write to the destination device, we print an error message, but the hydration process continues indefinitely, until it succeeds. We should stop the background hydration after a number of failures and emit a dm event for user space to notice.

### 34.4.7 Why not...?

We explored the following alternatives before implementing dm-clone:

1. Use dm-cache with cache size equal to the source device and implement a new cloning policy:
  - The resulting cache device is not a one-to-one mirror of the source device and thus we cannot remove the cache device once cloning completes.
  - dm-cache writes to the source device, which violates our requirement that the source device must be treated as read-only.
  - Caching is semantically different from cloning.
2. Use dm-snapshot with a COW device equal to the source device:
  - dm-snapshot stores its metadata in the COW device, so the resulting device is not a one-to-one mirror of the source device.
  - No background copying mechanism.
  - dm-snapshot needs to commit its metadata whenever a pending exception completes, to ensure snapshot consistency. In the case of cloning, we don't need to be so strict and can rely on committing metadata every time a FLUSH or FUA bio is written, or periodically, like dm-thin and dm-cache do. This improves the performance significantly.
3. Use dm-mirror: The mirror target has a background copying/mirroring mechanism, but it writes to all mirrors, thus violating our requirement that the source device must be treated as read-only.
4. Use dm-thin's external snapshot functionality. This approach is the most promising among all alternatives, as the thinly-provisioned volume is a one-to-one mirror of the source device and handles reads and writes to unprovisioned/not-yet-cloned areas the same way as dm-clone does.

Still:

- There is no background copying mechanism, though one could be implemented.
- Most importantly, we want to support arbitrary block devices as the destination of the cloning process and not restrict ourselves to thinly-provisioned volumes. Thin-provisioning has an inherent metadata overhead, for maintaining the thin volume mappings, which significantly degrades performance.

Moreover, cloning a device shouldn't force the use of thin-provisioning. On the other hand, if we wish to use thin provisioning, we can just use a thin LV as dm-clone's destination device.

## 34.5 dm-crypt

Device-Mapper's "crypt" target provides transparent encryption of block devices using the kernel crypto API.

For a more detailed description of supported parameters see: <https://gitlab.com/cryptsetup/cryptsetup/wikis/DMCrypt>

Parameters:

```
<cipher> <key> <iv_offset> <device path> \
<offset> [<#opt_params> <opt_params>]
```

**<cipher>** Encryption cipher, encryption mode and Initial Vector (IV) generator.

The cipher specifications format is:

```
cipher[:keycount]-chainmode-ivmode[:ivopts]
```

Examples:

```
aes-cbc-essiv:sha256
aes-xts-plain64
serpent-xts-plain64
```

Cipher format also supports direct specification with kernel crypt API format (selected by capi: prefix). The IV specification is the same as for the first format type. This format is mainly used for specification of authenticated modes.

The crypto API cipher specifications format is:

```
capi:cipher_api_spec-ivmode[:ivopts]
```

Examples:

```
capi:cbc(aes)-essiv:sha256
capi:xts(aes)-plain64
```

Examples of authenticated modes:

```
capi:gcm(aes)-random
capi:authenc(hmac(sha256),xts(aes))-random
capi:rfc7539(chacha20,poly1305)-random
```

The `/proc/crypto` contains a list of currently loaded crypto modes.

**<key>** Key used for encryption. It is encoded either as a hexadecimal number or it can be passed as `<key_string>` prefixed with single colon character ( ':' ) for keys residing in kernel keyring service. You can only use key sizes that are valid for the selected cipher in combination with the selected iv mode.

Note that for some iv modes the key string can contain additional keys (for example IV seed) so the key contains more parts concatenated into a single string.

**<key\_string>** The kernel keyring key is identified by string in following format:  
<key\_size>:<key\_type>:<key\_description>.

**<key\_size>** The encryption key size in bytes. The kernel key payload size must match the value passed in <key\_size>.

**<key\_type>** Either 'logon' or 'user' kernel key type.

**<key\_description>** The kernel keyring key description crypt target should look for when loading key of <key\_type>.

**<keycount>** Multi-key compatibility mode. You can define <keycount> keys and then sectors are encrypted according to their offsets (sector 0 uses key0; sector 1 uses key1 etc.). <keycount> must be a power of two.

**<iv\_offset>** The IV offset is a sector count that is added to the sector number before creating the IV.

**<device path>** This is the device that is going to be used as backend and contains the encrypted data. You can specify it as a path like /dev/xxx or a device number <major>:<minor>.

**<offset>** Starting sector within the device where the encrypted data begins.

**<#opt\_params>** Number of optional parameters. If there are no optional parameters, the optional parameters section can be skipped or #opt\_params can be zero. Otherwise #opt\_params is the number of following arguments.

**Example of optional parameters section:** 3 allow\_discards  
same\_cpu\_crypt submit\_from\_crypt\_cpus

**allow\_discards** Block discard requests (a.k.a. TRIM) are passed through the crypt device. The default is to ignore discard requests.

WARNING: Assess the specific security risks carefully before enabling this option. For example, allowing discards on encrypted devices may lead to the leak of information about the ciphertext device (filesystem type, used space etc.) if the discarded blocks can be located easily on the device later.

**same\_cpu\_crypt** Perform encryption using the same cpu that IO was submitted on. The default is to use an unbound workqueue so that encryption work is automatically balanced between available CPUs.

**submit\_from\_crypt\_cpus** Disable offloading writes to a separate thread after encryption. There are some situations where offloading write bios from the encryption threads to a single thread degrades performance significantly. The default is to offload write bios to the same thread because it benefits CFQ to have writes submitted using the same context.

**integrity:<bytes>:<type>** The device requires additional <bytes> metadata per-sector stored in per-bio integrity structure. This metadata must be provided by underlying dm-integrity target.

The <type> can be "none" if metadata is used only for persistent IV.

For Authenticated Encryption with Additional Data (AEAD) the <type> is “aead” . An AEAD mode additionally calculates and verifies integrity for the encrypted device. The additional space is then used for storing authentication tag (and persistent IV if needed).

**sector\_size:<bytes>** Use <bytes> as the encryption unit instead of 512 bytes sectors. This option can be in range 512 - 4096 bytes and must be power of two. Virtual device will announce this size as a minimal IO and logical sector.

**iv\_large\_sectors** IV generators will use sector number counted in <sector\_size> units instead of default 512 bytes sectors.

For example, if <sector\_size> is 4096 bytes, plain64 IV for the second sector will be 8 (without flag) and 1 if iv\_large\_sectors is present. The <iv\_offset> must be multiple of <sector\_size> (in 512 bytes units) if this flag is specified.

### 34.5.1 Example scripts

LUKS (Linux Unified Key Setup) is now the preferred way to set up disk encryption with dm-crypt using the ‘cryptsetup’ utility, see <https://gitlab.com/cryptsetup/cryptsetup>

```
#!/bin/sh
# Create a crypt device using dmsetup
dmsetup create crypt1 --table "0 `blockdev --getsz $1` crypt aes-cbc-
↳essiv:sha256 babebabebabebabebabebabebabebabe 0 $1 0"
```

```
#!/bin/sh
# Create a crypt device using dmsetup when encryption key is stored in
↳keyring service
dmsetup create crypt2 --table "0 `blockdev --getsize $1` crypt aes-cbc-
↳essiv:sha256 :32:logon:my_prefix:my_key 0 $1 0"
```

```
#!/bin/sh
# Create a crypt device using cryptsetup and LUKS header with default
↳cipher
cryptsetup luksFormat $1
cryptsetup luksOpen $1 crypt1
```

## 34.6 dm-dust

This target emulates the behavior of bad sectors at arbitrary locations, and the ability to enable the emulation of the failures at an arbitrary time.

This target behaves similarly to a linear target. At a given time, the user can send a message to the target to start failing read requests on specific blocks (to emulate the behavior of a hard disk drive with bad sectors).

When the failure behavior is enabled (i.e.: when the output of “dmsetup status” displays “fail\_read\_on\_bad\_block” ), reads of blocks in the “bad block list” will fail with EIO ( “Input/output error” ).

Writes of blocks in the “bad block list will result in the following:

1. Remove the block from the “bad block list” .
2. Successfully complete the write.

This emulates the “remapped sector” behavior of a drive with bad sectors.

Normally, a drive that is encountering bad sectors will most likely encounter more bad sectors, at an unknown time or location. With dm-dust, the user can use the “addbadblock” and “removebadblock” messages to add arbitrary bad blocks at new locations, and the “enable” and “disable” messages to modulate the state of whether the configured “bad blocks” will be treated as bad, or bypassed. This allows the pre-writing of test data and metadata prior to simulating a “failure” event where bad sectors start to appear.

### 34.6.1 Table parameters

<device\_path> <offset> <blksz>

#### Mandatory parameters:

**<device\_path>:** Path to the block device.

**<offset>:** Offset to data area from start of device\_path

**<blksz>:** Block size in bytes

(minimum 512, maximum 1073741824, must be a power of 2)

### 34.6.2 Usage instructions

First, find the size (in 512-byte sectors) of the device to be used:

```
$ sudo blockdev --getsz /dev/vdb1
33552384
```

Create the dm-dust device: (For a device with a block size of 512 bytes)

```
$ sudo dmsetup create dust1 --table '0 33552384 dust /dev/vdb1 0 512'
```

(For a device with a block size of 4096 bytes)

```
$ sudo dmsetup create dust1 --table '0 33552384 dust /dev/vdb1 0 4096'
```

Check the status of the read behavior (“bypass” indicates that all I/O will be passed through to the underlying device):

```
$ sudo dmsetup status dust1
0 33552384 dust 252:17 bypass

$ sudo dd if=/dev/mapper/dust1 of=/dev/null bs=512 count=128 iflag=direct
128+0 records in
128+0 records out

$ sudo dd if=/dev/zero of=/dev/mapper/dust1 bs=512 count=128 oflag=direct
128+0 records in
128+0 records out
```

### 34.6.3 Adding and removing bad blocks

At any time (i.e.: whether the device has the “bad block” emulation enabled or disabled), bad blocks may be added or removed from the device via the “addbadblock” and “removebadblock” messages:

```
$ sudo dmsetup message dust1 0 addbadblock 60
kernel: device-mapper: dust: badblock added at block 60

$ sudo dmsetup message dust1 0 addbadblock 67
kernel: device-mapper: dust: badblock added at block 67

$ sudo dmsetup message dust1 0 addbadblock 72
kernel: device-mapper: dust: badblock added at block 72
```

These bad blocks will be stored in the “bad block list” . While the device is in “bypass” mode, reads and writes will succeed:

```
$ sudo dmsetup status dust1
0 33552384 dust 252:17 bypass
```

### 34.6.4 Enabling block read failures

To enable the “fail read on bad block” behavior, send the “enable” message:

```
$ sudo dmsetup message dust1 0 enable
kernel: device-mapper: dust: enabling read failures on bad sectors

$ sudo dmsetup status dust1
0 33552384 dust 252:17 fail_read_on_bad_block
```

With the device in “fail read on bad block” mode, attempting to read a block will encounter an “Input/output error” :

```
$ sudo dd if=/dev/mapper/dust1 of=/dev/null bs=512 count=1 skip=67
↳iflag=direct
dd: error reading '/dev/mapper/dust1': Input/output error
0+0 records in
0+0 records out
0 bytes copied, 0.00040651 s, 0.0 kB/s
```

...and writing to the bad blocks will remove the blocks from the list, therefore emulating the “remap” behavior of hard disk drives:

```
$ sudo dd if=/dev/zero of=/dev/mapper/dust1 bs=512 count=128 oflag=direct
128+0 records in
128+0 records out

kernel: device-mapper: dust: block 60 removed from badblocklist by write
kernel: device-mapper: dust: block 67 removed from badblocklist by write
kernel: device-mapper: dust: block 72 removed from badblocklist by write
kernel: device-mapper: dust: block 87 removed from badblocklist by write
```

### 34.6.5 Bad block add/remove error handling

Attempting to add a bad block that already exists in the list will result in an “Invalid argument” error, as well as a helpful message:

```
$ sudo dmsetup message dust1 0 addbadblock 88
device-mapper: message ioctl on dust1 failed: Invalid argument
kernel: device-mapper: dust: block 88 already in badblocklist
```

Attempting to remove a bad block that doesn't exist in the list will result in an “Invalid argument” error, as well as a helpful message:

```
$ sudo dmsetup message dust1 0 removebadblock 87
device-mapper: message ioctl on dust1 failed: Invalid argument
kernel: device-mapper: dust: block 87 not found in badblocklist
```

### 34.6.6 Counting the number of bad blocks in the bad block list

To count the number of bad blocks configured in the device, run the following message command:

```
$ sudo dmsetup message dust1 0 countbadblocks
```

A message will print with the number of bad blocks currently configured on the device:

```
kernel: device-mapper: dust: countbadblocks: 895 badblock(s) found
```

### 34.6.7 Querying for specific bad blocks

To find out if a specific block is in the bad block list, run the following message command:

```
$ sudo dmsetup message dust1 0 queryblock 72
```

The following message will print if the block is in the list:

```
device-mapper: dust: queryblock: block 72 found in badblocklist
```

The following message will print if the block is not in the list:

```
device-mapper: dust: queryblock: block 72 not found in badblocklist
```

The “queryblock” message command will work in both the “enabled” and “disabled” modes, allowing the verification of whether a block will be treated as “bad” without having to issue I/O to the device, or having to “enable” the bad block emulation.

### 34.6.8 Clearing the bad block list

To clear the bad block list (without needing to individually run a “removebadblock” message command for every block), run the following message command:

```
$ sudo dmsetup message dust1 0 clearbadblocks
```

After clearing the bad block list, the following message will appear:

```
kernel: device-mapper: dust: clearbadblocks: badblocks cleared
```

If there were no bad blocks to clear, the following message will appear:

```
kernel: device-mapper: dust: clearbadblocks: no badblocks found
```

### 34.6.9 Message commands list

Below is a list of the messages that can be sent to a dust device:

Operations on blocks (requires a <blknum> argument):

```
addbadblock <blknum>
queryblock <blknum>
removebadblock <blknum>
```

...where <blknum> is a block number within range of the device (corresponding to the block size of the device.)

Single argument message commands:

```
countbadblocks
clearbadblocks
disable
enable
quiet
```

### 34.6.10 Device removal

When finished, remove the device via the “dmsetup remove” command:

```
$ sudo dmsetup remove dust1
```

### 34.6.11 Quiet mode

On test runs with many bad blocks, it may be desirable to avoid excessive logging (from bad blocks added, removed, or “remapped” ). This can be done by enabling “quiet mode” via the following message:

```
$ sudo dmsetup message dust1 0 quiet
```

This will suppress log messages from add / remove / removed by write operations. Log messages from “countbadblocks” or “queryblock” message commands will still print in quiet mode.

The status of quiet mode can be seen by running “dmsetup status” :

```
$ sudo dmsetup status dust1
0 33552384 dust 252:17 fail_read_on_bad_block quiet
```

To disable quiet mode, send the “quiet” message again:

```
$ sudo dmsetup message dust1 0 quiet

$ sudo dmsetup status dust1
0 33552384 dust 252:17 fail_read_on_bad_block verbose
```

(The presence of “verbose” indicates normal logging.)

### 34.6.12 “Why not…?”

scsi\_debug has a “medium error” mode that can fail reads on one specified sector (sector 0x1234, hardcoded in the source code), but it uses RAM for the persistent storage, which drastically decreases the potential device size.

dm-flakey fails all I/O from all block locations at a specified time frequency, and not a given point in time.

When a bad sector occurs on a hard disk drive, reads to that sector are failed by the device, usually resulting in an error code of EIO ( “I/O error” ) or ENODATA ( “No data available” ). However, a write to the sector may succeed, and result in the sector becoming readable after the device controller no longer experiences errors reading the sector (or after a reallocation of the sector). However, there may be bad sectors that occur on the device in the future, in a different, unpredictable location.

This target seeks to provide a device that can exhibit the behavior of a bad sector at a known sector location, at a known time, based on a large storage device (at least tens of gigabytes, not occupying system memory).

## 34.7 dm-ebs

This target is similar to the linear target except that it emulates a smaller logical block size on a device with a larger logical block size. Its main purpose is to provide emulation of 512 byte sectors on devices that do not provide this emulation (i.e. 4K native disks).

Supported emulated logical block sizes 512, 1024, 2048 and 4096.

Underlying block size can be set to > 4K to test buffering larger units.

### 34.7.1 Table parameters

<dev path> <offset> <emulated sectors> [<underlying sectors>]

Mandatory parameters:

**<dev path>:** Full pathname to the underlying block-device, or a “major:minor” device-number.

**<offset>:** Starting sector within the device; has to be a multiple of <emulated sectors>.

**<emulated sectors>:** Number of sectors defining the logical block size to be emulated; 1, 2, 4, 8 sectors of 512 bytes supported.

Optional parameter:

**<underlying sectors>:** Number of sectors defining the logical block size of <dev path>.  $2^N$  supported, e.g. 8 = emulate 8 sectors of 512 bytes = 4KiB. If not provided, the logical block size of <dev path> will be used.

Examples:

Emulate 1 sector = 512 bytes logical block size on /dev/sda starting at offset 1024 sectors with underlying devices block size automatically set:

```
ebs /dev/sda 1024 1
```

Emulate 2 sector = 1KiB logical block size on /dev/sda starting at offset 128 sectors, enforce 2KiB underlying device block size. This presumes 2KiB logical block-size on /dev/sda or less to work:

```
ebs /dev/sda 128 2 4
```

## 34.8 dm-flakey

This target is the same as the linear target except that it exhibits unreliable behaviour periodically. It’s been found useful in simulating failing devices for testing purposes.

Starting from the time the table is loaded, the device is available for <up interval> seconds, then exhibits unreliable behaviour for <down interval> seconds, and then this cycle repeats.

Also, consider using this in combination with the dm-delay target too, which can delay reads and writes and/or send them to different underlying devices.

### 34.8.1 Table parameters

```
<dev path> <offset> <up interval> <down interval> \  
  [<num_features> [<feature arguments>]]
```

Mandatory parameters:

**<dev path>:** Full pathname to the underlying block-device, or a “major:minor” device-number.

**<offset>:** Starting sector within the device.

**<up interval>:** Number of seconds device is available.

**<down interval>:** Number of seconds device returns errors.

Optional feature parameters:

If no feature parameters are present, during the periods of unreliability, all I/O returns errors.

**drop\_writes:** All write I/O is silently ignored. Read I/O is handled correctly.

**error\_writes:** All write I/O is failed with an error signalled. Read I/O is handled correctly.

**corrupt\_bio\_byte <Nth\_byte> <direction> <value> <flags>:**

During <down interval>, replace <Nth\_byte> of the data of each matching bio with <value>.

**<Nth\_byte>:** The offset of the byte to replace. Counting starts at 1, to replace the first byte.

**<direction>:** Either ‘r’ to corrupt reads or ‘w’ to corrupt writes. ‘w’ is incompatible with drop\_writes.

**<value>:** The value (from 0-255) to write.

**<flags>:** Perform the replacement only if bio->bi\_opf has all the selected flags set.

Examples:

Replaces the 32nd byte of READ bios with the value 1:

```
corrupt_bio_byte 32 r 1 0
```

Replaces the 224th byte of REQ\_META (=32) bios with the value 0:

```
corrupt_bio_byte 224 w 0 32
```

## 34.9 Early creation of mapped devices

It is possible to configure a device-mapper device to act as the root device for your system in two ways.

The first is to build an initial ramdisk which boots to a minimal userspace which configures the device, then `pivot_root(8)` in to it.

The second is to create one or more device-mappers using the module parameter “`dm-mod.create=`” through the kernel boot command line argument.

The format is specified as a string of data separated by commas and optionally semi-colons, where:

- a comma is used to separate fields like name, uuid, flags and table (specifies one device)
- a semi-colon is used to separate devices.

So the format will look like this:

```
dm-mod.create=<name>,<uuid>,<minor>,<flags>,<table>[,<table>+][;<name>,<uuid>,<minor>,<flags>,<table>[,<table>+]]
```

Where:

```
<name>          ::= The device name.
<uuid>         ::= xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx | ""
<minor>        ::= The device minor number | ""
<flags>        ::= "ro" | "rw"
<table>        ::= <start_sector> <num_sectors> <target_type> <target_
↳args>
<target_type> ::= "verity" | "linear" | ... (see list below)
```

The `dm` line should be equivalent to the one used by the `dmsetup` tool with the `-concise` argument.

### 34.9.1 Target types

Not all target types are available as there are serious risks in allowing activation of certain DM targets without first using userspace tools to check the validity of associated metadata.

cache	constrained, userspace should verify cache device
crypt	allowed
delay	allowed
era	constrained, userspace should verify metadata device
flakey	constrained, meant for test
linear	allowed
log-writes	constrained, userspace should verify metadata device
mirror	constrained, userspace should verify main/mirror device
raid	constrained, userspace should verify metadata device
snapshot	constrained, userspace should verify src/dst device
snapshot-origin	allowed
snapshot-merge	constrained, userspace should verify src/dst device
striped	allowed
switch	constrained, userspace should verify dev path
thin	constrained, requires dm target message from userspace
thin-pool	constrained, requires dm target message from userspace
verity	allowed
writocache	constrained, userspace should verify cache device
zero	constrained, not meant for rootfs

If the target is not listed above, it is constrained by default (not tested).

### 34.9.2 Examples

An example of booting to a linear array made up of user-mode linux block devices:

```
dm-mod.create="lroot,,rw, 0 4096 linear 98:16 0, 4096 4096 linear 98:32 0
↪" root=/dev/dm-0
```

This will boot to a rw dm-linear target of 8192 sectors split across two block devices identified by their major:minor numbers. After boot, udev will rename this target to /dev/mapper/lroot (depending on the rules). No uuid was assigned.

An example of multiple device-mappers, with the dm-mod.create=" ..." contents is shown here split on multiple lines for readability:

```
dm-linear,,1,rw,
 0 32768 linear 8:1 0,
 32768 1024000 linear 8:2 0;
dm-verity,,3,ro,
 0 1638400 verity 1 /dev/sdc1 /dev/sdc2 4096 4096 204800 1 sha256
ac87db56303c9c1da433d7209b5a6ef3e4779df141200cbd7c157dcb8dd89c42
5ebfe87f7df3235b80a117ebc4078e44f55045487ad4a96581d1adb564615b51
```

Other examples (per target):



The dm-integrity target can also be used as a standalone target, in this mode it calculates and verifies the integrity tag internally. In this mode, the dm-integrity target can be used to detect silent data corruption on the disk or in the I/O path.

There's an alternate mode of operation where dm-integrity uses bitmap instead of a journal. If a bit in the bitmap is 1, the corresponding region's data and integrity tags are not synchronized - if the machine crashes, the unsynchronized regions will be recalculated. The bitmap mode is faster than the journal mode, because we don't have to write the data twice, but it is also less reliable, because if data corruption happens when the machine crashes, it may not be detected.

When loading the target for the first time, the kernel driver will format the device. But it will only format the device if the superblock contains zeroes. If the superblock is neither valid nor zeroed, the dm-integrity target can't be loaded.

To use the target for the first time:

1. overwrite the superblock with zeroes
2. load the dm-integrity target with one-sector size, the kernel driver will format the device
3. unload the dm-integrity target
4. read the "provided\_data\_sectors" value from the superblock
5. load the dm-integrity target with the the target size "provided\_data\_sectors"
6. if you want to use dm-integrity with dm-crypt, load the dm-crypt target with the size "provided\_data\_sectors"

Target arguments:

1. the underlying block device
2. the number of reserved sector at the beginning of the device - the dm-integrity won't read or write these sectors
3. the size of the integrity tag (if "-" is used, the size is taken from the internal-hash algorithm)
4. mode:

**D - direct writes (without journal)** in this mode, journaling is not used and data sectors and integrity tags are written separately. In case of crash, it is possible that the data and integrity tag doesn't match.

**J - journaled writes** data and integrity tags are written to the journal and atomicity is guaranteed. In case of crash, either both data and tag or none of them are written. The journaled mode degrades write throughput twice because the data have to be written twice.

**B - bitmap mode - data and metadata are written without any synchronization**, the driver maintains a bitmap of dirty regions where data and metadata don't match. This mode can only be used with internal hash.

**R - recovery mode - in this mode, journal is not replayed,** checksums are not checked and writes to the device are not allowed. This mode is useful for data recovery if the device cannot be activated in any of the other standard modes.

5. the number of additional arguments

Additional arguments:

**journal\_sectors:number** The size of journal, this argument is used only if formatting the device. If the device is already formatted, the value from the superblock is used.

**interleave\_sectors:number** The number of interleaved sectors. This values is rounded down to a power of two. If the device is already formatted, the value from the superblock is used.

**meta\_device:device** Don't interleave the data and metadata on on device. Use a separate device for metadata.

**buffer\_sectors:number** The number of sectors in one buffer. The value is rounded down to a power of two.

The tag area is accessed using buffers, the buffer size is configurable. The large buffer size means that the I/O size will be larger, but there could be less I/Os issued.

**journal\_watermark:number** The journal watermark in percents. When the size of the journal exceeds this watermark, the thread that flushes the journal will be started.

**commit\_time:number** Commit time in milliseconds. When this time passes, the journal is written. The journal is also written immediatelly if the FLUSH request is received.

**internal\_hash:algorithm(:key) (the key is optional)** Use internal hash or crc. When this argument is used, the dm-integrity target won't accept integrity tags from the upper target, but it will automatically generate and verify the integrity tags.

You can use a crc algorithm (such as crc32), then integrity target will protect the data against accidental corruption. You can also use a hmac algorithm (for example "hmac(sha256):0123456789abcdef" ), in this mode it will provide cryptographic authentication of the data without encryption.

When this argument is not used, the integrity tags are accepted from an upper layer target, such as dm-crypt. The upper layer target should check the validity of the integrity tags.

**recalculate** Recalculate the integrity tags automatically. It is only valid when using internal hash.

**journal\_crypt:algorithm(:key) (the key is optional)** Encrypt the journal using given algorithm to make sure that the attacker can't read the journal. You can use a block cipher here (such as "cbc(aes)" ) or a stream cipher (for example "chacha20" , "salsa20" or "ctr(aes)" ).

The journal contains history of last writes to the block device, an attacker reading the journal could see the last sector numbrers that were written. From

the sector numbers, the attacker can infer the size of files that were written. To protect against this situation, you can encrypt the journal.

**journal\_mac:algorithm(:key) (the key is optional)** Protect sector numbers in the journal from accidental or malicious modification. To protect against accidental modification, use a crc algorithm, to protect against malicious modification, use a hmac algorithm with a key.

This option is not needed when using internal-hash because in this mode, the integrity of journal entries is checked when replaying the journal. Thus, modified sector number would be detected at this stage.

**block\_size:number** The size of a data block in bytes. The larger the block size the less overhead there is for per-block integrity metadata. Supported values are 512, 1024, 2048 and 4096 bytes. If not specified the default block size is 512 bytes.

**sectors\_per\_bit:number** In the bitmap mode, this parameter specifies the number of 512-byte sectors that corresponds to one bitmap bit.

**bitmap\_flush\_interval:number** The bitmap flush interval in milliseconds. The metadata buffers are synchronized when this interval expires.

**fix\_padding** Use a smaller padding of the tag area that is more space-efficient. If this option is not present, large padding is used - that is for compatibility with older kernels.

**allow\_discards** Allow block discard requests (a.k.a. TRIM) for the integrity device. Discards are only allowed to devices using internal hash.

The journal mode (D/J), `buffer_sectors`, `journal_watermark`, `commit_time` and `allow_discards` can be changed when reloading the target (load an inactive table and swap the tables with `suspend` and `resume`). The other arguments should not be changed when reloading the target because the layout of disk data depend on them and the reloaded target would be non-functional.

Status line:

1. the number of integrity mismatches
2. provided data sectors - that is the number of sectors that the user could use
3. the current recalculating position (or ‘-’ ‘if we didn’ t recalculate)

The layout of the formatted block device:

- **reserved sectors** (they are not used by this target, they can be used for storing LUKS metadata or for other purpose), the size of the reserved area is specified in the target arguments
- **superblock (4kiB)**
  - magic string - identifies that the device was formatted
  - version
  - log2(interleave sectors)
  - integrity tag size
  - the number of journal sections

- provided data sectors - the number of sectors that this target provides (i.e. the size of the device minus the size of all metadata and padding). The user of this target should not send bios that access data beyond the “provided data sectors” limit.

- **flags**

- SB\_FLAG\_HAVE\_JOURNAL\_MAC**

- \* a flag is set if journal\_mac is used

- SB\_FLAG\_RECALCULATING**

- \* recalculating is in progress

- SB\_FLAG\_DIRTY\_BITMAP**

- \* journal area contains the bitmap of dirty blocks

- log2(sectors per block)

- a position where recalculating finished

- **journal** The journal is divided into sections, each section contains:

- metadata area (4kiB), it contains journal entries

- \* every journal entry contains:

- logical sector (specifies where the data and tag should be written)
      - last 8 bytes of data
      - integrity tag (the size is specified in the superblock)

- \* every metadata sector ends with

- mac (8-bytes), all the macs in 8 metadata sectors form a 64-byte value. It is used to store hmac of sector numbers in the journal section, to protect against a possibility that the attacker tampers with sector numbers in the journal.
      - commit id

- data area (the size is variable; it depends on how many journal entries fit into the metadata area)

- \* every sector in the data area contains:

- data (504 bytes of data, the last 8 bytes are stored in the journal entry)
      - commit id

To test if the whole journal section was written correctly, every 512-byte sector of the journal ends with 8-byte commit id. If the commit id matches on all sectors in a journal section, then it is assumed that the section was written correctly. If the commit id doesn't match, the section was written partially and it should not be replayed.

- **one or more runs of interleaved tags and data.** Each run contains:

- tag area - it contains integrity tags. There is one tag for each sector in the data area
- data area - it contains data sectors. The number of data sectors in one run must be a power of two.  $\log_2$  of this value is stored in the superblock.

### 34.11 dm-io

Dm-io provides synchronous and asynchronous I/O services. There are three types of I/O services available, and each type has a sync and an async version.

The user must set up an `io_region` structure to describe the desired location of the I/O. Each `io_region` indicates a block-device along with the starting sector and size of the region:

```
struct io_region {
    struct block_device *bdev;
    sector_t sector;
    sector_t count;
};
```

Dm-io can read from one `io_region` or write to one or more `io_regions`. Writes to multiple regions are specified by an array of `io_region` structures.

The first I/O service type takes a list of memory pages as the data buffer for the I/O, along with an offset into the first page:

```
struct page_list {
    struct page_list *next;
    struct page *page;
};

int dm_io_sync(unsigned int num_regions, struct io_region *where, int rw,
              struct page_list *pl, unsigned int offset,
              unsigned long *error_bits);
int dm_io_async(unsigned int num_regions, struct io_region *where, int rw,
               struct page_list *pl, unsigned int offset,
               io_notify_fn fn, void *context);
```

The second I/O service type takes an array of bio vectors as the data buffer for the I/O. This service can be handy if the caller has a pre-assembled bio, but wants to direct different portions of the bio to different devices:

```
int dm_io_sync_bvec(unsigned int num_regions, struct io_region *where,
                   int rw, struct bio_vec *bvec,
                   unsigned long *error_bits);
int dm_io_async_bvec(unsigned int num_regions, struct io_region *where,
                    int rw, struct bio_vec *bvec,
                    io_notify_fn fn, void *context);
```

The third I/O service type takes a pointer to a `vmalloc`'d memory buffer as the data buffer for the I/O. This service can be handy if the caller needs to do I/O to a large region but doesn't want to allocate a large number of individual memory pages:

```
int dm_io_sync_vm(unsigned int num_regions, struct io_region *where, int
↳rw,
                void *data, unsigned long *error_bits);
int dm_io_async_vm(unsigned int num_regions, struct io_region *where, int
↳rw,
                 void *data, io_notify_fn fn, void *context);
```

Callers of the asynchronous I/O services must include the name of a completion callback routine and a pointer to some context data for the I/O:

```
typedef void (*io_notify_fn)(unsigned long error, void *context);
```

The “error” parameter in this callback, as well as the `*error` parameter in all of the synchronous versions, is a bitset (instead of a simple error value). In the case of an write-I/O to multiple regions, this bitset allows dm-io to indicate success or failure on each individual region.

Before using any of the dm-io services, the user should call `dm_io_get()` and specify the number of pages they expect to perform I/O on concurrently. Dm-io will attempt to resize its mempool to make sure enough pages are always available in order to avoid unnecessary waiting while performing I/O.

When the user is finished using the dm-io services, they should call `dm_io_put()` and specify the same number of pages that were given on the `dm_io_get()` call.

## 34.12 Device-Mapper Logging

The device-mapper logging code is used by some of the device-mapper RAID targets to track regions of the disk that are not consistent. A region (or portion of the address space) of the disk may be inconsistent because a RAID stripe is currently being operated on or a machine died while the region was being altered. In the case of mirrors, a region would be considered dirty/inconsistent while you are writing to it because the writes need to be replicated for all the legs of the mirror and may not reach the legs at the same time. Once all writes are complete, the region is considered clean again.

There is a generic logging interface that the device-mapper RAID implementations use to perform logging operations (see `dm_dirty_log_type` in `include/linux/dm-dirty-log.h`). Various different logging implementations are available and provide different capabilities. The list includes:

Type	Files
disk	drivers/md/dm-log.c
core	drivers/md/dm-log.c
userspace	drivers/md/dm-log-userspace* include/linux/dm-log-userspace.h

### 34.12.1 The “disk” log type

This log implementation commits the log state to disk. This way, the logging state survives reboots/crashes.

### 34.12.2 The “core” log type

This log implementation keeps the log state in memory. The log state will not survive a reboot or crash, but there may be a small boost in performance. This method can also be used if no storage device is available for storing log state.

### 34.12.3 The “userspace” log type

This log type simply provides a way to export the log API to userspace, so log implementations can be done there. This is done by forwarding most logging requests to userspace, where a daemon receives and processes the request.

The structure used for communication between kernel and userspace are located in `include/linux/dm-log-userspace.h`. Due to the frequency, diversity, and 2-way communication nature of the exchanges between kernel and userspace, ‘connector’ is used as the interface for communication.

There are currently two userspace log implementations that leverage this framework - “clustered-disk” and “clustered-core”. These implementations provide a cluster-coherent log for shared-storage. Device-mapper mirroring can be used in a shared-storage environment when the cluster log implementations are employed.

## 34.13 dm-queue-length

`dm-queue-length` is a path selector module for device-mapper targets, which selects a path with the least number of in-flight I/Os. The path selector name is ‘queue-length’.

Table parameters for each path: [`<repeat_count>`]

<code>&lt;repeat_count&gt;</code> : The number of I/Os to dispatch using the selected path before switching to the next path. If not given, internal default is used. To check the default value, see the activated table.
---

Status for each path: `<status>` `<fail-count>` `<in-flight>`

<code>&lt;status&gt;</code> : 'A' if the path is active, 'F' if the path is failed. <code>&lt;fail-count&gt;</code> : The number of path failures. <code>&lt;in-flight&gt;</code> : The number of in-flight I/Os on the path.
---

### 34.13.1 Algorithm

dm-queue-length increments/decrements ‘in-flight’ when an I/O is dispatched/completed respectively. dm-queue-length selects a path with the minimum ‘in-flight’ .

### 34.13.2 Examples

In case that 2 paths (sda and sdb) are used with repeat\_count == 128.

```
# echo "0 10 multipath 0 0 1 1 queue-length 0 2 1 8:0 128 8:16 128" \
  dmsetup create test
#
# dmsetup table
test: 0 10 multipath 0 0 1 1 queue-length 0 2 1 8:0 128 8:16 128
#
# dmsetup status
test: 0 10 multipath 2 0 0 0 1 1 E 0 2 1 8:0 A 0 0 8:16 A 0 0
```

## 34.14 dm-raid

The device-mapper RAID (dm-raid) target provides a bridge from DM to MD. It allows the MD RAID drivers to be accessed using a device-mapper interface.

### 34.14.1 Mapping Table Interface

The target is named “raid” and it accepts the following parameters:

```
<raid_type> <#raid_params> <raid_params> \
  <#raid_devs> <metadata_dev0> <dev0> [... <metadata_devN> <devN>]
```

<raid\_type>:

raid0	RAID0 striping (no resilience)
raid1	RAID1 mirroring
raid4	RAID4 with dedicated last parity disk
raid5_n	RAID5 with dedicated last parity disk supporting takeover Same as raid4 <ul style="list-style-type: none"> <li>• Transitory layout</li> </ul>
raid5_la	RAID5 left asymmetric <ul style="list-style-type: none"> <li>• rotating parity 0 with data continuation</li> </ul>
raid5_ra	RAID5 right asymmetric <ul style="list-style-type: none"> <li>• rotating parity N with data continuation</li> </ul>
raid5_ls	RAID5 left symmetric <ul style="list-style-type: none"> <li>• rotating parity 0 with data restart</li> </ul>
raid5_rs	RAID5 right symmetric <ul style="list-style-type: none"> <li>• rotating parity N with data restart</li> </ul>
raid6_zr	RAID6 zero restart <ul style="list-style-type: none"> <li>• rotating parity zero (left-to-right) with data restart</li> </ul>
raid6_nr	RAID6 N restart <ul style="list-style-type: none"> <li>• rotating parity N (right-to-left) with data restart</li> </ul>
raid6_nc	RAID6 N continue <ul style="list-style-type: none"> <li>• rotating parity N (right-to-left) with data continuation</li> </ul>
raid6_n_6	RAID6 with dedicate parity disks <ul style="list-style-type: none"> <li>• parity and Q-syndrome on the last 2 disks; layout for takeover from/to raid4/raid5_n</li> </ul>
raid6_la_6	Same as “raid_la” plus dedicated last Q-syndrome disk <ul style="list-style-type: none"> <li>• layout for takeover from raid5_la from/to raid6</li> </ul>
raid6_ra_6	Same as “raid5_ra” dedicated last Q-syndrome disk <ul style="list-style-type: none"> <li>• layout for takeover from raid5_ra from/to raid6</li> </ul>
710	<b>Chapter 34. Device Mapper</b>
raid6_ls_6	Same as “raid5_ls” dedicated last Q-syndrome disk <ul style="list-style-type: none"> <li>• layout for takeover from</li> </ul>

Reference: Chapter 4 of [http://www.snia.org/sites/default/files/SNIA\\_DDF\\_Technical\\_Position\\_v2.0.pdf](http://www.snia.org/sites/default/files/SNIA_DDF_Technical_Position_v2.0.pdf)

<#raid\_params>: The number of parameters that follow.

<raid\_params> consists of

**Mandatory parameters:**

**<chunk\_size>:** Chunk size in sectors. This parameter is often known as “stripe size” . It is the only mandatory parameter and is placed first.

**followed by optional parameters (in any order):**

**[sync|nosync]** Force or prevent RAID initialization.

**[rebuild <idx>]** Rebuild drive number ‘idx’ (first drive is 0).

**[daemon\_sleep <ms>]** Interval between runs of the bitmap daemon that clear bits. A longer interval means less bitmap I/O but resyncing after a failure is likely to take longer.

**[min\_recovery\_rate <kB/sec/disk>]** Throttle RAID initialization

**[max\_recovery\_rate <kB/sec/disk>]** Throttle RAID initialization

**[write\_mostly <idx>]** Mark drive index ‘idx’ write-mostly.

**[max\_write\_behind <sectors>]** See ‘-write-behind=’ (man mdadm)

**[stripe\_cache <sectors>]** Stripe cache size (RAID 4/5/6 only)

**[region\_size <sectors>]** The region\_size multiplied by the number of regions is the logical size of the array. The bitmap records the device synchronisation state for each region.

**[raid10\_copies <# copies>], [raid10\_format <near|far|offset>]**

These two options are used to alter the default layout of a RAID10 configuration. The number of copies is can be specified, but the default is 2. There are also three variations to how the copies are laid down - the default is “near” . Near copies are what most people think of with respect to mirroring. If these options are left unspecified, or ‘raid10\_copies 2’ and/or ‘raid10\_format near’ are given, then the layouts for 2, 3 and 4 devices are:

2 drives	3 drives	4 drives
A1 A1	A1 A1 A2	A1 A1 A2 A2
A2 A2	A2 A3 A3	A3 A3 A4 A4
A3 A3	A4 A4 A5	A5 A5 A6 A6
A4 A4	A5 A6 A6	A7 A7 A8 A8

The 2-device layout is equivalent 2-way RAID1. The 4-device layout is what a traditional RAID10 would look like. The 3-device

layout is what might be called a ‘RAID1E - Integrated Adjacent Stripe Mirroring’ .

If ‘raid10\_copies 2’ and ‘raid10\_format far’ , then the layouts for 2, 3 and 4 devices are:

2 drives	3 drives	4 drives
A1 A2	A1 A2 A3	A1 A2 A3 A4
A3 A4	A4 A5 A6	A5 A6 A7 A8
A5 A6	A7 A8 A9	A9 A10 A11 A12
A2 A1	A3 A1 A2	A2 A1 A4 A3
A4 A3	A6 A4 A5	A6 A5 A8 A7
A6 A5	A9 A7 A8	A10 A9 A12 A11

If ‘raid10\_copies 2’ and ‘raid10\_format offset’ , then the layouts for 2, 3 and 4 devices are:

2 drives	3 drives	4 drives
A1 A2	A1 A2 A3	A1 A2 A3 A4
A2 A1	A3 A1 A2	A2 A1 A4 A3
A3 A4	A4 A5 A6	A5 A6 A7 A8
A4 A3	A6 A4 A5	A6 A5 A8 A7
A5 A6	A7 A8 A9	A9 A10 A11 A12
A6 A5	A9 A7 A8	A10 A9 A12 A11

Here we see layouts closely akin to ‘RAID1E - Integrated Offset Stripe Mirroring’ .

**[delta\_disks <N>]** The delta\_disks option value (-251 < N < +251) triggers device removal (negative value) or device addition (positive value) to any reshape supporting raid levels 4/5/6 and 10. RAID levels 4/5/6 allow for addition of devices (metadata and data device tuple), raid10\_near and raid10\_offset only allow for device addition. raid10\_far does not support any reshaping at all. A minimum of devices have to be kept to enforce resilience, which is 3 devices for raid4/5 and 4 devices for raid6.

**[data\_offset <sectors>]** This option value defines the offset into each data device where the data starts. This is used to provide out-of-place reshaping space to avoid writing over data while changing the layout of stripes, hence an interruption/crash may happen at any time without the risk of losing data. E.g. when adding devices to an existing raid set during forward reshaping, the out-of-place space will be allocated at the beginning of each raid device. The kernel raid4/5/6/10 MD personalities supporting such device addition will read the data from the existing first stripes (those with smaller number of stripes) starting at data\_offset to fill up a new stripe with the larger number of

stripes, calculate the redundancy blocks (CRC/Q-syndrome) and write that new stripe to offset 0. Same will be applied to all N-1 other new stripes. This out-of-place scheme is used to change the RAID type (i.e. the allocation algorithm) as well, e.g. changing from raid5\_ls to raid5\_n.

**[journal\_dev <dev>]** This option adds a journal device to raid4/5/6 raid sets and uses it to close the ‘write hole’ caused by the non-atomic updates to the component devices which can cause data loss during recovery. The journal device is used as writethrough thus causing writes to be throttled versus non-journalled raid4/5/6 sets. Takeover/reshape is not possible with a raid4/5/6 journal device; it has to be deconfigured before requesting these.

**[journal\_mode <mode>]** This option sets the caching mode on journalled raid4/5/6 raid sets (see ‘journal\_dev <dev>’ above) to ‘writethrough’ or ‘writeback’. If ‘writeback’ is selected the journal device has to be resilient and must not suffer from the ‘write hole’ problem itself (e.g. use raid1 or raid10) to avoid a single point of failure.

**<#raid\_devs>: The number of devices composing the array.** Each device consists of two entries. The first is the device containing the metadata (if any); the second is the one containing the data. A Maximum of 64 metadata/data device entries are supported up to target version 1.8.0. 1.9.0 supports up to 253 which is enforced by the used MD kernel runtime.

If a drive has failed or is missing at creation time, a ‘-’ can be given for both the metadata and data drives for a given position.

### 34.14.2 Example Tables

```
# RAID4 - 4 data drives, 1 parity (no metadata devices)
# No metadata devices specified to hold superblock/bitmap info
# Chunk size of 1MiB
# (Lines separated for easy reading)

0 1960893648 raid \
   raid4 1 2048 \
   5 - 8:17 - 8:33 - 8:49 - 8:65 - 8:81

# RAID4 - 4 data drives, 1 parity (with metadata devices)
# Chunk size of 1MiB, force RAID initialization,
#   min recovery rate at 20 kiB/sec/disk

0 1960893648 raid \
   raid4 4 2048 sync min_recovery_rate 20 \
   5 8:17 8:18 8:33 8:34 8:49 8:50 8:65 8:66 8:81 8:82
```

### 34.14.3 Status Output

'dmsetup table' displays the table used to construct the mapping. The optional parameters are always printed in the order listed above with "sync" or "nosync" always output ahead of the other arguments, regardless of the order used when originally loading the table. Arguments that can be repeated are ordered by value.

'dmsetup status' yields information on the state and health of the array. The output is as follows (normally a single line, but expanded here for clarity):

```
1: <s> <l> raid \  
2:   <raid_type> <#devices> <health_chars> \  
3:   <sync_ratio> <sync_action> <mismatch_cnt>
```

Line 1 is the standard output produced by device-mapper.

Line 2 & 3 are produced by the raid target and are best explained by example:

```
0 1960893648 raid raid4 5 AAAAA 2/490221568 init 0
```

Here we can see the RAID type is raid4, there are 5 devices - all of which are 'A' live, and the array is 2/490221568 complete with its initial recovery. Here is a fuller description of the individual fields:

<raid_type>	Same as the <raid_type> used to create the array.
<health_chars>	One char for each device, indicating: <ul style="list-style-type: none"> <li>• ‘A’ = alive and in-sync</li> <li>• ‘a’ = alive but not in-sync</li> <li>• ‘D’ = dead/failed.</li> </ul>
<sync_ratio>	The ratio indicating how much of the array has undergone the process described by ‘sync_action’. If the ‘sync_action’ is “check” or “repair”, then the process of “resync” or “recover” can be considered complete.
<sync_action>	One of the following possible states: <p><b>idle</b></p> <ul style="list-style-type: none"> <li>• No synchronization action is being performed.</li> </ul> <p><b>frozen</b></p> <ul style="list-style-type: none"> <li>• The current action has been halted.</li> </ul> <p><b>resync</b></p> <ul style="list-style-type: none"> <li>• Array is undergoing its initial synchronization or is resynchronizing after an unclean shutdown (possibly aided by a bitmap).</li> </ul> <p><b>recover</b></p> <ul style="list-style-type: none"> <li>• A device in the array is being rebuilt or replaced.</li> </ul> <p><b>check</b></p> <ul style="list-style-type: none"> <li>• A user-initiated full check of the array is being performed. All blocks are read and checked for consistency. The number of discrepancies found are recorded in &lt;mismatch_cnt&gt;. No changes are made to the array by this action.</li> </ul> <p><b>repair</b></p> <ul style="list-style-type: none"> <li>• The same as “check”, but discrepancies are corrected.</li> </ul> <p><b>reshape</b></p> <ul style="list-style-type: none"> <li>• The array is undergoing a reshape.</li> </ul>

### 34.14.4 Message Interface

The dm-raid target will accept certain actions through the ‘message’ interface. ( ‘man dmsetup’ for more information on the message interface.) These actions include:

“idle”	Halt the current sync action.
“frozen”	Freeze the current sync action.
“resync”	Initiate/continue a resync.
“recover”	Initiate/continue a recover process.
“check”	Initiate a check (i.e. a “scrub” ) of the array.
“repair”	Initiate a repair of the array.

### 34.14.5 Discard Support

The implementation of discard support among hardware vendors varies. When a block is discarded, some storage devices will return zeroes when the block is read. These devices set the ‘discard\_zeroes\_data’ attribute. Other devices will return random data. Confusingly, some devices that advertise ‘discard\_zeroes\_data’ will not reliably return zeroes when discarded blocks are read! Since RAID 4/5/6 uses blocks from a number of devices to calculate parity blocks and (for performance reasons) relies on ‘discard\_zeroes\_data’ being reliable, it is important that the devices be consistent. Blocks may be discarded in the middle of a RAID 4/5/6 stripe and if subsequent read results are not consistent, the parity blocks may be calculated differently at any time; making the parity blocks useless for redundancy. It is important to understand how your hardware behaves with discards if you are going to enable discards with RAID 4/5/6.

Since the behavior of storage devices is unreliable in this respect, even when reporting ‘discard\_zeroes\_data’ , by default RAID 4/5/6 discard support is disabled - this ensures data integrity at the expense of losing some performance.

Storage devices that properly support ‘discard\_zeroes\_data’ are increasingly whitelisted in the kernel and can thus be trusted.

For trusted devices, the following dm-raid module parameter can be set to safely enable discard support for RAID 4/5/6:

`‘devices_handle_discards_safely’`

### 34.14.6 Version History

1.0.0	Initial version. Support for RAID 4/5/6
1.1.0	Added support for RAID 1
1.2.0	Handle creation of arrays that contain failed devices.
1.3.0	Added support for RAID 10
1.3.1	Allow device replacement/rebuild for RAID 10
1.3.2	Fix/improve redundancy checking for RAID10
1.4.0	Non-functional change. Removes arg from mapping function.
1.4.1	RAID10 fix redundancy validation checks (commit 55ebbb5).
1.4.2	Add RAID10 "far" and "offset" algorithm support.

(continues on next page)

(continued from previous page)

- 1.5.0 Add message interface to allow manipulation of the sync\_action. New status (STATUSTYPE\_INFO) fields: sync\_action and mismatch\_cnt.
- 1.5.1 Add ability to restore transiently failed devices on resume.
- 1.5.2 'mismatch\_cnt' is zero unless [last\_]sync\_action is "check".
- 1.6.0 Add discard support (and devices\_handle\_discard\_safely module ↪param).
- 1.7.0 Add support for MD RAID0 mappings.
- 1.8.0 Explicitly check for compatible flags in the superblock metadata and reject to start the raid set if any are set by a newer target version, thus avoiding data corruption on a raid set with a reshape in progress.
- 1.9.0 Add support for RAID level takeover/reshape/region size and set size reduction.
- 1.9.1 Fix activation of existing RAID 4/10 mapped devices
- 1.9.2 Don't emit '- -' on the status table line in case the constructor fails reading a superblock. Correctly emit 'maj:min1 maj:min2' and 'D' on the status line. If '- -' is passed into the constructor, ↪emit '- -' on the table line and '-' as the status line health character.
- 1.10.0 Add support for raid4/5/6 journal device
- 1.10.1 Fix data corruption on reshape request
- 1.11.0 Fix table line argument order (wrong raid10\_copies/raid10\_format sequence)
- 1.11.1 Add raid4/5/6 journal write-back support via journal\_mode option
- 1.12.1 Fix for MD deadlock between mddev\_suspend() and md\_write\_start(), ↪available
- 1.13.0 Fix dev\_health status at end of "recover" (was 'a', now 'A')
- 1.13.1 Fix deadlock caused by early md\_stop\_writes(). Also fix size an state races.
- 1.13.2 Fix raid redundancy validation and avoid keeping raid set frozen
- 1.14.0 Fix reshape race on small devices. Fix stripe adding reshape deadlock/potential data corruption. Update superblock when specific devices are requested via rebuild. Fix RAID leg rebuild errors.
- 1.15.0 Fix size extensions not being synchronized in case of new MD bitmap pages allocated; also fix those not occurring after previous ↪reductions
- 1.15.1 Fix argument count and arguments for rebuild/write\_mostly/journal ↪(dev|mode) on the status line.

## 34.15 dm-service-time

dm-service-time is a path selector module for device-mapper targets, which selects a path with the shortest estimated service time for the incoming I/O.

The service time for each path is estimated by dividing the total size of in-flight I/Os on a path with the performance value of the path. The performance value is a relative throughput value among all paths in a path-group, and it can be specified as a table argument.

The path selector name is 'service-time'.

Table parameters for each path:

**[<repeat\_count> [<relative\_throughput>]]**

**<repeat\_count>**: The number of I/Os to dispatch using the selected path before switching to the next path. If not given, internal default is used. To check the default value, see the activated table.

**<relative\_throughput>**: The relative throughput value of the path among all paths in the path-group. The valid range is 0-100. If not given, minimum value '1' is used. If '0' is given, the path isn't selected while other paths having a positive value are available.

Status for each path:

**<status> <fail-count> <in-flight-size> <relative\_throughput>**

**<status>**: 'A' if the path is active, 'F' if the path is failed.

**<fail-count>**: The number of path failures.

**<in-flight-size>**: The size of in-flight I/Os on the path.

**<relative\_throughput>**: The relative throughput value of the path among all paths in the path-group.

### 34.15.1 Algorithm

dm-service-time adds the I/O size to 'in-flight-size' when the I/O is dispatched and subtracts when completed. Basically, dm-service-time selects a path having minimum service time which is calculated by:

$$('in\_flight\_size' + 'size\_of\_incoming\_io') / 'relative\_throughput'$$

However, some optimizations below are used to reduce the calculation as much as possible.

1. If the paths have the same 'relative\_throughput', skip the division and just compare the 'in-flight-size'.
2. If the paths have the same 'in-flight-size', skip the division and just compare the 'relative\_throughput'.
3. If some paths have non-zero 'relative\_throughput' and others have zero 'relative\_throughput', ignore those paths with zero 'relative\_throughput'.

If such optimizations can't be applied, calculate service time, and compare service time. If calculated service time is equal, the path having maximum 'relative\_throughput' may be better. So compare 'relative\_throughput' then.

### 34.15.2 Examples

In case that 2 paths (sda and sdb) are used with `repeat_count == 128` and sda has an average throughput 1GB/s and sdb has 4GB/s, 'relative\_throughput' value may be '1' for sda and '4' for sdb:

```
# echo "0 10 multipath 0 0 1 1 service-time 0 2 2 8:0 128 1 8:16 128 4" \
dmsetup create test
#
# dmsetup table
test: 0 10 multipath 0 0 1 1 service-time 0 2 2 8:0 128 1 8:16 128 4
#
# dmsetup status
test: 0 10 multipath 2 0 0 0 1 1 E 0 2 2 8:0 A 0 0 1 8:16 A 0 0 4
```

Or '2' for sda and '8' for sdb would be also true:

```
# echo "0 10 multipath 0 0 1 1 service-time 0 2 2 8:0 128 2 8:16 128 8" \
dmsetup create test
#
# dmsetup table
test: 0 10 multipath 0 0 1 1 service-time 0 2 2 8:0 128 2 8:16 128 8
#
# dmsetup status
test: 0 10 multipath 2 0 0 0 1 1 E 0 2 2 8:0 A 0 0 2 8:16 A 0 0 8
```

## 34.16 device-mapper uevent

The device-mapper uevent code adds the capability to device-mapper to create and send kobject uevents (uevents). Previously device-mapper events were only available through the `ioctl` interface. The advantage of the uevents interface is the event contains environment attributes providing increased context for the event avoiding the need to query the state of the device-mapper device after the event is received.

There are two functions currently for device-mapper events. The first function listed creates the event and the second function sends the event(s):

```
void dm_path_uevent(enum dm_uevent_type event_type, struct dm_target *ti,
                   const char *path, unsigned nr_valid_paths)

void dm_send_uevents(struct list_head *events, struct kobject *kobj)
```

The variables added to the uevent environment are:

### 34.16.1 Variable Name: **DM\_TARGET**

**Uevent Action(s)** KOBJ\_CHANGE

**Type** string

**Description**

**Value** Name of device-mapper target that generated the event.

### 34.16.2 Variable Name: **DM\_ACTION**

**Uevent Action(s)** KOBJ\_CHANGE

**Type** string

**Description**

**Value** Device-mapper specific action that caused the uevent action.  
PATH\_FAILED - A path has failed; PATH\_REINSTATED - A path has been reinstated.

### 34.16.3 Variable Name: **DM\_SEQNUM**

**Uevent Action(s)** KOBJ\_CHANGE

**Type** unsigned integer

**Description** A sequence number for this specific device-mapper device.

**Value** Valid unsigned integer range.

### 34.16.4 Variable Name: **DM\_PATH**

**Uevent Action(s)** KOBJ\_CHANGE

**Type** string

**Description** Major and minor number of the path device pertaining to this event.

**Value** Path name in the form of “Major:Minor”

### 34.16.5 Variable Name: **DM\_NR\_VALID\_PATHS**

**Uevent Action(s)** KOBJ\_CHANGE

**Type** unsigned integer

**Description**

**Value** Valid unsigned integer range.

### 34.16.6 Variable Name: DM\_NAME

**Uevent Action(s)** KOBJ\_CHANGE

**Type** string

**Description** Name of the device-mapper device.

**Value** Name

### 34.16.7 Variable Name: DM\_UUID

**Uevent Action(s)** KOBJ\_CHANGE

**Type** string

**Description** UUID of the device-mapper device.

**Value** UUID. (Empty string if there isn't one.)

An example of the uevents generated as captured by udevmonitor is shown below

1.) Path failure:

```
UEVENT[1192521009.711215] change@/block/dm-3
ACTION=change
DEVPATH=/block/dm-3
SUBSYSTEM=block
DM_TARGET=multipath
DM_ACTION=PATH_FAILED
DM_SEQNUM=1
DM_PATH=8:32
DM_NR_VALID_PATHS=0
DM_NAME=mpath2
DM_UUID=mpath-35333333000002328
MINOR=3
MAJOR=253
SEQNUM=1130
```

2.) Path reinstate:

```
UEVENT[1192521132.989927] change@/block/dm-3
ACTION=change
DEVPATH=/block/dm-3
SUBSYSTEM=block
DM_TARGET=multipath
DM_ACTION=PATH_REINSTATED
DM_SEQNUM=2
DM_PATH=8:32
DM_NR_VALID_PATHS=1
DM_NAME=mpath2
DM_UUID=mpath-35333333000002328
MINOR=3
MAJOR=253
SEQNUM=1131
```

### 34.17 dm-zoned

The dm-zoned device mapper target exposes a zoned block device (ZBC and ZAC compliant devices) as a regular block device without any write pattern constraints. In effect, it implements a drive-managed zoned block device which hides from the user (a file system or an application doing raw block device accesses) the sequential write constraints of host-managed zoned block devices and can mitigate the potential device-side performance degradation due to excessive random writes on host-aware zoned block devices.

For a more detailed description of the zoned block device models and their constraints see (for SCSI devices):

[http://www.t10.org/drafts.htm#ZBC\\_Family](http://www.t10.org/drafts.htm#ZBC_Family)

and (for ATA devices):

[http://www.t13.org/Documents/UploadedDocuments/docs2015/di537r05-Zoned\\_Device\\_ATA\\_Command\\_Set\\_ZAC.pdf](http://www.t13.org/Documents/UploadedDocuments/docs2015/di537r05-Zoned_Device_ATA_Command_Set_ZAC.pdf)

The dm-zoned implementation is simple and minimizes system overhead (CPU and memory usage as well as storage capacity loss). For a 10TB host-managed disk with 256 MB zones, dm-zoned memory usage per disk instance is at most 4.5 MB and as little as 5 zones will be used internally for storing metadata and performing reclaim operations.

dm-zoned target devices are formatted and checked using the dmzadm utility available at:

<https://github.com/hgst/dm-zoned-tools>

#### 34.17.1 Algorithm

dm-zoned implements an on-disk buffering scheme to handle non-sequential write accesses to the sequential zones of a zoned block device. Conventional zones are used for caching as well as for storing internal metadata. It can also use a regular block device together with the zoned block device; in that case the regular block device will be split logically in zones with the same size as the zoned block device. These zones will be placed in front of the zones from the zoned block device and will be handled just like conventional zones.

The zones of the device(s) are separated into 2 types:

- 1) Metadata zones: these are conventional zones used to store metadata. Metadata zones are not reported as useable capacity to the user.
- 2) Data zones: all remaining zones, the vast majority of which will be sequential zones used exclusively to store user data. The conventional zones of the device may be used also for buffering user random writes. Data in these zones may be directly mapped to the conventional zone, but later moved to a sequential zone so that the conventional zone can be reused for buffering incoming random writes.

dm-zoned exposes a logical device with a sector size of 4096 bytes, irrespective of the physical sector size of the backend zoned block device being used. This allows reducing the amount of metadata needed to manage valid blocks (blocks written).

The on-disk metadata format is as follows:

- 1) The first block of the first conventional zone found contains the super block which describes the on disk amount and position of metadata blocks.
- 2) Following the super block, a set of blocks is used to describe the mapping of the logical device blocks. The mapping is done per chunk of blocks, with the chunk size equal to the zoned block device size. The mapping table is indexed by chunk number and each mapping entry indicates the zone number of the device storing the chunk of data. Each mapping entry may also indicate if the zone number of a conventional zone used to buffer random modification to the data zone.
- 3) A set of blocks used to store bitmaps indicating the validity of blocks in the data zones follows the mapping table. A valid block is defined as a block that was written and not discarded. For a buffered data chunk, a block is always valid only in the data zone mapping the chunk or in the buffer zone of the chunk.

For a logical chunk mapped to a conventional zone, all write operations are processed by directly writing to the zone. If the mapping zone is a sequential zone, the write operation is processed directly only if the write offset within the logical chunk is equal to the write pointer offset within of the sequential data zone (i.e. the write operation is aligned on the zone write pointer). Otherwise, write operations are processed indirectly using a buffer zone. In that case, an unused conventional zone is allocated and assigned to the chunk being accessed. Writing a block to the buffer zone of a chunk will automatically invalidate the same block in the sequential zone mapping the chunk. If all blocks of the sequential zone become invalid, the zone is freed and the chunk buffer zone becomes the primary zone mapping the chunk, resulting in native random write performance similar to a regular block device.

Read operations are processed according to the block validity information provided by the bitmaps. Valid blocks are read either from the sequential zone mapping a chunk, or if the chunk is buffered, from the buffer zone assigned. If the accessed chunk has no mapping, or the accessed blocks are invalid, the read buffer is zeroed and the read operation terminated.

After some time, the limited number of conventional zones available may be exhausted (all used to map chunks or buffer sequential zones) and unaligned writes to unbuffered chunks become impossible. To avoid this situation, a reclaim process regularly scans used conventional zones and tries to reclaim the least recently used zones by copying the valid blocks of the buffer zone to a free sequential zone. Once the copy completes, the chunk mapping is updated to point to the sequential zone and the buffer zone freed for reuse.

### **34.17.2 Metadata Protection**

To protect metadata against corruption in case of sudden power loss or system crash, 2 sets of metadata zones are used. One set, the primary set, is used as the main metadata region, while the secondary set is used as a staging area. Modified metadata is first written to the secondary set and validated by updating the super block in the secondary set, a generation counter is used to indicate that this set contains the newest metadata. Once this operation completes, in place of metadata block updates can be done in the primary metadata set. This ensures that one of the set is always consistent (all modifications committed or none at all).

Flush operations are used as a commit point. Upon reception of a flush request, metadata modification activity is temporarily blocked (for both incoming BIO processing and reclaim process) and all dirty metadata blocks are staged and updated. Normal operation is then resumed. Flushing metadata thus only temporarily delays write and discard requests. Read requests can be processed concurrently while metadata flush is being executed.

If a regular device is used in conjunction with the zoned block device, a third set of metadata (without the zone bitmaps) is written to the start of the zoned block device. This metadata has a generation counter of '0' and will never be updated during normal operation; it just serves for identification purposes. The first and second copy of the metadata are located at the start of the regular block device.

### 34.17.3 Usage

A zoned block device must first be formatted using the `dmzadm` tool. This will analyze the device zone configuration, determine where to place the metadata sets on the device and initialize the metadata sets.

Ex:

```
dmzadm --format /dev/sdxx
```

If two drives are to be used, both devices must be specified, with the regular block device as the first device.

Ex:

```
dmzadm --format /dev/sdxx /dev/sdyy
```

Formatted device(s) can be started with the `dmzadm` utility, too.:

Ex:

```
dmzadm --start /dev/sdxx /dev/sdyy
```

Information about the internal layout and current usage of the zones can be obtained with the 'status' callback from `dmsetup`:

Ex:

```
dmsetup status /dev/dm-X
```

will return a line

```
0 <size> zoned <nr_zones> zones <nr_unmap_rnd>/<nr_rnd> random
<nr_unmap_seq>/<nr_seq> sequential
```

where `<nr_zones>` is the total number of zones, `<nr_unmap_rnd>` is the number of unmapped (ie free) random zones, `<nr_rnd>` the total number of zones, `<nr_unmap_seq>` the number of unmapped sequential zones, and `<nr_seq>` the total number of sequential zones.

Normally the reclaim process will be started once there are less than 50 percent free random zones. In order to start the reclaim process manually even before reaching this threshold the 'dmsetup message' function can be used:

Ex:

```
dmsetup message /dev/dm-X 0 reclaim
```

will start the reclaim process and random zones will be moved to sequential zones.

## 34.18 dm-era

### 34.18.1 Introduction

dm-era is a target that behaves similar to the linear target. In addition it keeps track of which blocks were written within a user defined period of time called an 'era'. Each era target instance maintains the current era as a monotonically increasing 32-bit counter.

Use cases include tracking changed blocks for backup software, and partially invalidating the contents of a cache to restore cache coherency after rolling back a vendor snapshot.

### 34.18.2 Constructor

```
era <metadata dev> <origin dev> <block size>
```

metadata dev	fast device holding the persistent metadata
origin dev	device holding data blocks that may change
block size	block size of origin data device, granularity that is tracked by the target

### 34.18.3 Messages

None of the dm messages take any arguments.

#### checkpoint

Possibly move to a new era. You shouldn't assume the era has incremented. After sending this message, you should check the current era via the status line.

### take\_metadata\_snap

Create a clone of the metadata, to allow a userland process to read it.

### drop\_metadata\_snap

Drop the metadata snapshot.

### 34.18.4 Status

<metadata block size> <#used metadata blocks>/<#total metadata blocks>  
<current era> <held metadata root | ‘- ‘>

metadata block size	Fixed block size for each metadata block in sectors
#used metadata blocks	Number of metadata blocks used
#total metadata blocks	Total number of metadata blocks
current era	The current era
held meta-data root	The location, in blocks, of the metadata root that has been ‘held’ for userspace read access. ‘- ‘ indicates there is no held root

### 34.18.5 Detailed use case

The scenario of invalidating a cache when rolling back a vendor snapshot was the primary use case when developing this target:

#### Taking a vendor snapshot

- Send a checkpoint message to the era target
- Make a note of the current era in its status line
- Take vendor snapshot (the era and snapshot should be forever associated now).

#### Rolling back to an vendor snapshot

- Cache enters passthrough mode (see: dm-cache’ s docs in cache.txt)
- Rollback vendor storage
- Take metadata snapshot
- Ascertain which blocks have been written since the snapshot was taken by checking each block’ s era

- Invalidate those blocks in the caching software
- Cache returns to writeback/writethrough mode

### 34.18.6 Memory usage

The target uses a bitset to record writes in the current era. It also has a spare bitset ready for switching over to a new era. Other than that it uses a few 4k blocks for updating metadata:

```
(4 * nr_blocks) bytes + buffers
```

### 34.18.7 Resilience

Metadata is updated on disk before a write to a previously unwritten block is performed. As such dm-era should not be effected by a hard crash such as power failure.

### 34.18.8 Userland tools

Userland tools are found in the increasingly poorly named thin-provisioning-tools project:

<https://github.com/jthorner/thin-provisioning-tools>

## 34.19 kcopyd

Kcopyd provides the ability to copy a range of sectors from one block-device to one or more other block-devices, with an asynchronous completion notification. It is used by dm-snapshot and dm-mirror.

Users of kcopyd must first create a client and indicate how many memory pages to set aside for their copy jobs. This is done with a call to `kcopyd_client_create()`:

```
int kcopyd_client_create(unsigned int num_pages,
                        struct kcopyd_client **result);
```

To start a copy job, the user must set up `io_region` structures to describe the source and destinations of the copy. Each `io_region` indicates a block-device along with the starting sector and size of the region. The source of the copy is given as one `io_region` structure, and the destinations of the copy are given as an array of `io_region` structures:

```
struct io_region {
    struct block_device *bdev;
    sector_t sector;
    sector_t count;
};
```

To start the copy, the user calls `kcopyd_copy()`, passing in the client pointer, pointers to the source and destination `io_regions`, the name of a completion callback routine, and a pointer to some context data for the copy:

```
int kcopyd_copy(struct kcopyd_client *kc, struct io_region *from,
               unsigned int num_dests, struct io_region *dests,
               unsigned int flags, kcopyd_notify_fn fn, void *context);

typedef void (*kcopyd_notify_fn)(int read_err, unsigned int write_err,
                                void *context);
```

When the copy completes, `kc` will call the user's completion routine, passing back the user's context pointer. It will also indicate if a read or write error occurred during the copy.

When a user is done with all their copy jobs, they should call `kcopyd_client_destroy()` to delete the `kc` client, which will release the associated memory pages:

```
void kcopyd_client_destroy(struct kcopyd_client *kc);
```

## 34.20 dm-linear

Device-Mapper's "linear" target maps a linear range of the Device-Mapper device onto a linear range of another device. This is the basic building block of logical volume managers.

**Parameters:** `<dev path>` `<offset>`

**<dev path>:** Full pathname to the underlying block-device, or a "major:minor" device-number.

**<offset>:** Starting sector within the device.

### 34.20.1 Example scripts

```
#!/bin/sh
# Create an identity mapping for a device
echo "0 `blockdev --getsz $1` linear $1 0" | dmsetup create identity
```

```
#!/bin/sh
# Join 2 devices together
size1=`blockdev --getsz $1`
size2=`blockdev --getsz $2`
echo "0 $size1 linear $1 0
      $size1 $size2 linear $2 0" | dmsetup create joined
```

```
#!/usr/bin/perl -w
# Split a device into 4M chunks and then join them together in reverse
↳ order.

my $name = "reverse";
```

(continues on next page)

(continued from previous page)

```
my $extent_size = 4 * 1024 * 2;
my $dev = $ARGV[0];
my $table = "";
my $count = 0;

if (!defined($dev)) {
    die("Please specify a device.\n");
}

my $dev_size = `blockdev --getsz $dev`;
my $extents = int($dev_size / $extent_size) -
    (($dev_size % $extent_size) ? 1 : 0);

while ($extents > 0) {
    my $this_start = $count * $extent_size;
    $extents--;
    $count++;
    my $this_offset = $extents * $extent_size;

    $table .= "$this_start $extent_size linear $dev $this_offset\n";
}

`echo \"\$table\" | dmsetup create $name`;
```

## 34.21 dm-log-writes

This target takes 2 devices, one to pass all IO to normally, and one to log all of the write operations to. This is intended for file system developers wishing to verify the integrity of metadata or data as the file system is written to. There is a `log_write_entry` written for every WRITE request and the target is able to take arbitrary data from userspace to insert into the log. The data that is in the WRITE requests is copied into the log to make the replay happen exactly as it happened originally.

### 34.21.1 Log Ordering

We log things in order of completion once we are sure the write is no longer in cache. This means that normal WRITE requests are not actually logged until the next `REQ_PREFLUSH` request. This is to make it easier for userspace to replay the log in a way that correlates to what is on disk and not what is in cache, to make it easier to detect improper waiting/flushing.

This works by attaching all WRITE requests to a list once the write completes. Once we see a `REQ_PREFLUSH` request we splice this list onto the request and once the FLUSH request completes we log all of the WRITES and then the FLUSH. Only completed WRITES, at the time the `REQ_PREFLUSH` is issued, are added in order to simulate the worst case scenario with regard to power failures. Consider the following example (W means write, C means complete):

```
W1,W2,W3,C3,C2,Wflush,C1,Cflush
```

The log would show the following:

W3,W2,flush,W1...

Again this is to simulate what is actually on disk, this allows us to detect cases where a power failure at a particular point in time would create an inconsistent file system.

Any REQ\_FUA requests bypass this flushing mechanism and are logged as soon as they complete as those requests will obviously bypass the device cache.

Any REQ\_OP\_DISCARD requests are treated like WRITE requests. Otherwise we would have all the DISCARD requests, and then the WRITE requests and then the FLUSH request. Consider the following example:

WRITE block 1, DISCARD block 1, FLUSH

If we logged DISCARD when it completed, the replay would look like this:

DISCARD 1, WRITE 1, FLUSH

which isn't quite what happened and wouldn't be caught during the log replay.

### 34.21.2 Target interface

#### i) Constructor

log-writes <dev\_path> <log\_dev\_path>

dev_path	Device that all of the IO will go to normally.
log_dev_path	Device where the log entries are written to.

#### ii) Status

<#logged entries> <highest allocated sector>

#logged entries	Number of logged entries
highest allocated sector	Highest allocated sector

#### iii) Messages

mark <description>

You can use a dmsetup message to set an arbitrary mark in a log. For example say you want to fsck a file system after every write, but first you need to replay up to the mkfs to make sure we're fsck'ing something reasonable, you would do something like this:

```
mkfs.btrfs -f /dev/mapper/log
dmsetup message log 0 mark mkfs
<run test>
```

This would allow you to replay the log up to the mkfs mark and then replay from that point on doing the fsck check in the interval that you want.

Every log has a mark at the end labeled "dm-log-writes-end" .

### 34.21.3 Userspace component

There is a userspace tool that will replay the log for you in various ways. It can be found here: <https://github.com/josefbacik/log-writes>

### 34.21.4 Example usage

Say you want to test fsync on your file system. You would do something like this:

```
TABLE="0 $(blockdev --getsz /dev/sdb) log-writes /dev/sdb /dev/sdc"
dmsetup create log --table "$TABLE"
mkfs.btrfs -f /dev/mapper/log
dmsetup message log 0 mark mkfs

mount /dev/mapper/log /mnt/btrfs-test
<some test that does fsync at the end>
dmsetup message log 0 mark fsync
md5sum /mnt/btrfs-test/foo
umount /mnt/btrfs-test

dmsetup remove log
replay-log --log /dev/sdc --replay /dev/sdb --end-mark fsync
mount /dev/sdb /mnt/btrfs-test
md5sum /mnt/btrfs-test/foo
<verify md5sum's are correct>
```

Another option is to do a complicated file system operation and verify the file system is consistent during the entire operation. You could do this with:

```
TABLE="0 $(blockdev --getsz /dev/sdb) log-writes /dev/sdb /dev/sdc"
dmsetup create log --table "$TABLE"
mkfs.btrfs -f /dev/mapper/log
dmsetup message log 0 mark mkfs

mount /dev/mapper/log /mnt/btrfs-test
<fsstress to dirty the fs>
btrfs filesystem balance /mnt/btrfs-test
umount /mnt/btrfs-test
dmsetup remove log

replay-log --log /dev/sdc --replay /dev/sdb --end-mark mkfs
btrfsck /dev/sdb
replay-log --log /dev/sdc --replay /dev/sdb --start-mark mkfs \
  --fsck "btrfsck /dev/sdb" --check fua
```

And that will replay the log until it sees a FUA request, run the fsck command and if the fsck passes it will replay to the next FUA, until it is completed or the fsck command exists abnormally.

## 34.22 Persistent data

### 34.22.1 Introduction

The more-sophisticated device-mapper targets require complex metadata that is managed in kernel. In late 2010 we were seeing that various different targets were rolling their own data structures, for example:

- Mikulas Patocka' s multisnap implementation
- Heinz Mauelshagen' s thin provisioning target
- Another btree-based caching target posted to dm-devel
- Another multi-snapshot target based on a design of Daniel Phillips

Maintaining these data structures takes a lot of work, so if possible we' d like to reduce the number.

The persistent-data library is an attempt to provide a re-usable framework for people who want to store metadata in device-mapper targets. It' s currently used by the thin-provisioning target and an upcoming hierarchical storage target.

### 34.22.2 Overview

The main documentation is in the header files which can all be found under `drivers/md/persistent-data`.

#### The block manager

`dm-block-manager.[hc]`

This provides access to the data on disk in fixed sized-blocks. There is a read/write locking interface to prevent concurrent accesses, and keep data that is being used in the cache.

Clients of persistent-data are unlikely to use this directly.

#### The transaction manager

`dm-transaction-manager.[hc]`

This restricts access to blocks and enforces copy-on-write semantics. The only way you can get hold of a writable block through the transaction manager is by shadowing an existing block (ie. doing copy-on-write) or allocating a fresh one. Shadowing is elided within the same transaction so performance is reasonable. The commit method ensures that all data is flushed before it writes the superblock. On power failure your metadata will be as it was when last committed.

## The Space Maps

dm-space-map.h dm-space-map-metadata.[hc] dm-space-map-disk.[hc]

On-disk data structures that keep track of reference counts of blocks. Also acts as the allocator of new blocks. Currently two implementations: a simpler one for managing blocks on a different device (eg. thinly-provisioned data blocks); and one for managing the metadata space. The latter is complicated by the need to store its own data within the space it's managing.

## The data structures

dm-btree.[hc] dm-btree-remove.c dm-btree-spine.c dm-btree-internal.h

Currently there is only one data structure, a hierarchical btree. There are plans to add more. For example, something with an array-like interface would see a lot of use.

The btree is 'hierarchical' in that you can define it to be composed of nested btrees, and take multiple keys. For example, the thin-provisioning target uses a btree with two levels of nesting. The first maps a device id to a mapping tree, and that in turn maps a virtual block to a physical block.

Values stored in the btrees can have arbitrary size. Keys are always 64bits, although nesting allows you to use multiple keys.

## 34.23 Device-mapper snapshot support

Device-mapper allows you, without massive data copying:

- To create snapshots of any block device i.e. mountable, saved states of the block device which are also writable without interfering with the original content;
- To create device "forks" , i.e. multiple different versions of the same data stream.
- To merge a snapshot of a block device back into the snapshot's origin device.

In the first two cases, dm copies only the chunks of data that get changed and uses a separate copy-on-write (COW) block device for storage.

For snapshot merge the contents of the COW storage are merged back into the origin device.

There are three dm targets available: snapshot, snapshot-origin, and snapshot-merge.

- snapshot-origin <origin>

which will normally have one or more snapshots based on it. Reads will be mapped directly to the backing device. For each write, the original data will be saved in the <COW device> of each snapshot to keep its visible content unchanged, at least until the <COW device> fills up.

- snapshot <origin> <COW device> <persistent?> <chunksize> [<# feature args> [<arg>]\*]

A snapshot of the <origin> block device is created. Changed chunks of <chunksize> sectors will be stored on the <COW device>. Writes will only go to the <COW device>. Reads will come from the <COW device> or from <origin> for unchanged data. <COW device> will often be smaller than the origin and if it fills up the snapshot will become useless and be disabled, returning errors. So it is important to monitor the amount of free space and expand the <COW device> before it fills up.

<persistent?> is P (Persistent) or N (Not persistent - will not survive after reboot). O (Overflow) can be added as a persistent store option to allow userspace to advertise its support for seeing “Overflow” in the snapshot status. So supported store types are “P” , “PO” and “N” .

The difference between persistent and transient is with transient snapshots less metadata must be saved on disk - they can be kept in memory by the kernel.

When loading or unloading the snapshot target, the corresponding snapshot-origin or snapshot-merge target must be suspended. A failure to suspend the origin target could result in data corruption.

Optional features:

discard\_zeroes\_cow - a discard issued to the snapshot device that maps to entire chunks to will zero the corresponding exception(s) in the snapshot’ s exception store.

discard\_passthrough\_origin - a discard to the snapshot device is passed down to the snapshot-origin’ s underlying device. This doesn’ t cause copy-out to the snapshot exception store because the snapshot-origin target is bypassed.

The discard\_passthrough\_origin feature depends on the discard\_zeroes\_cow feature being enabled.

- snapshot-merge <origin> <COW device> <persistent> <chunksize> [<# feature args> [<arg>]\*]

takes the same table arguments as the snapshot target except it only works with persistent snapshots. This target assumes the role of the “snapshot-origin” target and must not be loaded if the “snapshot-origin” is still present for <origin>.

Creates a merging snapshot that takes control of the changed chunks stored in the <COW device> of an existing snapshot, through a handover procedure, and merges these chunks back into the <origin>. Once merging has started (in the background) the <origin> may be opened and the merge will continue while I/O is flowing to it. Changes to the <origin> are deferred until the merging snapshot’ s corresponding chunk(s) have been merged. Once merging has started the snapshot device, associated with the “snapshot” target, will return -EIO when accessed.

### 34.23.1 How snapshot is used by LVM2

When you create the first LVM2 snapshot of a volume, four dm devices are used:

- 1) a device containing the original mapping table of the source volume;
- 2) a device used as the <COW device>;
- 3) a “snapshot” device, combining #1 and #2, which is the visible snapshot volume;
- 4) the “original” volume (which uses the device number used by the original source volume), whose table is replaced by a “snapshot-origin” mapping from device #1.

A fixed naming scheme is used, so with the following commands:

```
lvcreate -L 1G -n base volumeGroup
lvcreate -L 100M --snapshot -n snap volumeGroup/base
```

we’ ll have this situation (with volumes in above order):

```
# dmsetup table|grep volumeGroup
volumeGroup-base-real: 0 2097152 linear 8:19 384
volumeGroup-snap-cow: 0 204800 linear 8:19 2097536
volumeGroup-snap: 0 2097152 snapshot 254:11 254:12 P 16
volumeGroup-base: 0 2097152 snapshot-origin 254:11

# ls -lL /dev/mapper/volumeGroup-*
brw----- 1 root root 254, 11 29 ago 18:15 /dev/mapper/volumeGroup-base-
↪real
brw----- 1 root root 254, 12 29 ago 18:15 /dev/mapper/volumeGroup-snap-
↪cow
brw----- 1 root root 254, 13 29 ago 18:15 /dev/mapper/volumeGroup-snap
brw----- 1 root root 254, 10 29 ago 18:14 /dev/mapper/volumeGroup-base
```

### 34.23.2 How snapshot-merge is used by LVM2

A merging snapshot assumes the role of the “snapshot-origin” while merging. As such the “snapshot-origin” is replaced with “snapshot-merge”. The “-real” device is not changed and the “-cow” device is renamed to <origin name>-cow to aid LVM2’ s cleanup of the merging snapshot after it completes. The “snapshot” that hands over its COW device to the “snapshot-merge” is deactivated (unless using `lvchange -refresh`); but if it is left active it will simply return I/O errors.

A snapshot will merge into its origin with the following command:

```
lvconvert --merge volumeGroup/snap
```

we’ ll now have this situation:

```
# dmsetup table|grep volumeGroup
volumeGroup-base-real: 0 2097152 linear 8:19 384
volumeGroup-base-cow: 0 204800 linear 8:19 2097536
```

(continues on next page)

(continued from previous page)

```

volumeGroup-base: 0 2097152 snapshot-merge 254:11 254:12 P 16

# ls -lL /dev/mapper/volumeGroup-*
brw----- 1 root root 254, 11 29 ago 18:15 /dev/mapper/volumeGroup-base-
↳ real
brw----- 1 root root 254, 12 29 ago 18:16 /dev/mapper/volumeGroup-base-
↳ cow
brw----- 1 root root 254, 10 29 ago 18:16 /dev/mapper/volumeGroup-base

```

### 34.23.3 How to determine when a merging is complete

The snapshot-merge and snapshot status lines end with:

```
<sectors_allocated>/<total_sectors> <metadata_sectors>
```

Both `<sectors_allocated>` and `<total_sectors>` include both data and metadata. During merging, the number of sectors allocated gets smaller and smaller. Merging has finished when the number of sectors holding data is zero, in other words `<sectors_allocated> == <metadata_sectors>`.

Here is a practical example (using a hybrid of lvm and dmsetup commands):

```

# lvs
LV      VG          Attr   LSize Origin  Snap%  Move Log Copy%  Convert
base    volumeGroup owi-a- 4.00g
snap    volumeGroup swi-a- 1.00g base   18.97

# dmsetup status volumeGroup-snap
0 8388608 snapshot 397896/2097152 1560
                        ^^^^ metadata sectors

# lvconvert --merge -b volumeGroup/snap
Merging of volume snap started.

# lvs volumeGroup/snap
LV      VG          Attr   LSize Origin  Snap%  Move Log Copy%  Convert
base    volumeGroup owi-a- 4.00g                17.23

# dmsetup status volumeGroup-base
0 8388608 snapshot-merge 281688/2097152 1104

# dmsetup status volumeGroup-base
0 8388608 snapshot-merge 180480/2097152 712

# dmsetup status volumeGroup-base
0 8388608 snapshot-merge 16/2097152 16

```

Merging has finished.

```

# lvs
LV      VG          Attr   LSize Origin  Snap%  Move Log Copy%  Convert
base    volumeGroup owi-a- 4.00g

```

## 34.24 DM statistics

Device Mapper supports the collection of I/O statistics on user-defined regions of a DM device. If no regions are defined no statistics are collected so there isn't any performance impact. Only bio-based DM devices are currently supported.

Each user-defined region specifies a starting sector, length and step. Individual statistics will be collected for each step-sized area within the range specified.

The I/O statistics counters for each step-sized area of a region are in the same format as `/sys/block/*/stat` or `/proc/diskstats` (see: `Documentation/admin-guide/iostats.rst`). But two extra counters (12 and 13) are provided: total time spent reading and writing. When the histogram argument is used, the 14th parameter is reported that represents the histogram of latencies. All these counters may be accessed by sending the `@stats_print` message to the appropriate DM device via `dmsetup`.

The reported times are in milliseconds and the granularity depends on the kernel ticks. When the option `precise_timestamps` is used, the reported times are in nanoseconds.

Each region has a corresponding unique identifier, which we call a `region_id`, that is assigned when the region is created. The `region_id` must be supplied when querying statistics about the region, deleting the region, etc. Unique `region_ids` enable multiple userspace programs to request and process statistics for the same DM device without stepping on each other's data.

The creation of DM statistics will allocate memory via `kmalloc` or fallback to using `vmalloc` space. At most, 1/4 of the overall system memory may be allocated by DM statistics. The admin can see how much memory is used by reading:

```
/sys/module/dm_mod/parameters/stats_current_allocated_bytes
```

### 34.24.1 Messages

**@stats\_create** `<range>` `<step>` [`<number_of_optional_arguments>`] `<optional_arg>`  
Create a new region and return the `region_id`.

**<range>**

“- “ whole device

“**<start\_sector>+<length>**” a range of `<length>` 512-byte sectors starting with `<start_sector>`.

**<step>**

“**<area\_size>**” the range is subdivided into areas each containing `<area_size>` sectors.

“/**<number\_of\_areas>**” the range is subdivided into the specified number of areas.

**<number\_of\_optional\_arguments>** The number of optional arguments

**<optional\_arguments>** The following optional arguments are supported:

**precise\_timestamps** use precise timer with nanosecond resolution instead of the “jiffies” variable. When this argument is used, the resulting times are in nanoseconds instead of milliseconds. Precise timestamps are a little bit slower to obtain than jiffies-based timestamps.

**histogram:n1,n2,n3,n4,...** collect histogram of latencies. The numbers n1, n2, etc are times that represent the boundaries of the histogram. If `precise_timestamps` is not used, the times are in milliseconds, otherwise they are in nanoseconds. For each range, the kernel will report the number of requests that completed within this range. For example, if we use “`histogram:10,20,30`”, the kernel will report four numbers a:b:c:d. a is the number of requests that took 0-10 ms to complete, b is the number of requests that took 10-20 ms to complete, c is the number of requests that took 20-30 ms to complete and d is the number of requests that took more than 30 ms to complete.

**<program\_id>** An optional parameter. A name that uniquely identifies the userspace owner of the range. This groups ranges together so that userspace programs can identify the ranges they created and ignore those created by others. The kernel returns this string back in the output of `@stats_list` message, but it doesn't use it for anything else. If we omit the number of optional arguments, program id must not be a number, otherwise it would be interpreted as the number of optional arguments.

**<aux\_data>** An optional parameter. A word that provides auxiliary data that is useful to the client program that created the range. The kernel returns this string back in the output of `@stats_list` message, but it doesn't use this value for anything.

**@stats\_delete <region\_id>** Delete the region with the specified id.

**<region\_id>** region\_id returned from `@stats_create`

**@stats\_clear <region\_id>** Clear all the counters except the in-flight i/o counters.

**<region\_id>** region\_id returned from `@stats_create`

**@stats\_list [<program\_id>]** List all regions registered with `@stats_create`.

**<program\_id>** An optional parameter. If this parameter is specified, only matching regions are returned. If it is not specified, all regions are returned.

### Output format:

**<region\_id>: <start\_sector>+<length> <step> <program\_id> <aux\_data>**  
precise\_timestamps histogram:n1,n2,n3,...

The strings “precise\_timestamps” and “histogram” are printed only if they were specified when creating the region.

**@stats\_print <region\_id> [<starting\_line> <number\_of\_lines>]**

Print counters for each step-sized area of a region.

**<region\_id>** region\_id returned from @stats\_create

**<starting\_line>** The index of the starting line in the output. If omitted, all lines are returned.

**<number\_of\_lines>** The number of lines to include in the output. If omitted, all lines are returned.

Output format for each step-sized area of a region:

**<start\_sector>+<length>** counters

The first 11 counters have the same meaning as /sys/block/\*/stat or /proc/diskstats.

Please refer to Documentation/admin-guide/iostats.rst for details.

1. the number of reads completed
2. the number of reads merged
3. the number of sectors read
4. the number of milliseconds spent reading
5. the number of writes completed
6. the number of writes merged
7. the number of sectors written
8. the number of milliseconds spent writing
9. the number of I/Os currently in progress
10. the number of milliseconds spent doing I/Os
11. the weighted number of milliseconds spent doing I/Os

Additional counters:

12. the total time spent reading in milliseconds
13. the total time spent writing in milliseconds

**@stats\_print\_clear <region\_id> [<starting\_line> <number\_of\_lines>]**

Atomically print and then clear all the counters except the in-flight i/o counters. Useful when the client consuming the statistics does not want to lose any statistics (those updated between printing and clearing).

**<region\_id>** region\_id returned from @stats\_create

**<starting\_line>** The index of the starting line in the output. If omitted, all lines are printed and then cleared.

**<number\_of\_lines>** The number of lines to process. If omitted, all lines are printed and then cleared.

**@stats\_set\_aux <region\_id> <aux\_data>** Store auxiliary data aux\_data for the specified region.

**<region\_id>** region\_id returned from @stats\_create

**<aux\_data>** The string that identifies data which is useful to the client program that created the range. The kernel returns this string back in the output of @stats\_list message, but it doesn't use this value for anything.

### 34.24.2 Examples

Subdivide the DM device 'vol' into 100 pieces and start collecting statistics on them:

```
dmsetup message vol 0 @stats_create - /100
```

Set the auxiliary data string to "foo bar baz" (the escape for each space must also be escaped, otherwise the shell will consume them):

```
dmsetup message vol 0 @stats_set_aux 0 foo\\ bar\\ baz
```

List the statistics:

```
dmsetup message vol 0 @stats_list
```

Print the statistics:

```
dmsetup message vol 0 @stats_print 0
```

Delete the statistics:

```
dmsetup message vol 0 @stats_delete 0
```

## 34.25 dm-stripe

Device-Mapper's "striped" target is used to create a striped (i.e. RAID-0) device across one or more underlying devices. Data is written in "chunks", with consecutive chunks rotating among the underlying devices. This can potentially provide improved I/O throughput by utilizing several physical devices in parallel.

**Parameters:** **<num devs> <chunk size> [<dev path> <offset>]+**

**<num devs>:** Number of underlying devices.

**<chunk size>:** Size of each chunk of data. Must be at least as large as the system's PAGE\_SIZE.

**<dev path>:** Full pathname to the underlying block-device, or a "major:minor" device-number.

**<offset>**: Starting sector within the device.

One or more underlying devices can be specified. The striped device size must be a multiple of the chunk size multiplied by the number of underlying devices.

### 34.25.1 Example scripts

```
#!/usr/bin/perl -w
# Create a striped device across any number of underlying devices. The
# device
# will be called "stripe_dev" and have a chunk-size of 128k.

my $chunk_size = 128 * 2;
my $dev_name = "stripe_dev";
my $num_devs = @ARGV;
my @devs = @ARGV;
my ($min_dev_size, $stripe_dev_size, $i);

if (!$num_devs) {
    die("Specify at least one device\n");
}

$min_dev_size = `blockdev --getsz $devs[0]`;
for ($i = 1; $i < $num_devs; $i++) {
    my $this_size = `blockdev --getsz $devs[$i]`;
    $min_dev_size = ($min_dev_size < $this_size) ?
        $min_dev_size : $this_size;
}

$stripe_dev_size = $min_dev_size * $num_devs;
$stripe_dev_size -= $stripe_dev_size % ($chunk_size * $num_devs);

$table = "0 $stripe_dev_size striped $num_devs $chunk_size";
for ($i = 0; $i < $num_devs; $i++) {
    $table .= " $devs[$i] 0";
}

`echo $table | dmsetup create $dev_name`;
```

## 34.26 dm-switch

The device-mapper switch target creates a device that supports an arbitrary mapping of fixed-size regions of I/O across a fixed set of paths. The path used for any specific region can be switched dynamically by sending the target a message.

It maps I/O to underlying block devices efficiently when there is a large number of fixed-sized address regions but there is no simple pattern that would allow for a compact representation of the mapping such as dm-stripe.

### 34.26.1 Background

Dell EqualLogic and some other iSCSI storage arrays use a distributed frameless architecture. In this architecture, the storage group consists of a number of distinct storage arrays ( “members” ) each having independent controllers, disk storage and network adapters. When a LUN is created it is spread across multiple members. The details of the spreading are hidden from initiators connected to this storage system. The storage group exposes a single target discovery portal, no matter how many members are being used. When iSCSI sessions are created, each session is connected to an eth port on a single member. Data to a LUN can be sent on any iSCSI session, and if the blocks being accessed are stored on another member the I/O will be forwarded as required. This forwarding is invisible to the initiator. The storage layout is also dynamic, and the blocks stored on disk may be moved from member to member as needed to balance the load.

This architecture simplifies the management and configuration of both the storage group and initiators. In a multipathing configuration, it is possible to set up multiple iSCSI sessions to use multiple network interfaces on both the host and target to take advantage of the increased network bandwidth. An initiator could use a simple round robin algorithm to send I/O across all paths and let the storage array members forward it as necessary, but there is a performance advantage to sending data directly to the correct member.

A device-mapper table already lets you map different regions of a device onto different targets. However in this architecture the LUN is spread with an address region size on the order of 10s of MBs, which means the resulting table could have more than a million entries and consume far too much memory.

Using this device-mapper switch target we can now build a two-layer device hierarchy:

- Upper Tier - Determine which array member the I/O should be sent to.
- Lower Tier - Load balance amongst paths to a particular member.

The lower tier consists of a single dm multipath device for each member. Each of these multipath devices contains the set of paths directly to the array member in one priority group, and leverages existing path selectors to load balance amongst these paths. We also build a non-preferred priority group containing paths to other array members for failover reasons.

The upper tier consists of a single dm-switch device. This device uses a bitmap to look up the location of the I/O and choose the appropriate lower tier device to route the I/O. By using a bitmap we are able to use 4 bits for each address range in a 16 member group (which is very large for us). This is a much denser representation than the dm table b-tree can achieve.

## Construction Parameters

**<num\_paths> <region\_size> <num\_optional\_args> [<optional\_args>...] [<dev\_pa**

**<num\_paths>** The number of paths across which to distribute the I/O.

**<region\_size>** The number of 512-byte sectors in a region. Each region can be redirected to any of the available paths.

**<num\_optional\_args>** The number of optional arguments. Currently, no optional arguments are supported and so this must be zero.

**<dev\_path>** The block device that represents a specific path to the device.

**<offset>** The offset of the start of data on the specific <dev\_path> (in units of 512-byte sectors). This number is added to the sector number when forwarding the request to the specific path. Typically it is zero.

## Messages

```
set_region_mappings      <index>:<path_nr>          [<index>]:<path_nr>
[<index>]:<path_nr>...
```

Modify the region table by specifying which regions are redirected to which paths.

**<index>** The region number (region size was specified in constructor parameters). If index is omitted, the next region (previous index + 1) is used. Expressed in hexadecimal (WITHOUT any prefix like 0x).

**<path\_nr>** The path number in the range 0 ..(<num\_paths> - 1). Expressed in hexadecimal (WITHOUT any prefix like 0x).

**R<n>,<m>** This parameter allows repetitive patterns to be loaded quickly. <n> and <m> are hexadecimal numbers. The last <n> mappings are repeated in the next <m> slots.

## Status

No status line is reported.

### Example

Assume that you have volumes `vg1/switch0` `vg1/switch1` `vg1/switch2` with the same size.

Create a switch device with 64kB region size:

```
dmsetup create switch --table "0 `blockdev --getsz /dev/vg1/switch0`
    switch 3 128 0 /dev/vg1/switch0 0 /dev/vg1/switch1 0 /dev/vg1/switch2 0
↪ "
```

Set mappings for the first 7 entries to point to devices `switch0`, `switch1`, `switch2`, `switch0`, `switch1`, `switch2`, `switch1`:

```
dmsetup message switch 0 set_region_mappings 0:0 :1 :2 :0 :1 :2 :1
```

Set repetitive mapping. This command:

```
dmsetup message switch 0 set_region_mappings 1000:1 :2 R2,10
```

is equivalent to:

```
dmsetup message switch 0 set_region_mappings 1000:1 :2 :1 :2 :1 :2 :1 :2 \
    :1 :2 :1 :2 :1 :2 :1 :2 :1 :2
```

## 34.27 Thin provisioning

### 34.27.1 Introduction

This document describes a collection of device-mapper targets that between them implement thin-provisioning and snapshots.

The main highlight of this implementation, compared to the previous implementation of snapshots, is that it allows many virtual devices to be stored on the same data volume. This simplifies administration and allows the sharing of data between volumes, thus reducing disk usage.

Another significant feature is support for an arbitrary depth of recursive snapshots (snapshots of snapshots of snapshots ...). The previous implementation of snapshots did this by chaining together lookup tables, and so performance was  $O(\text{depth})$ . This new implementation uses a single data structure to avoid this degradation with depth. Fragmentation may still be an issue, however, in some scenarios.

Metadata is stored on a separate device from data, giving the administrator some freedom, for example to:

- Improve metadata resilience by storing metadata on a mirrored volume but data on a non-mirrored one.
- Improve performance by storing the metadata on SSD.

### 34.27.2 Status

These targets are considered safe for production use. But different use cases will have different performance characteristics, for example due to fragmentation of the data volume.

If you find this software is not performing as expected please mail [dm-devel@redhat.com](mailto:dm-devel@redhat.com) with details and we'll try our best to improve things for you.

Userspace tools for checking and repairing the metadata have been fully developed and are available as 'thin\_check' and 'thin\_repair'. The name of the package that provides these utilities varies by distribution (on a Red Hat distribution it is named 'device-mapper-persistent-data').

### 34.27.3 Cookbook

This section describes some quick recipes for using thin provisioning. They use the `dmsetup` program to control the device-mapper driver directly. End users will be advised to use a higher-level volume manager such as LVM2 once support has been added.

#### Pool device

The pool device ties together the metadata volume and the data volume. It maps I/O linearly to the data volume and updates the metadata via two mechanisms:

- Function calls from the thin targets
- Device-mapper 'messages' from userspace which control the creation of new virtual devices amongst other things.

#### Setting up a fresh pool device

Setting up a pool device requires a valid metadata device, and a data device. If you do not have an existing metadata device you can make one by zeroing the first 4k to indicate empty metadata.

```
dd if=/dev/zero of=$metadata_dev bs=4096 count=1
```

The amount of metadata you need will vary according to how many blocks are shared between thin devices (i.e. through snapshots). If you have less sharing than average you'll need a larger-than-average metadata device.

As a guide, we suggest you calculate the number of bytes to use in the metadata device as  $48 * \$data\_dev\_size / \$data\_block\_size$  but round it up to 2MB if the answer is smaller. If you're creating large numbers of snapshots which are recording large amounts of change, you may find you need to increase this.

The largest size supported is 16GB: If the device is larger, a warning will be issued and the excess space will not be used.

### Reloading a pool table

You may reload a pool's table, indeed this is how the pool is resized if it runs out of space. (N.B. While specifying a different metadata device when reloading is not forbidden at the moment, things will go wrong if it does not route I/O to exactly the same on-disk location as previously.)

### Using an existing pool device

```
dmsetup create pool \  
  --table "0 20971520 thin-pool $metadata_dev $data_dev \  
          $data_block_size $low_water_mark"
```

`$data_block_size` gives the smallest unit of disk space that can be allocated at a time expressed in units of 512-byte sectors. `$data_block_size` must be between 128 (64KB) and 2097152 (1GB) and a multiple of 128 (64KB). `$data_block_size` cannot be changed after the thin-pool is created. People primarily interested in thin provisioning may want to use a value such as 1024 (512KB). People doing lots of snapshotting may want a smaller value such as 128 (64KB). If you are not zeroing newly-allocated data, a larger `$data_block_size` in the region of 256000 (128MB) is suggested.

`$low_water_mark` is expressed in blocks of size `$data_block_size`. If free space on the data device drops below this level then a dm event will be triggered which a userspace daemon should catch allowing it to extend the pool device. Only one such event will be sent.

No special event is triggered if a just resumed device's free space is below the low water mark. However, resuming a device always triggers an event; a userspace daemon should verify that free space exceeds the low water mark when handling this event.

A low water mark for the metadata device is maintained in the kernel and will trigger a dm event if free space on the metadata device drops below it.

### Updating on-disk metadata

On-disk metadata is committed every time a FLUSH or FUA bio is written. If no such requests are made then commits will occur every second. This means the thin-provisioning target behaves like a physical disk that has a volatile write cache. If power is lost you may lose some recent writes. The metadata should always be consistent in spite of any crash.

If data space is exhausted the pool will either error or queue IO according to the configuration (see: `error_if_no_space`). If metadata space is exhausted or a metadata operation fails: the pool will error IO until the pool is taken offline and repair is performed to 1) fix any potential inconsistencies and 2) clear the flag that imposes repair. Once the pool's metadata device is repaired it may be resized, which will allow the pool to return to normal operation. Note that if a pool is flagged as needing repair, the pool's data and metadata devices cannot be resized until repair is performed. It should also be noted that when the pool's metadata space is exhausted the current metadata transaction is aborted. Given that the pool will

cache IO whose completion may have already been acknowledged to upper IO layers (e.g. filesystem) it is strongly suggested that consistency checks (e.g. fsck) be performed on those layers when repair of the pool is required.

## Thin provisioning

- i) Creating a new thinly-provisioned volume.

To create a new thinly- provisioned volume you must send a message to an active pool device, /dev/mapper/pool in this example:

```
dmsetup message /dev/mapper/pool 0 "create_thin 0"
```

Here ‘0’ is an identifier for the volume, a 24-bit number. It’s up to the caller to allocate and manage these identifiers. If the identifier is already in use, the message will fail with -EEXIST.

- ii) Using a thinly-provisioned volume.

Thinly-provisioned volumes are activated using the ‘thin’ target:

```
dmsetup create thin --table "0 2097152 thin /dev/mapper/pool 0"
```

The last parameter is the identifier for the thinp device.

## Internal snapshots

- i) Creating an internal snapshot.

Snapshots are created with another message to the pool.

N.B. If the origin device that you wish to snapshot is active, you must suspend it before creating the snapshot to avoid corruption. This is NOT enforced at the moment, so please be careful!

```
dmsetup suspend /dev/mapper/thin  
dmsetup message /dev/mapper/pool 0 "create_snap 1 0"  
dmsetup resume /dev/mapper/thin
```

Here ‘1’ is the identifier for the volume, a 24-bit number. ‘0’ is the identifier for the origin device.

- ii) Using an internal snapshot.

Once created, the user doesn’t have to worry about any connection between the origin and the snapshot. Indeed the snapshot is no different from any other thinly-provisioned device and can be snapshotted itself via the same method. It’s perfectly legal to have only one of them active, and there’s no ordering requirement on activating or removing them both. (This differs from conventional device-mapper snapshots.)

Activate it exactly the same way as any other thinly-provisioned volume:

```
dmsetup create snap --table "0 2097152 thin /dev/mapper/pool 1"
```

### External snapshots

You can use an external **read only** device as an origin for a thinly-provisioned volume. Any read to an unprovisioned area of the thin device will be passed through to the origin. Writes trigger the allocation of new blocks as usual.

One use case for this is VM hosts that want to run guests on thinly-provisioned volumes but have the base image on another device (possibly shared between many VMs).

You must not write to the origin device if you use this technique! Of course, you may write to the thin device and take internal snapshots of the thin volume.

#### i) Creating a snapshot of an external device

This is the same as creating a thin device. You don't mention the origin at this stage.

```
dmsetup message /dev/mapper/pool 0 "create_thin 0"
```

#### ii) Using a snapshot of an external device.

Append an extra parameter to the thin target specifying the origin:

```
dmsetup create snap --table "0 2097152 thin /dev/mapper/pool 0 /  
↪ dev/image"
```

N.B. All descendants (internal snapshots) of this snapshot require the same extra origin parameter.

### Deactivation

All devices using a pool must be deactivated before the pool itself can be.

```
dmsetup remove thin  
dmsetup remove snap  
dmsetup remove pool
```

## 34.27.4 Reference

### 'thin-pool' target

#### i) Constructor

```
thin-pool <metadata dev> <data dev> <data block size>_  
↪ (sectors)> \  
    <low water mark (blocks)> [<number of feature args>_  
↪ [<arg>]*]
```

Optional feature arguments:

**skip\_block\_zeroing:** Skip the zeroing of newly-provisioned blocks.

**ignore\_discard:** Disable discard support.

**no\_discard\_passthrough:** Don't pass discards down to the underlying data device, but just remove the mapping.

**read\_only:** Don't allow any changes to be made to the pool metadata. This mode is only available after the thin-pool has been created and first used in full read/write mode. It cannot be specified on initial thin-pool creation.

**error\_if\_no\_space:** Error IOs, instead of queueing, if no space.

Data block size must be between 64KB (128 sectors) and 1GB (2097152 sectors) inclusive.

## ii) Status

```
<transaction id> <used metadata blocks>/<total metadata blocks>
<used data blocks>/<total data blocks> <held metadata root>
ro|rw|out_of_data_space [no_]discard_passthrough [error|queue]_if_no_
↪space
needs_check|- metadata_low_watermark
```

**transaction id:** A 64-bit number used by userspace to help synchronise with metadata from volume managers.

**used data blocks / total data blocks** If the number of free blocks drops below the pool's low water mark a dm event will be sent to userspace. This event is edge-triggered and it will occur only once after each resume so volume manager writers should register for the event and then check the target's status.

**held metadata root:** The location, in blocks, of the metadata root that has been 'held' for userspace read access. '-' indicates there is no held root.

**discard\_passthrough|no\_discard\_passthrough** Whether or not discards are actually being passed down to the underlying device. When this is enabled when loading the table, it can get disabled if the underlying device doesn't support it.

**ro|rw|out\_of\_data\_space** If the pool encounters certain types of device failures it will drop into a read-only metadata mode in which no changes to the pool metadata (like allocating new blocks) are permitted.

In serious cases where even a read-only mode is deemed unsafe no further I/O will be permitted and the status will just contain the string 'Fail'. The userspace recovery tools should then be used.

**error\_if\_no\_space|queue\_if\_no\_space** If the pool runs out of data or metadata space, the pool will either queue or error the IO destined to the data device. The default is to queue the IO until more space is added or the 'no\_space\_timeout' expires. The 'no\_space\_timeout' dm-thin-pool module parameter can be used to change this timeout - it defaults to 60 seconds but may be disabled using a value of 0.

**needs\_check** A metadata operation has failed, resulting in the needs\_check flag being set in the metadata's superblock. The metadata device must

be deactivated and checked/repared before the thin-pool can be made fully operational again. ‘-‘ indicates needs\_check is not set.

**metadata\_low\_watermark:** Value of metadata low watermark in blocks. The kernel sets this value internally but userspace needs to know this value to determine if an event was caused by crossing this threshold.

### iii) Messages

**create\_thin <dev id>** Create a new thinly-provisioned device. <dev id> is an arbitrary unique 24-bit identifier chosen by the caller.

**create\_snap <dev id> <origin id>** Create a new snapshot of another thinly-provisioned device. <dev id> is an arbitrary unique 24-bit identifier chosen by the caller. <origin id> is the identifier of the thinly-provisioned device of which the new device will be a snapshot.

**delete <dev id>** Deletes a thin device. Irreversible.

**set\_transaction\_id <current id> <new id>** Userland volume managers, such as LVM, need a way to synchronise their external metadata with the internal metadata of the pool target. The thin-pool target offers to store an arbitrary 64-bit transaction id and return it on the target’s status line. To avoid races you must provide what you think the current transaction id is when you change it with this compare-and-swap message.

**reserve\_metadata\_snap** Reserve a copy of the data mapping btree for use by userland. This allows userland to inspect the mappings as they were when this message was executed. Use the pool’s status command to get the root block associated with the metadata snapshot.

**release\_metadata\_snap** Release a previously reserved copy of the data mapping btree.

## ‘thin’ target

### i) Constructor

```
thin <pool dev> <dev id> [<external origin dev>]
```

**pool dev:** the thin-pool device, e.g. /dev/mapper/my\_pool or 253:0

**dev id:** the internal device identifier of the device to be activated.

**external origin dev:** an optional block device outside the pool to be treated as a read-only snapshot origin: reads to unprovisioned areas of the thin target will be mapped to this device.

The pool doesn’t store any size against the thin devices. If you load a thin target that is smaller than you’ve been using previously, then you’ll have no access to blocks mapped beyond the end. If you load a target that is bigger than before, then extra blocks will be provisioned as and when needed.

### ii) Status

**<nr mapped sectors>** **<highest mapped sector>** If the pool has encountered device errors and failed, the status will just contain the string ‘Fail’. The userspace recovery tools should then be used.

In the case where **<nr mapped sectors>** is 0, there is no highest mapped sector and the value of **<highest mapped sector>** is unspecified.

## 34.28 Device-mapper “unstriped” target

### 34.28.1 Introduction

The device-mapper “unstriped” target provides a transparent mechanism to unstripe a device-mapper “striped” target to access the underlying disks without having to touch the true backing block-device. It can also be used to unstripe a hardware RAID-0 to access backing disks.

Parameters: **<number of stripes>** **<chunk size>** **<stripe #>** **<dev\_path>** **<offset>**

**<number of stripes>** The number of stripes in the RAID 0.

**<chunk size>** The amount of 512B sectors in the chunk striping.

**<dev\_path>** The block device you wish to unstripe.

**<stripe #>** The stripe number within the device that corresponds to physical drive you wish to unstripe. This must be 0 indexed.

### 34.28.2 Why use this module?

#### An example of undoing an existing dm-stripe

This small bash script will setup 4 loop devices and use the existing striped target to combine the 4 devices into one. It then will use the unstriped target ontop of the striped device to access the individual backing loop devices. We write data to the newly exposed unstriped devices and verify the data written matches the correct underlying device on the striped array:

```
#!/bin/bash

MEMBER_SIZE=$((128 * 1024 * 1024))
NUM=4
SEQ_END=$(( ${NUM} - 1 ))
CHUNK=256
BS=4096

RAID_SIZE=$(( ${MEMBER_SIZE} * ${NUM} / 512 ))
DM_PARAMS="0 ${RAID_SIZE} striped ${NUM} ${CHUNK}"
COUNT=$(( ${MEMBER_SIZE} / ${BS} ))

for i in $(seq 0 ${SEQ_END}); do
  dd if=/dev/zero of=member-${i} bs=${MEMBER_SIZE} count=1 oflag=direct
  losetup /dev/loop${i} member-${i}
  DM_PARAMS+=" /dev/loop${i} 0"
done
```

(continues on next page)

(continued from previous page)

```
done

echo $DM_PARMS | dmsetup create raid0
for i in $(seq 0 ${SEQ_END}); do
    echo "0 1 unstriped ${NUM} ${CHUNK} ${i} /dev/mapper/raid0 0" | dmsetup_
    ↪create set-${i}
done;

for i in $(seq 0 ${SEQ_END}); do
    dd if=/dev/urandom of=/dev/mapper/set-${i} bs=${BS} count=${COUNT}_
    ↪oflag=direct
    diff /dev/mapper/set-${i} member-${i}
done;

for i in $(seq 0 ${SEQ_END}); do
    dmsetup remove set-${i}
done

dmsetup remove raid0

for i in $(seq 0 ${SEQ_END}); do
    losetup -d /dev/loop${i}
    rm -f member-${i}
done
```

### Another example

Intel NVMe drives contain two cores on the physical device. Each core of the drive has segregated access to its LBA range. The current LBA model has a RAID 0 128k chunk on each core, resulting in a 256k stripe across the two cores:

Core 0:	Core 1:
LBA 512	LBA 768
LBA 0	LBA 256
-----	-----

The purpose of this unstriping is to provide better QoS in noisy neighbor environments. When two partitions are created on the aggregate drive without this unstriping, reads on one partition can affect writes on another partition. This is because the partitions are striped across the two cores. When we unstripe this hardware RAID 0 and make partitions on each new exposed device the two partitions are now physically separated.

With the dm-unstriped target we're able to segregate an fio script that has read and write jobs that are independent of each other. Compared to when we run the test on a combined drive with partitions, we were able to get a 92% reduction in read latency using this device mapper target.

### 34.28.3 Example dmsetup usage

#### unstriped ontop of Intel NVMe device that has 2 cores

```
dmsetup create nvms0 --table '0 512 unstriped 2 256 0 /dev/nvme0n1 0'
dmsetup create nvms1 --table '0 512 unstriped 2 256 1 /dev/nvme0n1 0'
```

There will now be two devices that expose Intel NVMe core 0 and 1 respectively:

```
/dev/mapper/nvms0
/dev/mapper/nvms1
```

#### unstriped ontop of striped with 4 drives using 128K chunk size

```
dmsetup create raid_disk0 --table '0 512 unstriped 4 256 0 /dev/mapper/
↳ striped 0'
dmsetup create raid_disk1 --table '0 512 unstriped 4 256 1 /dev/mapper/
↳ striped 0'
dmsetup create raid_disk2 --table '0 512 unstriped 4 256 2 /dev/mapper/
↳ striped 0'
dmsetup create raid_disk3 --table '0 512 unstriped 4 256 3 /dev/mapper/
↳ striped 0'
```

## 34.29 dm-verity

Device-Mapper's "verity" target provides transparent integrity checking of block devices using a cryptographic digest provided by the kernel crypto API. This target is read-only.

### 34.29.1 Construction Parameters

```
<version> <dev> <hash_dev>
<data_block_size> <hash_block_size>
<num_data_blocks> <hash_start_block>
<algorithm> <digest> <salt>
[<#opt_params> <opt_params>]
```

**<version>** This is the type of the on-disk hash format.

**0 is the original format used in the Chromium OS.** The salt is appended when hashing, digests are stored continuously and the rest of the block is padded with zeroes.

**1 is the current format that should be used for new devices.** The salt is prepended when hashing and each digest is padded with zeroes to the power of two.

**<dev>** This is the device containing data, the integrity of which needs to be checked. It may be specified as a path, like /dev/sdaX, or a device number, <major>:<minor>.

**<hash\_dev>** This is the device that supplies the hash tree data. It may be specified similarly to the device path and may be the same device. If the same device is used, the `hash_start` should be outside the configured dm-verity device.

**<data\_block\_size>** The block size on a data device in bytes. Each block corresponds to one digest on the hash device.

**<hash\_block\_size>** The size of a hash block in bytes.

**<num\_data\_blocks>** The number of data blocks on the data device. Additional blocks are inaccessible. You can place hashes to the same partition as data, in this case hashes are placed after `<num_data_blocks>`.

**<hash\_start\_block>** This is the offset, in `<hash_block_size>`-blocks, from the start of `hash_dev` to the root block of the hash tree.

**<algorithm>** The cryptographic hash algorithm used for this device. This should be the name of the algorithm, like “sha1” .

**<digest>** The hexadecimal encoding of the cryptographic hash of the root hash block and the salt. This hash should be trusted as there is no other authenticity beyond this point.

**<salt>** The hexadecimal encoding of the salt value.

**<#opt\_params>** Number of optional parameters. If there are no optional parameters, the optional parameters section can be skipped or `#opt_params` can be zero. Otherwise `#opt_params` is the number of following arguments.

**Example of optional parameters section:** 1 ignore\_corruption

**ignore\_corruption** Log corrupted blocks, but allow read operations to proceed normally.

**restart\_on\_corruption** Restart the system when a corrupted block is discovered. This option is not compatible with `ignore_corruption` and requires user space support to avoid restart loops.

**ignore\_zero\_blocks** Do not verify blocks that are expected to contain zeroes and always return zeroes instead. This may be useful if the partition contains unused blocks that are not guaranteed to contain zeroes.

**use\_fec\_from\_device <fec\_dev>** Use forward error correction (FEC) to recover from corruption if hash verification fails. Use encoding data from the specified device. This may be the same device where data and hash blocks reside, in which case `fec_start` must be outside data and hash areas.

If the encoding data covers additional metadata, it must be accessible on the hash device after the hash blocks.

Note: block sizes for data and hash devices must match. Also, if the verity `<dev>` is encrypted the `<fec_dev>` should be too.

**fec\_roots <num>** Number of generator roots. This equals to the number of parity bytes in the encoding data. For example, in RS(M, N) encoding, the number of roots is M-N.

**fec\_blocks <num>** The number of encoding data blocks on the FEC device. The block size for the FEC device is `<data_block_size>`.

**fec\_start <offset>** This is the offset, in <data\_block\_size> blocks, from the start of the FEC device to the beginning of the encoding data.

**check\_at\_most\_once** Verify data blocks only the first time they are read from the data device, rather than every time. This reduces the overhead of dm-verity so that it can be used on systems that are memory and/or CPU constrained. However, it provides a reduced level of security because only offline tampering of the data device's content will be detected, not online tampering.

Hash blocks are still verified each time they are read from the hash device, since verification of hash blocks is less performance critical than data blocks, and a hash block will not be verified any more after all the data blocks it covers have been verified anyway.

**root\_hash\_sig\_key\_desc <key\_description>** This is the description of the USER\_KEY that the kernel will lookup to get the pkcs7 signature of the roothash. The pkcs7 signature is used to validate the root hash during the creation of the device mapper block device. Verification of roothash depends on the config DM\_VERITY\_VERIFY\_ROOTHASH\_SIG being set in the kernel.

### 34.29.2 Theory of operation

dm-verity is meant to be set up as part of a verified boot path. This may be anything ranging from a boot using tboot or trustedgrub to just booting from a known-good device (like a USB drive or CD).

When a dm-verity device is configured, it is expected that the caller has been authenticated in some way (cryptographic signatures, etc). After instantiation, all hashes will be verified on-demand during disk access. If they cannot be verified up to the root node of the tree, the root hash, then the I/O will fail. This should detect tampering with any data on the device and the hash data.

Cryptographic hashes are used to assert the integrity of the device on a per-block basis. This allows for a lightweight hash computation on first read into the page cache. Block hashes are stored linearly, aligned to the nearest block size.

If forward error correction (FEC) support is enabled any recovery of corrupted data will be verified using the cryptographic hash of the corresponding data. This is why combining error correction with integrity checking is essential.

#### Hash Tree

Each node in the tree is a cryptographic hash. If it is a leaf node, the hash of some data block on disk is calculated. If it is an intermediary node, the hash of a number of child nodes is calculated.

Each entry in the tree is a collection of neighboring nodes that fit in one block. The number is determined based on block\_size and the size of the selected cryptographic digest algorithm. The hashes are linearly-ordered in this entry and any unaligned trailing space is ignored but included when calculating the parent node.

The tree looks something like:

```
alg = sha256, num_blocks = 32768, block_size = 4096
```



```
# veritysetup create vroot /dev/sda1 /dev/sda2 \
4392712ba01368efdf14b05c76f9e4df0d53664630b5d48632ed17a137f39076
```

## 34.30 Writecache target

The writecache target caches writes on persistent memory or on SSD. It doesn't cache reads because reads are supposed to be cached in page cache in normal RAM.

When the device is constructed, the first sector should be zeroed or the first sector should contain valid superblock from previous invocation.

Constructor parameters:

1. type of the cache device - “p” or “s”
  - p - persistent memory
  - s - SSD
2. the underlying device that will be cached
3. the cache device
4. block size (4096 is recommended; the maximum block size is the page size)
5. the number of optional parameters (the parameters with an argument count as two)

**start\_sector n (default: 0)** offset from the start of cache device in 512-byte sectors

**high\_watermark n (default: 50)** start writeback when the number of used blocks reach this watermark

**low\_watermark x (default: 45)** stop writeback when the number of used blocks drops below this watermark

**writeback\_jobs n (default: unlimited)** limit the number of blocks that are in flight during writeback. Setting this value reduces writeback throughput, but it may improve latency of read requests

**autocommit\_blocks n (default: 64 for pmem, 65536 for ssd)** when the application writes this amount of blocks without issuing the FLUSH request, the blocks are automatically committed

**autocommit\_time ms (default: 1000)** autocommit time in milliseconds. The data is automatically committed if this time passes and no FLUSH request is received

**fua (by default on)** applicable only to persistent memory - use the FUA flag when writing data from persistent memory back to the underlying device

**nofua** applicable only to persistent memory - don't use the FUA flag when writing back data and send the FLUSH request afterwards

- some underlying devices perform better with fua, some with nofua. The user should test it

Status: 1. error indicator - 0 if there was no error, otherwise error number 2. the number of blocks 3. the number of free blocks 4. the number of blocks under writeback

### Messages:

**flush** flush the cache device. The message returns successfully if the cache device was flushed without an error

**flush\_on\_suspend** flush the cache device on next suspend. Use this message when you are going to remove the cache device. The proper sequence for removing the cache device is:

1. send the “flush\_on\_suspend” message
2. load an inactive table with a linear target that maps to the underlying device
3. suspend the device
4. ask for status and verify that there are no errors
5. resume the device, so that it will use the linear target
6. the cache device is now inactive and it can be deleted

## 34.31 dm-zero

Device-Mapper’s “zero” target provides a block-device that always returns zero’ d data on reads and silently drops writes. This is similar behavior to /dev/zero, but as a block-device instead of a character-device.

Dm-zero has no target-specific parameters.

One very interesting use of dm-zero is for creating “sparse” devices in conjunction with dm-snapshot. A sparse device reports a device-size larger than the amount of actual storage space available for that device. A user can write data anywhere within the sparse device and read it back like a normal device. Reads to previously unwritten areas will return a zero’ d buffer. When enough data has been written to fill up the actual storage space, the sparse device is deactivated. This can be very useful for testing device and filesystem limitations.

To create a sparse device, start by creating a dm-zero device that’ s the desired size of the sparse device. For this example, we’ ll assume a 10TB sparse device:

```
TEN_TERABYTES=`expr 10 \* 1024 \* 1024 \* 1024 \* 2` # 10 TB in sectors
echo "0 $TEN_TERABYTES zero" | dmsetup create zero1
```

Then create a snapshot of the zero device, using any available block-device as the COW device. The size of the COW device will determine the amount of real space available to the sparse device. For this example, we’ ll assume /dev/sdb1 is an available 10GB partition:

```
echo "0 $TEN_TERABYTES snapshot /dev/mapper/zero1 /dev/sdb1 p 128" | \
dmsetup create sparse1
```

This will create a 10TB sparse device called `/dev/mapper/sparse1` that has 10GB of actual storage space available. If more than 10GB of data is written to this device, it will start returning I/O errors.



## **EDID**

In the good old days when graphics parameters were configured explicitly in a file called `xorg.conf`, even broken hardware could be managed.

Today, with the advent of Kernel Mode Setting, a graphics board is either correctly working because all components follow the standards - or the computer is unusable, because the screen remains dark after booting or it displays the wrong area. Cases when this happens are:

- The graphics board does not recognize the monitor.
- The graphics board is unable to detect any EDID data.
- The graphics board incorrectly forwards EDID data to the driver.
- The monitor sends no or bogus EDID data.
- A KVM sends its own EDID data instead of querying the connected monitor.

Adding the kernel parameter “`nomodeset`” helps in most cases, but causes restrictions later on.

As a remedy for such situations, the kernel configuration item `CONFIG_DRM_LOAD_EDID_FIRMWARE` was introduced. It allows to provide an individually prepared or corrected EDID data set in the `/lib/firmware` directory from where it is loaded via the firmware interface. The code (see `drivers/gpu/drm/drm_edid_load.c`) contains built-in data sets for commonly used screen resolutions (800x600, 1024x768, 1280x1024, 1600x1200, 1680x1050, 1920x1080) as binary blobs, but the kernel source tree does not contain code to create these data. In order to elucidate the origin of the built-in binary EDID blobs and to facilitate the creation of individual data for a specific misbehaving monitor, commented sources and a Makefile environment are given here.

To create binary EDID and C source code files from the existing data material, simply type “`make`” in `tools/edid/`.

If you want to create your own EDID file, copy the file `1024x768.S`, replace the settings with your own data and add a new target to the Makefile. Please note that the EDID data structure expects the timing values in a different way as compared to the standard X11 format.

### **X11:**

**HTimings:** `hdisp hsyncstart hsyncend htotal`

**VTimings:** `vdisp vsyncstart vsyncend vtotal`

EDID:

```
#define XPIX hdisp
#define XBLANK htotal-hdisp
#define XOFFSET hsyncstart-hdisp
#define XPULSE hsyncend-hsyncstart

#define YPIX vdisp
#define YBLANK vtotal-vdisp
#define YOFFSET vsyncstart-vdisp
#define YPULSE vsyncend-vsyncstart
```

## **THE EFI BOOT STUB**

On the x86 and ARM platforms, a kernel zImage/bzImage can masquerade as a PE/COFF image, thereby convincing EFI firmware loaders to load it as an EFI executable. The code that modifies the bzImage header, along with the EFI-specific entry point that the firmware loader jumps to are collectively known as the “EFI boot stub”, and live in arch/x86/boot/header.S and arch/x86/boot/compressed/eboot.c, respectively. For ARM the EFI stub is implemented in arch/arm/boot/compressed/efi-header.S and arch/arm/boot/compressed/efi-stub.c. EFI stub code that is shared between architectures is in drivers/firmware/efi/libstub.

For arm64, there is no compressed kernel support, so the Image itself masquerades as a PE/COFF image and the EFI stub is linked into the kernel. The arm64 EFI stub lives in arch/arm64/kernel/efi-entry.S and drivers/firmware/efi/libstub/arm64-stub.c.

By using the EFI boot stub it's possible to boot a Linux kernel without the use of a conventional EFI boot loader, such as grub or elilo. Since the EFI boot stub performs the jobs of a boot loader, in a certain sense it IS the boot loader.

The EFI boot stub is enabled with the CONFIG\_EFI\_STUB kernel option.

### **36.1 How to install bzImage.efi**

The bzImage located in arch/x86/boot/bzImage must be copied to the EFI System Partition (ESP) and renamed with the extension “.efi”. Without the extension the EFI firmware loader will refuse to execute it. It's not possible to execute bzImage.efi from the usual Linux file systems because EFI firmware doesn't have support for them. For ARM the arch/arm/boot/zImage should be copied to the system partition, and it may not need to be renamed. Similarly for arm64, arch/arm64/boot/Image should be copied but not necessarily renamed.

## 36.2 Passing kernel parameters from the EFI shell

Arguments to the kernel can be passed after `bzImage.efi`, e.g.:

```
fs0:> bzImage.efi console=ttyS0 root=/dev/sda4
```

## 36.3 The “initrd=” option

Like most boot loaders, the EFI stub allows the user to specify multiple `initrd` files using the “`initrd=`” option. This is the only EFI stub-specific command line parameter, everything else is passed to the kernel when it boots.

The path to the `initrd` file must be an absolute path from the beginning of the ESP, relative path names do not work. Also, the path is an EFI-style path and directory elements must be separated with backslashes (`\`). For example, given the following directory layout:

```
fs0:>
  Kernels\
          bzImage.efi
          initrd-large.img

  Ramdisks\
          initrd-small.img
          initrd-medium.img
```

to boot with the `initrd-large.img` file if the current working directory is `fs0:Kernels`, the following command must be used:

```
fs0:\Kernels> bzImage.efi initrd=\Kernels\initrd-large.img
```

Notice how `bzImage.efi` can be specified with a relative path. That’s because the image we’re executing is interpreted by the EFI shell, which understands relative paths, whereas the rest of the command line is passed to `bzImage.efi`.

## 36.4 The “dtb=” option

For the ARM and arm64 architectures, a device tree must be provided to the kernel. Normally firmware shall supply the device tree via the EFI CONFIGURATION TABLE. However, the “`dtb=`” command line option can be used to override the firmware supplied device tree, or to supply one when firmware is unable to.

Please note: Firmware adds runtime configuration information to the device tree before booting the kernel. If `dtb=` is used to override the device tree, then any runtime data provided by firmware will be lost. The `dtb=` option should only be used either as a debug tool, or as a last resort when a device tree is not provided in the EFI CONFIGURATION TABLE.

“`dtb=`” is processed in the same manner as the “`initrd=`” option that is described above.

## EXT4 GENERAL INFORMATION

Ext4 is an advanced level of the ext3 filesystem which incorporates scalability and reliability enhancements for supporting large filesystems (64 bit) in keeping with increasing disk capacities and state-of-the-art feature requirements.

Mailing list: [linux-ext4@vger.kernel.org](mailto:linux-ext4@vger.kernel.org) Web site: <http://ext4.wiki.kernel.org>

### 37.1 Quick usage instructions

Note: More extensive information for getting started with ext4 can be found at the ext4 wiki site at the URL: [http://ext4.wiki.kernel.org/index.php/Ext4\\_Howto](http://ext4.wiki.kernel.org/index.php/Ext4_Howto)

- The latest version of e2fsprogs can be found at:

<https://www.kernel.org/pub/linux/kernel/people/tytso/e2fsprogs/>

or

[http://sourceforge.net/project/showfiles.php?group\\_id=2406](http://sourceforge.net/project/showfiles.php?group_id=2406)

or grab the latest git repository from:

<https://git.kernel.org/pub/scm/fs/ext2/e2fsprogs.git>

- Create a new filesystem using the ext4 filesystem type:

```
# mke2fs -t ext4 /dev/hda1
```

Or to configure an existing ext3 filesystem to support extents:

```
# tune2fs -O extents /dev/hda1
```

If the filesystem was created with 128 byte inodes, it can be converted to use 256 byte for greater efficiency via:

```
# tune2fs -I 256 /dev/hda1
```

- Mounting:

```
# mount -t ext4 /dev/hda1 /wherever
```

- When comparing performance with other filesystems, it's always important to try multiple workloads; very often a subtle change in a workload parameter can completely change the ranking of which filesystems do well compared to others. When comparing versus ext3, note that ext4 enables write barriers by default, while ext3

does not enable write barriers by default. So it is useful to use explicitly specify whether barriers are enabled or not when via the ‘-o barriers=[0|1]’ mount option for both ext3 and ext4 filesystems for a fair comparison. When tuning ext3 for best benchmark numbers, it is often worthwhile to try changing the data journaling mode; ‘-o data=writeback’ can be faster for some workloads. (Note however that running mounted with data=writeback can potentially leave stale data exposed in recently written files in case of an unclean shutdown, which could be a security exposure in some situations.) Configuring the filesystem with a large journal can also be helpful for metadata-intensive workloads.

## 37.2 Features

### 37.2.1 Currently Available

- ability to use filesystems > 16TB (e2fsprogs support not available yet)
- extent format reduces metadata overhead (RAM, IO for access, transactions)
- extent format more robust in face of on-disk corruption due to magics,
- internal redundancy in tree
- improved file allocation (multi-block alloc)
- lift 32000 subdirectory limit imposed by `i_links_count[1]`
- nsec timestamps for mtime, atime, ctime, create time
- inode version field on disk (NFSv4, Lustre)
- reduced e2fsck time via `uninit_bg` feature
- journal checksumming for robustness, performance
- persistent file preallocation (e.g for streaming media, databases)
- ability to pack bitmaps and inode tables into larger virtual groups via the `flex_bg` feature
- large file support
- inode allocation using large virtual block groups via `flex_bg`
- delayed allocation
- large block (up to pagesize) support
- efficient new ordered mode in JBD2 and ext4 (avoid using buffer head to force the ordering)
- Case-insensitive file name lookups
- file-based encryption support (fscrypt)
- file-based verity support (fsverity)

[1] Filesystems with a block size of 1k may see a limit imposed by the directory hash tree having a maximum depth of two.

## 37.3 case-insensitive file name lookups

The case-insensitive file name lookup feature is supported on a per-directory basis, allowing the user to mix case-insensitive and case-sensitive directories in the same filesystem. It is enabled by flipping the +F inode attribute of an empty directory. The case-insensitive string match operation is only defined when we know how text is encoded in a byte sequence. For that reason, in order to enable case-insensitive directories, the filesystem must have the casefold feature, which stores the filesystem-wide encoding model used. By default, the charset adopted is the latest version of Unicode (12.1.0, by the time of this writing), encoded in the UTF-8 form. The comparison algorithm is implemented by normalizing the strings to the Canonical decomposition form, as defined by Unicode, followed by a byte per byte comparison.

The case-awareness is name-preserving on the disk, meaning that the file name provided by userspace is a byte-per-byte match to what is actually written in the disk. The Unicode normalization format used by the kernel is thus an internal representation, and not exposed to the userspace nor to the disk, with the important exception of disk hashes, used on large case-insensitive directories with DX feature. On DX directories, the hash must be calculated using the casefolded version of the filename, meaning that the normalization format used actually has an impact on where the directory entry is stored.

When we change from viewing filenames as opaque byte sequences to seeing them as encoded strings we need to address what happens when a program tries to create a file with an invalid name. The Unicode subsystem within the kernel leaves the decision of what to do in this case to the filesystem, which select its preferred behavior by enabling/disabling the strict mode. When Ext4 encounters one of those strings and the filesystem did not require strict mode, it falls back to considering the entire string as an opaque byte sequence, which still allows the user to operate on that file, but the case-insensitive lookups won't work.

## 37.4 Options

When mounting an ext4 filesystem, the following options are accepted: (\*) == default

**ro** Mount filesystem read only. Note that ext4 will replay the journal (and thus write to the partition) even when mounted “read only”. The mount options “ro,noload” can be used to prevent writes to the filesystem.

**journal\_checksum** Enable checksumming of the journal transactions. This will allow the recovery code in e2fsck and the kernel to detect corruption in the kernel. It is a compatible change and will be ignored by older kernels.

**journal\_async\_commit** Commit block can be written to disk without waiting for descriptor blocks. If enabled older kernels cannot mount the device. This will enable ‘journal\_checksum’ internally.

**journal\_path=path, journal\_dev=devnum** When the external journal

device's major/minor numbers have changed, these options allow the user to specify the new journal location. The journal device is identified through either its new major/minor numbers encoded in `devnum`, or via a path to the device.

**norecovery, noload** Don't load the journal on mounting. Note that if the filesystem was not unmounted cleanly, skipping the journal replay will lead to the filesystem containing inconsistencies that can lead to any number of problems.

**data=journal** All data are committed into the journal prior to being written into the main file system. Enabling this mode will disable delayed allocation and `O_DIRECT` support.

**data=ordered (\*)** All data are forced directly out to the main file system prior to its metadata being committed to the journal.

**data=writeback** Data ordering is not preserved, data may be written into the main file system after its metadata has been committed to the journal.

**commit=nrsec (\*)** This setting limits the maximum age of the running transaction to 'nrsec' seconds. The default value is 5 seconds. This means that if you lose your power, you will lose as much as the latest 5 seconds of metadata changes (your filesystem will not be damaged though, thanks to the journaling). This default value (or any low value) will hurt performance, but it's good for data-safety. Setting it to 0 will have the same effect as leaving it at the default (5 seconds). Setting it to very large values will improve performance. Note that due to delayed allocation even older data can be lost on power failure since writeback of those data begins only after time set in `/proc/sys/vm/dirty_expire_centisecs`.

**barrier=<0|1(\*)>, barrier(\*), nobarrier** This enables/disables the use of write barriers in the `jbd` code. `barrier=0` disables, `barrier=1` enables. This also requires an IO stack which can support barriers, and if `jbd` gets an error on a barrier write, it will disable again with a warning. Write barriers enforce proper on-disk ordering of journal commits, making volatile disk write caches safe to use, at some performance penalty. If your disks are battery-backed in one way or another, disabling barriers may safely improve performance. The mount options "barrier" and "nobarrier" can also be used to enable or disable barriers, for consistency with other `ext4` mount options.

**inode\_readahead\_blks=n** This tuning parameter controls the maximum number of inode table blocks that `ext4`'s inode table readahead algorithm will pre-read into the buffer cache. The default value is 32 blocks.

**nouser\_xattr** Disables Extended User Attributes. See the `attr(5)` manual page for more information about extended attributes.

**noacl** This option disables POSIX Access Control List support. If ACL support is enabled in the kernel configuration (`CONFIG_EXT4_FS_POSIX_ACL`), ACL is enabled by default on mount. See the `acl(5)` manual page for more information about `acl`.

- bsddf (\*)** Make 'df' act like BSD.
- minixdf** Make 'df' act like Minix.
- debug** Extra debugging information is sent to syslog.
- abort** Simulate the effects of calling `ext4_abort()` for debugging purposes. This is normally used while remounting a filesystem which is already mounted.
- errors=remount-ro** Remount the filesystem read-only on an error.
- errors=continue** Keep going on a filesystem error.
- errors=panic** Panic and halt the machine if an error occurs. (These mount options override the errors behavior specified in the superblock, which can be configured using `tune2fs`)
- data\_err=ignore(\*)** Just print an error message if an error occurs in a file data buffer in ordered mode.
- data\_err=abort** Abort the journal if an error occurs in a file data buffer in ordered mode.
- grpuid | bsdgroups** New objects have the group ID of their parent.
- nogrpuid (\*) | sysvgroups** New objects have the group ID of their creator.
- resgid=n** The group ID which may use the reserved blocks.
- resuid=n** The user ID which may use the reserved blocks.
- sb=** Use alternate superblock at this location.
- quota, noquota, grpquota, usrquota** These options are ignored by the filesystem. They are used only by quota tools to recognize volumes where quota should be turned on. See documentation in the quota-tools package for more details (<http://sourceforge.net/projects/linuxquota>).
- jqfmt=<quota type>, usrjquota=<file>, grpjquota=<file>** These options tell filesystem details about quota so that quota information can be properly updated during journal replay. They replace the above quota options. See documentation in the quota-tools package for more details (<http://sourceforge.net/projects/linuxquota>).
- stripe=n** Number of filesystem blocks that `mballoc` will try to use for allocation size and alignment. For RAID5/6 systems this should be the number of data disks \* RAID chunk size in file system blocks.
- delalloc (\*)** Defer block allocation until just before `ext4` writes out the block(s) in question. This allows `ext4` to better allocation decisions more efficiently.
- nodelalloc** Disable delayed allocation. Blocks are allocated when the data is copied from userspace to the page cache, either via the `write(2)` system call or when an `mmap`'ed page which was previously unallocated is written for the first time.

**max\_batch\_time=usec** Maximum amount of time ext4 should wait for additional filesystem operations to be batch together with a synchronous write operation. Since a synchronous write operation is going to force a commit and then a wait for the I/O complete, it doesn't cost much, and can be a huge throughput win, we wait for a small amount of time to see if any other transactions can piggyback on the synchronous write. The algorithm used is designed to automatically tune for the speed of the disk, by measuring the amount of time (on average) that it takes to finish committing a transaction. Call this time the "commit time". If the time that the transaction has been running is less than the commit time, ext4 will try sleeping for the commit time to see if other operations will join the transaction. The commit time is capped by the `max_batch_time`, which defaults to 15000us (15ms). This optimization can be turned off entirely by setting `max_batch_time` to 0.

**min\_batch\_time=usec** This parameter sets the commit time (as described above) to be at least `min_batch_time`. It defaults to zero microseconds. Increasing this parameter may improve the throughput of multi-threaded, synchronous workloads on very fast disks, at the cost of increasing latency.

**journal\_ioprio=prio** The I/O priority (from 0 to 7, where 0 is the highest priority) which should be used for I/O operations submitted by `kjournald2` during a commit operation. This defaults to 3, which is a slightly higher priority than the default I/O priority.

**auto\_da\_alloc(\*), noauto\_da\_alloc** Many broken applications don't use `fsync()` when replacing existing files via patterns such as `fd = open("foo.new")/write(fd,..)/close(fd)/rename("foo.new", "foo")`, or worse yet, `fd = open("foo", O_TRUNC)/write(fd,..)/close(fd)`. If `auto_da_alloc` is enabled, ext4 will detect the replace-via-rename and replace-via-truncate patterns and force that any delayed allocation blocks are allocated such that at the next journal commit, in the default `data=ordered` mode, the data blocks of the new file are forced to disk before the `rename()` operation is committed. This provides roughly the same level of guarantees as ext3, and avoids the "zero-length" problem that can happen when a system crashes before the delayed allocation blocks are forced to disk.

**noinit\_itable** Do not initialize any uninitialized inode table blocks in the background. This feature may be used by installation CD's so that the install process can complete as quickly as possible; the inode table initialization process would then be deferred until the next time the file system is unmounted.

**init\_itable=n** The lazy itable init code will wait `n` times the number of milliseconds it took to zero out the previous block group's inode table. This minimizes the impact on the system performance while file system's inode table is being initialized.

**discard, nodiscard(\*)** Controls whether ext4 should issue `discard/TRIM` commands to the underlying block device when blocks are freed. This is useful for SSD devices and sparse/thinly-

provisioned LUNs, but it is off by default until sufficient testing has been done.

**noid32** Disables 32-bit UIDs and GIDs. This is for interoperability with older kernels which only store and expect 16-bit values.

**block\_validity(\*), noblock\_validity** These options enable or disable the in-kernel facility for tracking filesystem metadata blocks within internal data structures. This allows multi-block allocator and other routines to notice bugs or corrupted allocation bitmaps which cause blocks to be allocated which overlap with filesystem metadata blocks.

**dioread\_lock, dioread\_nolock** Controls whether or not ext4 should use the DIO read locking. If the `dioread_nolock` option is specified ext4 will allocate uninitialized extent before buffer write and convert the extent to initialized after IO completes. This approach allows ext4 code to avoid using inode mutex, which improves scalability on high speed storages. However this does not work with data journaling and `dioread_nolock` option will be ignored with kernel warning. Note that `dioread_nolock` code path is only used for extent-based files. Because of the restrictions this options comprises it is off by default (e.g. `dioread_lock`).

**max\_dir\_size\_kb=n** This limits the size of directories so that any attempt to expand them beyond the specified limit in kilobytes will cause an ENOSPC error. This is useful in memory constrained environments, where a very large directory can cause severe performance problems or even provoke the Out Of Memory killer. (For example, if there is only 512mb memory available, a 176mb directory may seriously cramp the system's style.)

**i\_version** Enable 64-bit inode version support. This option is off by default.

**dax** Use direct access (no page cache). See Documentation/filesystems/dax.txt. Note that this option is incompatible with `data=journal`.

## 37.5 Data Mode

There are 3 different data modes:

- writeback mode

In `data=writeback` mode, ext4 does not journal data at all. This mode provides a similar level of journaling as that of XFS, JFS, and ReiserFS in its default mode - metadata journaling. A crash+recovery can cause incorrect data to appear in files which were written shortly before the crash. This mode will typically provide the best ext4 performance.

- ordered mode

In `data=ordered` mode, ext4 only officially journals metadata, but it logically groups metadata information related to data changes with the data blocks

into a single unit called a transaction. When it's time to write the new metadata out to disk, the associated data blocks are written first. In general, this mode performs slightly slower than writeback but significantly faster than journal mode.

- journal mode

data=journal mode provides full data and metadata journaling. All new data is written to the journal first, and then to its final location. In the event of a crash, the journal can be replayed, bringing both data and metadata into a consistent state. This mode is the slowest except when data needs to be read from and written to disk at the same time where it outperforms all others modes. Enabling this mode will disable delayed allocation and O\_DIRECT support.

### 37.6 /proc entries

Information about mounted ext4 file systems can be found in /proc/fs/ext4. Each mounted filesystem will have a directory in /proc/fs/ext4 based on its device name (i.e., /proc/fs/ext4/hdc or /proc/fs/ext4/dm-0). The files in each per-device directory are shown in table below.

Files in /proc/fs/ext4/<devname>

**mb\_groups** details of multiblock allocator buddy cache of free blocks

### 37.7 /sys entries

Information about mounted ext4 file systems can be found in /sys/fs/ext4. Each mounted filesystem will have a directory in /sys/fs/ext4 based on its device name (i.e., /sys/fs/ext4/hdc or /sys/fs/ext4/dm-0). The files in each per-device directory are shown in table below.

Files in /sys/fs/ext4/<devname>:

(see also Documentation/ABI/testing/sysfs-fs-ext4)

**delayed\_allocation\_blocks** This file is read-only and shows the number of blocks that are dirty in the page cache, but which do not have their location in the filesystem allocated yet.

**inode\_goal** Tuning parameter which (if non-zero) controls the goal inode used by the inode allocator in preference to all other allocation heuristics. This is intended for debugging use only, and should be 0 on production systems.

**inode\_readahead\_blks** Tuning parameter which controls the maximum number of inode table blocks that ext4's inode table readahead algorithm will pre-read into the buffer cache.

**lifetime\_write\_kbytes** This file is read-only and shows the number of kilobytes of data that have been written to this filesystem since it was created.

**max\_writeback\_mb\_bump** The maximum number of megabytes the writeback code will try to write out before move on to another inode.

**mb\_group\_prealloc** The multiblock allocator will round up allocation requests to a multiple of this tuning parameter if the stripe size is not set in the ext4 superblock

**mb\_max\_to\_scan** The maximum number of extents the multiblock allocator will search to find the best extent.

**mb\_min\_to\_scan** The minimum number of extents the multiblock allocator will search to find the best extent.

**mb\_order2\_req** Tuning parameter which controls the minimum size for requests (as a power of 2) where the buddy cache is used.

**mb\_stats** Controls whether the multiblock allocator should collect statistics, which are shown during the unmount. 1 means to collect statistics, 0 means not to collect statistics.

**mb\_stream\_req** Files which have fewer blocks than this tunable parameter will have their blocks allocated out of a block group specific preallocation pool, so that small files are packed closely together. Each large file will have its blocks allocated out of its own unique preallocation pool.

**session\_write\_kbytes** This file is read-only and shows the number of kilobytes of data that have been written to this filesystem since it was mounted.

**reserved\_clusters** This is RW file and contains number of reserved clusters in the file system which will be used in the specific situations to avoid costly zeroout, unexpected ENOSPC, or possible data loss. The default is 2% or 4096 clusters, whichever is smaller and this can be changed however it can never exceed number of clusters in the file system. If there is not enough space for the reserved space when mounting the file mount will `_not_ fail`.

## 37.8 ioctls

There is some Ext4 specific functionality which can be accessed by applications through the system call interfaces. The list of all Ext4 specific ioctls are shown in the table below.

Table of Ext4 specific ioctls

**EXT4\_IOC\_GETFLAGS** Get additional attributes associated with inode. The ioctl argument is an integer bitfield, with bit values described in `ext4.h`. This ioctl is an alias for `FS_IOC_GETFLAGS`.

**EXT4\_IOC\_SETFLAGS** Set additional attributes associated with inode. The ioctl argument is an integer bitfield, with bit values described in `ext4.h`. This ioctl is an alias for `FS_IOC_SETFLAGS`.

**EXT4\_IOC\_GETVERSION, EXT4\_IOC\_GETVERSION\_OLD** Get the inode `i_generation` number stored for each inode. The `i_generation`

number is normally changed only when new inode is created and it is particularly useful for network filesystems. The ‘\_OLD’ version of this ioctl is an alias for FS\_IOC\_GETVERSION.

**EXT4\_IOC\_SETVERSION, EXT4\_IOC\_SETVERSION\_OLD** Set the inode `i_generation` number stored for each inode. The ‘\_OLD’ version of this ioctl is an alias for FS\_IOC\_SETVERSION.

**EXT4\_IOC\_GROUP\_EXTEND** This ioctl has the same purpose as the `resize mount` option. It allows to resize filesystem to the end of the last existing block group, further resize has to be done with `resize2fs`, either online, or offline. The argument points to the unsigned `logn` number representing the filesystem new block count.

**EXT4\_IOC\_MOVE\_EXT** Move the block extents from `orig_fd` (the one this ioctl is pointing to) to the `donor_fd` (the one specified in `move_extent` structure passed as an argument to this ioctl). Then, exchange inode metadata between `orig_fd` and `donor_fd`. This is especially useful for online defragmentation, because the allocator has the opportunity to allocate moved blocks better, ideally into one contiguous extent.

**EXT4\_IOC\_GROUP\_ADD** Add a new group descriptor to an existing or new group descriptor block. The new group descriptor is described by `ext4_new_group_input` structure, which is passed as an argument to this ioctl. This is especially useful in conjunction with `EXT4_IOC_GROUP_EXTEND`, which allows online resize of the filesystem to the end of the last existing block group. Those two ioctls combined is used in userspace online resize tool (e.g. `resize2fs`).

**EXT4\_IOC\_MIGRATE** This ioctl operates on the filesystem itself. It converts (migrates) `ext3` indirect block mapped inode to `ext4` extent mapped inode by walking through indirect block mapping of the original inode and converting contiguous block ranges into `ext4` extents of the temporary inode. Then, inodes are swapped. This ioctl might help, when migrating from `ext3` to `ext4` filesystem, however suggestion is to create fresh `ext4` filesystem and copy data from the backup. Note, that filesystem has to support extents for this ioctl to work.

**EXT4\_IOC\_ALLOC\_DA\_BLKs** Force all of the delay allocated blocks to be allocated to preserve application-expected `ext3` behaviour. Note that this will also start triggering a write of the data blocks, but this behaviour may change in the future as it is not necessary and has been done this way only for sake of simplicity.

**EXT4\_IOC\_RESIZE\_FS** Resize the filesystem to a new size. The number of blocks of resized filesystem is passed in via 64 bit integer argument. The kernel allocates bitmaps and inode table, the userspace tool thus just passes the new number of blocks.

**EXT4\_IOC\_SWAP\_BOOT** Swap `i_blocks` and associated attributes (like `i_size`, `i_flags`, ...) from the specified inode with inode `EXT4_BOOT_LOADER_INO` (#5). This is typically used to store a

boot loader in a secure part of the filesystem, where it can't be changed by a normal user by accident. The data blocks of the previous boot loader will be associated with the given inode.

## 37.9 References

**kernel source:** [<file:fs/ext4/>](#) [<file:fs/jbd2/>](#)

programs: <http://e2fsprogs.sourceforge.net/>

**useful links:** <http://fedoraproject.org/wiki/ext3-devel> <http://www.bullopen-source.org/ext4/> [http://ext4.wiki.kernel.org/index.php/Main\\_Page](http://ext4.wiki.kernel.org/index.php/Main_Page)  
<http://fedoraproject.org/wiki/Features/Ext4>



## **38.1 NFS Client**

### **38.1.1 The NFS client**

The NFS version 2 protocol was first documented in RFC1094 (March 1989). Since then two more major releases of NFS have been published, with NFSv3 being documented in RFC1813 (June 1995), and NFSv4 in RFC3530 (April 2003).

The Linux NFS client currently supports all the above published versions, and work is in progress on adding support for minor version 1 of the NFSv4 protocol.

The purpose of this document is to provide information on some of the special features of the NFS client that can be configured by system administrators.

### **38.1.2 The `nfs4_unique_id` parameter**

NFSv4 requires clients to identify themselves to servers with a unique string. File open and lock state shared between one client and one server is associated with this identity. To support robust NFSv4 state recovery and transparent state migration, this identity string must not change across client reboots.

Without any other intervention, the Linux client uses a string that contains the local system's node name. System administrators, however, often do not take care to ensure that node names are fully qualified and do not change over the lifetime of a client system. Node names can have other administrative requirements that require particular behavior that does not work well as part of an `nfs_client_id4` string.

The `nfs.nfs4_unique_id` boot parameter specifies a unique string that can be used instead of a system's node name when an NFS client identifies itself to a server. Thus, if the system's node name is not unique, or it changes, its `nfs.nfs4_unique_id` stays the same, preventing collision with other clients or loss of state during NFS reboot recovery or transparent state migration.

The `nfs.nfs4_unique_id` string is typically a UUID, though it can contain anything that is believed to be unique across all NFS clients. An `nfs4_unique_id` string should be chosen when a client system is installed, just as a system's root file system gets a fresh UUID in its label at install time.

The string should remain fixed for the lifetime of the client. It can be changed safely if care is taken that the client shuts down cleanly and all outstanding NFSv4 state has expired, to prevent loss of NFSv4 state.

This string can be stored in an NFS client's `grub.conf`, or it can be provided via a net boot facility such as PXE. It may also be specified as an `nfs.ko` module parameter. Specifying a unifier string is not support for NFS clients running in containers.

### 38.1.3 The DNS resolver

NFSv4 allows for one server to refer the NFS client to data that has been migrated onto another server by means of the special “`fs_locations`” attribute. See [RFC3530 Section 6: Filesystem Migration and Replication and Implementation Guide for Referrals in NFSv4](#).

The `fs_locations` information can take the form of either an ip address and a path, or a DNS hostname and a path. The latter requires the NFS client to do a DNS lookup in order to mount the new volume, and hence the need for an upcall to allow userland to provide this service.

Assuming that the user has the ‘`rpc_pipefs`’ filesystem mounted in the usual `/var/lib/nfs/rpc_pipefs`, the upcall consists of the following steps:

- (1) The process checks the `dns_resolve` cache to see if it contains a valid entry. If so, it returns that entry and exits.
- (2) If no valid entry exists, the helper script ‘`/sbin/nfs_cache_getent`’ (may be changed using the ‘`nfs.cache_getent`’ kernel boot parameter) is run, with two arguments: - the cache name, “`dns_resolve`” - the hostname to resolve
- (3) After looking up the corresponding ip address, the helper script writes the result into the `rpc_pipefs` pseudo-file ‘`/var/lib/nfs/rpc_pipefs/cache/dns_resolve/channel`’ in the following (text) format:

```
“<ip address> <hostname> <ttd>n”
```

Where `<ip address>` is in the usual IPv4 (123.456.78.90) or IPv6 (fee:ddcc:bbaa:9988:7766:5544:3322:1100, fee::1100, ...) format. `<hostname>` is identical to the second argument of the helper script, and `<ttd>` is the ‘time to live’ of this cache entry (in units of seconds).

---

**Note:** If `<ip address>` is invalid, say the string “0”, then a negative entry is created, which will cause the kernel to treat the hostname as having no valid DNS translation.

---

### 38.1.4 A basic sample /sbin/nfs\_cache\_getent

```
#!/bin/bash
#
ttl=600
#
cut=/usr/bin/cut
getent=/usr/bin/getent
rpc_pipefs=/var/lib/nfs/rpc_pipefs
#
die()
{
    echo "Usage: $0 cache_name entry_name"
    exit 1
}

[ $# -lt 2 ] && die
cachename="$1"
cache_path=${rpc_pipefs}/cache/${cachename}/channel

case "${cachename}" in
    dns_resolve)
        name="$2"
        result="$(getent hosts ${name} | cut -f1 -d\ )"
        [ -z "${result}" ] && result="0"
        ;;
    *)
        die
        ;;
esac
echo "${result} ${name} ${ttl}" >${cache_path}
```

## 38.2 Mounting the root filesystem via NFS (nfsroot)

**Authors** Written 1996 by Gero Kuhlmann <gero@gkminix.han.de>

Updated 1997 by Martin Mares <mj@atrey.karlin.mff.cuni.cz>

Updated 2006 by Nico Schottelius <nico-kernel-nfsroot@schottelius.org>

Updated 2006 by Horms <horms@verge.net.au>

Updated 2018 by Chris Novakovic <chris@chrisn.me.uk>

In order to use a diskless system, such as an X-terminal or printer server for example, it is necessary for the root filesystem to be present on a non-disk device. This may be an initramfs (see Documentation/filesystems/ramfs-rootfs-initramfs.rst), a ramdisk (see Documentation/admin-guide/initrd.rst) or a filesystem mounted via NFS. The following text describes on how to use NFS for the root filesystem. For the rest of this text ‘client’ means the diskless system, and ‘server’ means the NFS server.

### 38.2.1 Enabling nfsroot capabilities

In order to use nfsroot, NFS client support needs to be selected as built-in during configuration. Once this has been selected, the nfsroot option will become available, which should also be selected.

In the networking options, kernel level autoconfiguration can be selected, along with the types of autoconfiguration to support. Selecting all of DHCP, BOOTP and RARP is safe.

### 38.2.2 Kernel command line

When the kernel has been loaded by a boot loader (see below) it needs to be told what root fs device to use. And in the case of nfsroot, where to find both the server and the name of the directory on the server to mount as root. This can be established using the following kernel command line parameters:

**root=/dev/nfs** This is necessary to enable the pseudo-NFS-device. Note that it's not a real device but just a synonym to tell the kernel to use NFS instead of a real device.

**nfsroot=[<server-ip>:]<root-dir>[,<nfs-options>]** If the nfsroot' parameter is NOT given on the command line, the default `"/tftpboot/%s"` will be used.

**<server-ip> Specifies the IP address of the NFS server.** The default address is determined by the ip parameter (see below). This parameter allows the use of different servers for IP autoconfiguration and NFS.

**<root-dir> Name of the directory on the server to mount as root.** If there is a `"%s"` token in the string, it will be replaced by the ASCII-representation of the client's IP address.

**<nfs-options> Standard NFS options. All options are separated by commas.** The following defaults are used:

port	= as given by server portmap daemon
rsize	= 4096
wsizer	= 4096
timeo	= 7
retrans	= 3
acregmin	= 3
acregmax	= 60
acdirmin	= 30
acdirmax	= 60
flags	= hard, nointr, noposix, cto, ac

**ip=<client-ip>:<server-ip>:<gw-ip>:<netmask>:<hostname>:<device>:<autoconf>:**

This parameter tells the kernel how to configure IP addresses of devices and also how to set up the IP routing table. It was originally called nfsaddr, but now the boot-time IP configuration works independently of NFS, so it was renamed to ip and the old name remained as an alias for compatibility reasons.

If this parameter is missing from the kernel command line, all fields are assumed to be empty, and the defaults mentioned below apply. In general this means that the kernel tries to configure everything using autoconfiguration.

The `<autoconf>` parameter can appear alone as the value to the `ip` parameter (without all the `‘.’` characters before). If the value is `“ip=off”` or `“ip=none”`, no autoconfiguration will take place, otherwise autoconfiguration will take place. The most common way to use this is `“ip=dhcp”`.

**<client-ip> IP address of the client.** Default: Determined using autoconfiguration.

**<server-ip> IP address of the NFS server.** If RARP is used to determine the client address and this parameter is NOT empty only replies from the specified server are accepted.

Only required for NFS root. That is autoconfiguration will not be triggered if it is missing and NFS root is not in operation.

Value is exported to `/proc/net/pnp` with the prefix `“bootserver”` (see below).

Default: Determined using autoconfiguration. The address of the autoconfiguration server is used.

**<gw-ip> IP address of a gateway if the server is on a different subnet.**

Default: Determined using autoconfiguration.

**<netmask> Netmask for local network interface.** If unspecified the netmask is derived from the client IP address assuming classful addressing.

Default: Determined using autoconfiguration.

**<hostname> Name of the client.** If a `‘.’` character is present, anything before the first `‘.’` is used as the client’s hostname, and anything after it is used as its NIS domain name. May be supplied by autoconfiguration, but its absence will not trigger autoconfiguration. If specified and DHCP is used, the user-provided hostname (and NIS domain name, if present) will be carried in the DHCP request; this may cause a DNS record to be created or updated for the client.

Default: Client IP address is used in ASCII notation.

**<device> Name of network device to use.** Default: If the host only has one device, it is used. Otherwise the device is determined using autoconfiguration. This is done by sending autoconfiguration requests out of all devices, and using the device that received the first reply.

**<autoconf> Method to use for autoconfiguration.** In the case of options which specify multiple autoconfiguration protocols, requests are sent using all protocols, and the first one to reply is used.

Only autoconfiguration protocols that have been compiled into the kernel will be used, regardless of the value of this option:

<code>off</code> or <code>none</code> :	<code>don't use autoconfiguration</code> <code>(do static IP assignment instead)</code>
<code>on</code> or <code>any</code> :	<code>use any protocol available in the kernel</code> <code>(default)</code>
<code>dhcp</code> :	<code>use DHCP</code>
<code>bootp</code> :	<code>use BOOTP</code>
<code>rarp</code> :	<code>use RARP</code>

(continues on next page)

(continued from previous page)

both:	use both BOOTP and RARP but not DHCP (old option kept for backwards compatibility)
-------	---

if dhcp is used, the client identifier can be used by following format  
“ip=dhcp,client-id-type,client-id-value”

Default: any

**<dns0-ip> IP address of primary nameserver.** Value is exported to /proc/net/pnp with the prefix “nameserver ” (see below).

Default: None if not using autoconfiguration; determined automatically if using autoconfiguration.

**<dns1-ip> IP address of secondary nameserver.** See <dns0-ip>.

**<ntp0-ip> IP address of a Network Time Protocol (NTP) server.** Value is exported to /proc/net/ipconfig/ntp\_servers, but is otherwise unused (see below).

Default: None if not using autoconfiguration; determined automatically if using autoconfiguration.

After configuration (whether manual or automatic) is complete, two files are created in the following format; lines are omitted if their respective value is empty following configuration:

- /proc/net/pnp:
  - #PROTO: <DHCP|BOOTP|RARP|MANUAL> (depending on configuration method) domain <dns-domain> (if autoconfigured, the DNS domain) nameserver <dns0-ip> (primary name server IP) nameserver <dns1-ip> (secondary name server IP) nameserver <dns2-ip> (tertiary name server IP) bootserver <server-ip> (NFS server IP)
- /proc/net/ipconfig/ntp\_servers:
  - <ntp0-ip> (NTP server IP) <ntp1-ip> (NTP server IP) <ntp2-ip> (NTP server IP)

<dns-domain> and <dns2-ip> (in /proc/net/pnp) and <ntp1-ip> and <ntp2-ip> (in /proc/net/ipconfig/ntp\_servers) are requested during autoconfiguration; they cannot be specified as part of the “ip=” kernel command line parameter.

Because the “domain” and “nameserver” options are recognised by DNS resolvers, /etc/resolv.conf is often linked to /proc/net/pnp on systems that use an NFS root filesystem.

Note that the kernel will not synchronise the system time with any NTP servers it discovers; this is the responsibility of a user space process (e.g. an initrd/initramfs script that passes the IP addresses listed in /proc/net/ipconfig/ntp\_servers to an NTP client before mounting the real root filesystem if it is on NFS).

**nfsrootdebug** This parameter enables debugging messages to appear in the kernel log at boot time so that administrators can verify that the correct NFS mount options, server address, and root path are passed to the NFS client.

**rdinit=<executable file>** To specify which file contains the program that starts system initialization, administrators can use this command line parameter. The default value of this parameter is `"/init"`. If the specified file exists and the kernel can execute it, root filesystem related kernel command line parameters, including `'nfsroot='`, are ignored.

A description of the process of mounting the root file system can be found in [Documentation/driver-api/early-userspace/early\\_userspace\\_support.rst](#)

### 38.2.3 Boot Loader

To get the kernel into memory different approaches can be used. They depend on various facilities being available:

- Booting from a floppy using syslinux

When building kernels, an easy way to create a boot floppy that uses syslinux is to use the `zdisk` or `bzdisk` make targets which use `zimage` and `bzimage` images respectively. Both targets accept the `FDARGS` parameter which can be used to set the kernel command line.

e.g:

```
make bzdisk FDARGS="root=/dev/nfs"
```

Note that the user running this command will need to have access to the floppy drive device, `/dev/fd0`

For more information on syslinux, including how to create bootdisks for prebuilt kernels, see <http://syslinux.zytor.com/>

---

**Note:** Previously it was possible to write a kernel directly to a floppy using `dd`, configure the boot device using `rdev`, and boot using the resulting floppy. Linux no longer supports this method of booting.

---

- Booting from a cdrom using isolinux

When building kernels, an easy way to create a bootable cdrom that uses isolinux is to use the `isoimage` target which uses a `bzimage` image. Like `zdisk` and `bzdisk`, this target accepts the `FDARGS` parameter which can be used to set the kernel command line.

e.g:

```
make isoimage FDARGS="root=/dev/nfs"
```

The resulting iso image will be `arch/<ARCH>/boot/image.iso` This can be written to a cdrom using a variety of tools including `cdrecord`.

e.g:

```
cdrecord dev=ATAPI:1,0,0 arch/x86/boot/image.iso
```

For more information on isolinux, including how to create bootdisks for prebuilt kernels, see <http://syslinux.zytor.com/>

- Using LILO

When using LILO all the necessary command line parameters may be specified using the ‘append=’ directive in the LILO configuration file.

However, to use the ‘root=’ directive you also need to create a dummy root device, which may be removed after LILO is run.

e.g:

```
mknod /dev/boot255 c 0 255
```

For information on configuring LILO, please refer to its documentation.

- Using GRUB

When using GRUB, kernel parameter are simply appended after the kernel specification: kernel <kernel> <parameters>

- Using loadlin

loadlin may be used to boot Linux from a DOS command prompt without requiring a local hard disk to mount as root. This has not been thoroughly tested by the authors of this document, but in general it should be possible configure the kernel command line similarly to the configuration of LILO.

Please refer to the loadlin documentation for further information.

- Using a boot ROM

This is probably the most elegant way of booting a diskless client. With a boot ROM the kernel is loaded using the TFTP protocol. The authors of this document are not aware of any no commercial boot ROMs that support booting Linux over the network. However, there are two free implementations of a boot ROM, netboot-nfs and etherboot, both of which are available on [sunsite.unc.edu](http://sunsite.unc.edu), and both of which contain everything you need to boot a diskless Linux client.

- Using pxelinux

Pxelinux may be used to boot linux using the PXE boot loader which is present on many modern network cards.

When using pxelinux, the kernel image is specified using “kernel <relative-path-below /tftpboot>”. The nfsroot parameters are passed to the kernel by adding them to the “append” line. It is common to use serial console in conjunction with pxelinux, see [Documentation/admin-guide/serial-console.rst](#) for more information.

For more information on isolinux, including how to create bootdisks for prebuilt kernels, see <http://syslinux.zytor.com/>

### 38.2.4 Credits

The nfsroot code in the kernel and the RARP support have been written by Gero Kuhlmann <gero@gkminix.han.de>.

The rest of the IP layer autoconfiguration code has been written by Martin Mares <mj@atrey.karlin.mff.cuni.cz>.

In order to write the initial version of nfsroot I would like to thank Jens-Uwe Mager <jum@anubis.han.de> for his help.

## 38.3 Setting up NFS/RDMA

**Author** NetApp and Open Grid Computing (May 29, 2008)

<b>Warning:</b> This document is probably obsolete.
---

### 38.3.1 Overview

This document describes how to install and setup the Linux NFS/RDMA client and server software.

The NFS/RDMA client was first included in Linux 2.6.24. The NFS/RDMA server was first included in the following release, Linux 2.6.25.

In our testing, we have obtained excellent performance results (full 10Gbit wire bandwidth at minimal client CPU) under many workloads. The code passes the full Connectathon test suite and operates over both Infiniband and iWARP RDMA adapters.

### 38.3.2 Getting Help

If you get stuck, you can ask questions on the [nfs-rdma-devel@lists.sourceforge.net](mailto:nfs-rdma-devel@lists.sourceforge.net) mailing list.

### 38.3.3 Installation

These instructions are a step by step guide to building a machine for use with NFS/RDMA.

- Install an RDMA device

Any device supported by the drivers in drivers/infiniband/hw is acceptable.

Testing has been performed using several Mellanox-based IB cards, the Ammasso AMS1100 iWARP adapter, and the Chelsio cxgb3 iWARP adapter.

- Install a Linux distribution and tools

The first kernel release to contain both the NFS/RDMA client and server was Linux 2.6.25. Therefore, a distribution compatible with this and subsequent Linux kernel release should be installed.

The procedures described in this document have been tested with distributions from Red Hat's Fedora Project (<http://fedora.redhat.com/>).

- Install `nfs-utils-1.1.2` or greater on the client

An NFS/RDMA mount point can be obtained by using the `mount.nfs` command in `nfs-utils-1.1.2` or greater (`nfs-utils-1.1.1` was the first `nfs-utils` version with support for NFS/RDMA mounts, but for various reasons we recommend using `nfs-utils-1.1.2` or greater). To see which version of `mount.nfs` you are using, type:

```
$ /sbin/mount.nfs -V
```

If the version is less than 1.1.2 or the command does not exist, you should install the latest version of `nfs-utils`.

Download the latest package from: <http://www.kernel.org/pub/linux/utils/nfs>

Uncompress the package and follow the installation instructions.

If you will not need the `idmapper` and `gssd` executables (you do not need these to create an NFS/RDMA enabled mount command), the installation process can be simplified by disabling these features when running `configure`:

```
$ ./configure --disable-gss --disable-nfsv4
```

To build `nfs-utils` you will need the `tcp_wrappers` package installed. For more information on this see the package's README and INSTALL files.

After building the `nfs-utils` package, there will be a `mount.nfs` binary in the `utils/mount` directory. This binary can be used to initiate NFS v2, v3, or v4 mounts. To initiate a v4 mount, the binary must be called `mount.nfs4`. The standard technique is to create a symlink called `mount.nfs4` to `mount.nfs`.

This `mount.nfs` binary should be installed at `/sbin/mount.nfs` as follows:

```
$ sudo cp utils/mount/mount.nfs /sbin/mount.nfs
```

In this location, `mount.nfs` will be invoked automatically for NFS mounts by the system mount command.

---

**Note:** `mount.nfs` and therefore `nfs-utils-1.1.2` or greater is only needed on the NFS client machine. You do not need this specific version of `nfs-utils` on the server. Furthermore, only the `mount.nfs` command from `nfs-utils-1.1.2` is needed on the client.

---

- Install a Linux kernel with NFS/RDMA

The NFS/RDMA client and server are both included in the mainline Linux kernel version 2.6.25 and later. This and other versions of the Linux kernel can be found at: <https://www.kernel.org/pub/linux/kernel/>

Download the sources and place them in an appropriate location.

- Configure the RDMA stack

Make sure your kernel configuration has RDMA support enabled. Under Device Drivers -> InfiniBand support, update the kernel configuration to enable InfiniBand support [NOTE: the option name is misleading. Enabling InfiniBand support is required for all RDMA devices (IB, iWARP, etc.)].

Enable the appropriate IB HCA support (mlx4, mthca, ehca, ipath, etc.) or iWARP adapter support (amso, cxgb3, etc.).

If you are using InfiniBand, be sure to enable IP-over-InfiniBand support.

- Configure the NFS client and server

Your kernel configuration must also have NFS file system support and/or NFS server support enabled. These and other NFS related configuration options can be found under File Systems -> Network File Systems.

- Build, install, reboot

The NFS/RDMA code will be enabled automatically if NFS and RDMA are turned on. The NFS/RDMA client and server are configured via the hidden SUNRPC\_XPRT\_RDMA config option that depends on SUNRPC and INFINIBAND. The value of SUNRPC\_XPRT\_RDMA will be:

1. N if either SUNRPC or INFINIBAND are N, in this case the NFS/RDMA client and server will not be built
2. M if both SUNRPC and INFINIBAND are on (M or Y) and at least one is M, in this case the NFS/RDMA client and server will be built as modules
3. Y if both SUNRPC and INFINIBAND are Y, in this case the NFS/RDMA client and server will be built into the kernel

Therefore, if you have followed the steps above and turned no NFS and RDMA, the NFS/RDMA client and server will be built.

Build a new kernel, install it, boot it.

### 38.3.4 Check RDMA and NFS Setup

Before configuring the NFS/RDMA software, it is a good idea to test your new kernel to ensure that the kernel is working correctly. In particular, it is a good idea to verify that the RDMA stack is functioning as expected and standard NFS over TCP/IP and/or UDP/IP is working properly.

- Check RDMA Setup

If you built the RDMA components as modules, load them at this time. For example, if you are using a Mellanox Tavor/Sinai/Arbel card:

```
$ modprobe ib_mthca
$ modprobe ib_ipoib
```

If you are using InfiniBand, make sure there is a Subnet Manager (SM) running on the network. If your IB switch has an embedded SM, you can use it.

Otherwise, you will need to run an SM, such as OpenSM, on one of your end nodes.

If an SM is running on your network, you should see the following:

```
$ cat /sys/class/infiniband/driverX/ports/1/state
4: ACTIVE
```

where driverX is mthca0, ipath5, ehca3, etc.

To further test the InfiniBand software stack, use IPoIB (this assumes you have two IB hosts named host1 and host2):

```
host1$ ip link set dev ib0 up
host1$ ip address add dev ib0 a.b.c.x
host2$ ip link set dev ib0 up
host2$ ip address add dev ib0 a.b.c.y
host1$ ping a.b.c.y
host2$ ping a.b.c.x
```

For other device types, follow the appropriate procedures.

- Check NFS Setup

For the NFS components enabled above (client and/or server), test their functionality over standard Ethernet using TCP/IP or UDP/IP.

### 38.3.5 NFS/RDMA Setup

We recommend that you use two machines, one to act as the client and one to act as the server.

#### One time configuration:

- On the server system, configure the `/etc/exports` file and start the NFS/RDMA server.

Exports entries with the following formats have been tested:

```
/vol0 192.168.0.47(fsid=0,rw,async,insecure,no_root_squash)
/vol0 192.168.0.0/255.255.255.0(fsid=0,rw,async,insecure,no_root_
↪squash)
```

The IP address(es) is(are) the client's IPoIB address for an InfiniBand HCA or the client's iWARP address(es) for an RNIC.

---

**Note:** The “insecure” option must be used because the NFS/RDMA client does not use a reserved port.

---

**Each time a machine boots:**

- Load and configure the RDMA drivers

For InfiniBand using a Mellanox adapter:

```
$ modprobe ib_mthca
$ modprobe ib_ipoib
$ ip li set dev ib0 up
$ ip addr add dev ib0 a.b.c.d
```

---

**Note:** Please use unique addresses for the client and server!

---

- Start the NFS server

If the NFS/RDMA server was built as a module (CONFIG\_SUNRPC\_XPRT\_RDMA=m in kernel config), load the RDMA transport module:

```
$ modprobe svcrdma
```

Regardless of how the server was built (module or built-in), start the server:

```
$ /etc/init.d/nfs start
```

or

```
$ service nfs start
```

Instruct the server to listen on the RDMA transport:

```
$ echo rdma 20049 > /proc/fs/nfsd/portlist
```

- On the client system

If the NFS/RDMA client was built as a module (CONFIG\_SUNRPC\_XPRT\_RDMA=m in kernel config), load the RDMA client module:

```
$ modprobe xprtrdma.ko
```

Regardless of how the client was built (module or built-in), use this command to mount the NFS/RDMA server:

```
$ mount -o rdma,port=20049 <IPoIB-server-name-or-address>:<export> /
↪mnt
```

To verify that the mount is using RDMA, run “cat /proc/mounts” and check the “proto” field for the given mount.

Congratulations! You’ re using NFS/RDMA!

## 38.4 Administrative interfaces for nfsd

Note that normally these interfaces are used only by the utilities in nfs-utils.

nfsd is controlled mainly by pseudofiles under the “nfsd” filesystem, which is normally mounted at /proc/fs/nfsd/.

The server is always started by the first write of a nonzero value to nfsd/threads.

Before doing that, NFSD can be told which sockets to listen on by writing to nfsd/portlist; that write may be:

- an ascii-encoded file descriptor, which should refer to a bound (and listening, for tcp) socket, or
- “transportname port” , where transportname is currently either “udp” , “tcp” , or “rdma” .

If nfsd is started without doing any of these, then it will create one udp and one tcp listener at port 2049 (see nfsd\_init\_socks).

On startup, nfsd and lockd grace periods start. nfsd is shut down by a write of 0 to nfsd/threads. All locks and state are thrown away at that point.

Between startup and shutdown, the number of threads may be adjusted up or down by additional writes to nfsd/threads or by writes to nfsd/pool\_threads.

For more detail about files under nfsd/ and what they control, see fs/nfsd/nfsctl.c; most of them have detailed comments.

### 38.4.1 Implementation notes

Note that the rpc server requires the caller to serialize addition and removal of listening sockets, and startup and shutdown of the server. For nfsd this is done using nfsd\_mutex.

## 38.5 NFS ID Mapper

Id mapper is used by NFS to translate user and group ids into names, and to translate user and group names into ids. Part of this translation involves performing an upcall to userspace to request the information. There are two ways NFS could obtain this information: placing a call to /sbin/request-key or by placing a call to the rpc.idmap daemon.

NFS will attempt to call /sbin/request-key first. If this succeeds, the result will be cached using the generic request-key cache. This call should only fail if /etc/request-key.conf is not configured for the id\_resolver key type, see the “Configuring” section below if you wish to use the request-key method.

If the call to /sbin/request-key fails (if /etc/request-key.conf is not configured with the id\_resolver key type), then the idmapper will ask the legacy rpc.idmap daemon for the id mapping. This result will be stored in a custom NFS idmap cache.

### 38.5.1 Configuring

The file `/etc/request-key.conf` will need to be modified so `/sbin/request-key` can direct the upcall. The following line should be added:

```
#OP TYPE DESCRIPTION CALLOUT INFO PROGRAM ARG1 ARG2 ARG3
... #===== ===== ===== =====
===== create id_resolver * * /usr/sbin/
nfs.idmap %k %d 600
```

This will direct all `id_resolver` requests to the program `/usr/sbin/nfs.idmap`. The last parameter, 600, defines how many seconds into the future the key will expire. This parameter is optional for `/usr/sbin/nfs.idmap`. When the timeout is not specified, `nfs.idmap` will default to 600 seconds.

id mapper uses for key descriptions:

uid:	Find the UID for the given user
gid:	Find the GID for the given group
user:	Find the user name for the given UID
group:	Find the group name for the given GID

You can handle any of these individually, rather than using the generic upcall program. If you would like to use your own program for a uid lookup then you would edit your `request-key.conf` so it look similar to this:

```
#OP TYPE DESCRIPTION CALLOUT INFO PROGRAM ARG1 ARG2 ARG3
... #===== ===== ===== =====
===== create id_resolver uid:* * /some/
other/program %k %d 600 create id_resolver * * /usr/sbin/nfs.idmap %k
%d 600
```

Notice that the new line was added above the line for the generic program. `request-key` will find the first matching line and corresponding program. In this case, `/some/other/program` will handle all uid lookups and `/usr/sbin/nfs.idmap` will handle gid, user, and group lookups.

See [Documentation/security/keys/request-key.rst](#) for more information about the `request-key` function.

### 38.5.2 nfs.idmap

`nfs.idmap` is designed to be called by `request-key`, and should not be run “by hand”. This program takes two arguments, a serialized key and a key description. The serialized key is first converted into a `key_serial_t`, and then passed as an argument to `keyctl_instantiate` (both are part of `keyutils.h`).

The actual lookups are performed by functions found in `nfsidmap.h`. `nfs.idmap` determines the correct function to call by looking at the first part of the description string. For example, a uid lookup description will appear as “`uid:user@domain`”.

`nfs.idmap` will return 0 if the key was instantiated, and non-zero otherwise.

## 38.6 pNFS block layout server user guide

The Linux NFS server now supports the pNFS block layout extension. In this case the NFS server acts as Metadata Server (MDS) for pNFS, which in addition to handling all the metadata access to the NFS export also hands out layouts to the clients to directly access the underlying block devices that are shared with the client.

To use pNFS block layouts with with the Linux NFS server the exported file system needs to support the pNFS block layouts (currently just XFS), and the file system must sit on shared storage (typically iSCSI) that is accessible to the clients in addition to the MDS. As of now the file system needs to sit directly on the exported volume, striping or concatenation of volumes on the MDS and clients is not supported yet.

On the server, pNFS block volume support is automatically if the file system support it. On the client make sure the kernel has the CONFIG\_PNFS\_BLOCK option enabled, the blkmapd daemon from nfs-utils is running, and the file system is mounted using the NFSv4.1 protocol version (mount -o vers=4.1).

If the nfsd server needs to fence a non-responding client it calls /sbin/nfsd-recall-failed with the first argument set to the IP address of the client, and the second argument set to the device node without the /dev prefix for the file system to be fenced. Below is an example file that shows how to translate the device into a serial number from SCSI EVPD 0x80:

```
cat > /sbin/nfsd-recall-failed << EOF
```

```
#!/bin/sh

CLIENT="$1"
DEV="/dev/$2"
EVPD=`sg_inq --page=0x80 ${DEV} | \
    grep "Unit serial number:" | \
    awk -F ':' '{print $2}'`

echo "fencing client ${CLIENT} serial ${EVPD}" >> /var/log/pnfsd-fence.log
EOF
```

## 38.7 pNFS SCSI layout server user guide

This document describes support for pNFS SCSI layouts in the Linux NFS server. With pNFS SCSI layouts, the NFS server acts as Metadata Server (MDS) for pNFS, which in addition to handling all the metadata access to the NFS export, also hands out layouts to the clients so that they can directly access the underlying SCSI LUNs that are shared with the client.

To use pNFS SCSI layouts with with the Linux NFS server, the exported file system needs to support the pNFS SCSI layouts (currently just XFS), and the file system must sit on a SCSI LUN that is accessible to the clients in addition to the MDS. As of now the file system needs to sit directly on the exported LUN, striping or concatenation of LUNs on the MDS and clients is not supported yet.

On a server built with `CONFIG_NFSD_SCSI`, the pNFS SCSI volume support is automatically enabled if the file system is exported using the “pnfs” option and the underlying SCSI device support persistent reservations. On the client make sure the kernel has the `CONFIG_PNFS_BLOCK` option enabled, and the file system is mounted using the NFSv4.1 protocol version (`mount -o vers=4.1`).

## 38.8 NFS Fault Injection

Fault injection is a method for forcing errors that may not normally occur, or may be difficult to reproduce. Forcing these errors in a controlled environment can help the developer find and fix bugs before their code is shipped in a production system. Injecting an error on the Linux NFS server will allow us to observe how the client reacts and if it manages to recover its state correctly.

`NFSD_FAULT_INJECTION` must be selected when configuring the kernel to use this feature.

### 38.8.1 Using Fault Injection

On the client, mount the fault injection server through NFS v4.0+ and do some work over NFS (open files, take locks, ...).

On the server, mount the `debugfs` filesystem to `<debug_dir>` and `ls <debug_dir>/nfsd`. This will show a list of files that will be used for injecting faults on the NFS server. As root, write a number `n` to the file corresponding to the action you want the server to take. The server will then process the first `n` items it finds. So if you want to forget 5 locks, echo ‘5’ to `<debug_dir>/nfsd/forget_locks`. A value of 0 will tell the server to forget all corresponding items. A log message will be created containing the number of items forgotten (check `dmesg`).

Go back to work on the client and check if the client recovered from the error correctly.

### 38.8.2 Available Faults

**forget\_clients:** The NFS server keeps a list of clients that have placed a mount call. If this list is cleared, the server will have no knowledge of who the client is, forcing the client to reauthenticate with the server.

**forget\_openowners:** The NFS server keeps a list of what files are currently opened and who they were opened by. Clearing this list will force the client to reopen its files.

**forget\_locks:** The NFS server keeps a list of what files are currently locked in the VFS. Clearing this list will force the client to reclaim its locks (files are unlocked through the VFS as they are cleared from this list).

**forget\_delegations:** A delegation is used to assure the client that a file, or part of a file, has not changed since the delegation was awarded. Clearing this list will force the client to reacquire its delegation before accessing the file again.

**recall\_delegations:** Delegations can be recalled by the server when another client attempts to access a file. This test will notify the client that its delegation has been revoked, forcing the client to reacquire the delegation before using the file again.

### 38.8.3 tools/nfs/inject\_faults.sh script

This script has been created to ease the fault injection process. This script will detect the mounted debugfs directory and write to the files located there based on the arguments passed by the user. For example, running `inject_faults.sh forget_locks 1` as root will instruct the server to forget one lock. Running `inject_faults.sh forget_locks` will instruct the server to forget all locks.

## 39.1 GPIO Aggregator

The GPIO Aggregator provides a mechanism to aggregate GPIOs, and expose them as a new `gpio_chip`. This supports the following use cases.

### 39.1.1 Aggregating GPIOs using Sysfs

GPIO controllers are exported to userspace using `/dev/gpiochip*` character devices. Access control to these devices is provided by standard UNIX file system permissions, on an all-or-nothing basis: either a GPIO controller is accessible for a user, or it is not.

The GPIO Aggregator provides access control for a set of one or more GPIOs, by aggregating them into a new `gpio_chip`, which can be assigned to a group or user using standard UNIX file ownership and permissions. Furthermore, this simplifies and hardens exporting GPIOs to a virtual machine, as the VM can just grab the full GPIO controller, and no longer needs to care about which GPIOs to grab and which not, reducing the attack surface.

Aggregated GPIO controllers are instantiated and destroyed by writing to write-only attribute files in sysfs.

`/sys/bus/platform/drivers/gpio-aggregator/`

“**new\_device**” ... Userspace may ask the kernel to instantiate an aggregated GPIO controller by writing a string describing the GPIOs to aggregate to the “`new_device`” file, using the format

```
[<gpioA>] [<gpiochipB> <offsets>] ...
```

Where:

“**<gpioA>**” ... is a GPIO line name,

“**<gpiochipB>**” ... is a GPIO chip label, and

“**<offsets>**” ... is a comma-separated list of GPIO offsets and/or GPIO offset ranges denoted by dashes.

Example: Instantiate a new GPIO aggregator by aggregating GPIO line 19 of “e6052000.gpio” and GPIO lines 20-21 of “e6050000.gpio” into a new `gpio_chip`:

```
$ echo 'e6052000.gpio 19 e6050000.gpio 20-21' > new_
→device
```

“**delete\_device**” ... Userspace may ask the kernel to destroy an aggregated GPIO controller after use by writing its device name to the “`delete_device`” file.

Example: Destroy the previously-created aggregated GPIO controller, assumed to be “`gpio-aggregator.0`” :

```
$ echo gpio-aggregator.0 > delete_device
```

### 39.1.2 Generic GPIO Driver

The GPIO Aggregator can also be used as a generic driver for a simple GPIO-operated device described in DT, without a dedicated in-kernel driver. This is useful in industrial control, and is not unlike e.g. `spidev`, which allows the user to communicate with an SPI device from userspace.

Binding a device to the GPIO Aggregator is performed either by modifying the `gpio-aggregator` driver, or by writing to the “`driver_override`” file in Sysfs.

Example: If “`door`” is a GPIO-operated device described in DT, using its own compatible value:

```
door {
    compatible = "myvendor,mydoor";

    gpios = <&gpio2 19 GPIO_ACTIVE_HIGH>,
           <&gpio2 20 GPIO_ACTIVE_LOW>;
    gpio-line-names = "open", "lock";
};
```

it can be bound to the GPIO Aggregator by either:

1. Adding its compatible value to `gpio_aggregator_dt_ids[]`,
2. Binding manually using “`driver_override`” :

```
$ echo gpio-aggregator > /sys/bus/platform/devices/door/driver_override
$ echo door > /sys/bus/platform/drivers/gpio-aggregator/bind
```

After that, a new `gpiochip` “`door`” has been created:

```
$ gpioinfo door
gpiochip12 - 2 lines:
   line  0:      "open"         unused  input  active-high
   line  1:      "lock"         unused  input  active-high
```

## 39.2 GPIO Sysfs Interface for Userspace

**Warning:** THIS ABI IS DEPRECATED, THE ABI DOCUMENTATION HAS BEEN MOVED TO Documentation/ABI/obsolete/sysfs-gpio AND NEW USERSPACE CONSUMERS ARE SUPPOSED TO USE THE CHARACTER DEVICE ABI. THIS OLD SYSFS ABI WILL NOT BE DEVELOPED (NO NEW FEATURES), IT WILL JUST BE MAINTAINED.

Refer to the examples in tools/gpio/\* for an introduction to the new character device ABI. Also see the userspace header in include/uapi/linux/gpio.h

### 39.2.1 The deprecated sysfs ABI

Platforms which use the “gpiolib” implementors framework may choose to configure a sysfs user interface to GPIOs. This is different from the debugfs interface, since it provides control over GPIO direction and value instead of just showing a gpio state summary. Plus, it could be present on production systems without debugging support.

Given appropriate hardware documentation for the system, userspace could know for example that GPIO #23 controls the write protect line used to protect boot loader segments in flash memory. System upgrade procedures may need to temporarily remove that protection, first importing a GPIO, then changing its output state, then updating the code before re-enabling the write protection. In normal use, GPIO #23 would never be touched, and the kernel would have no need to know about it.

Again depending on appropriate hardware documentation, on some systems userspace GPIO can be used to determine system configuration data that standard kernels won't know about. And for some tasks, simple userspace GPIO drivers could be all that the system really needs.

DO NOT ABUSE SYSFS TO CONTROL HARDWARE THAT HAS PROPER KERNEL DRIVERS. PLEASE READ THE DOCUMENT AT Documentation/driver-api/gpio/drivers-on-gpio.rst TO AVOID REINVENTING KERNEL WHEELS IN USERSPACE. I MEAN IT. REALLY.

### 39.2.2 Paths in Sysfs

There are three kinds of entries in /sys/class/gpio:

- Control interfaces used to get userspace control over GPIOs;
- GPIOs themselves; and
- GPIO controllers ( “gpio\_chip” instances).

That's in addition to standard files including the “device” symlink.

The control interfaces are write-only:

```
/sys/class/gpio/
```

**“export”** ... Userspace may ask the kernel to export control of a GPIO to userspace by writing its number to this file.

Example: “echo 19 > export” will create a “gpio19” node for GPIO #19, if that’s not requested by kernel code.

**“unexport”** ... Reverses the effect of exporting to userspace.

Example: “echo 19 > unexport” will remove a “gpio19” node exported using the “export” file.

GPIO signals have paths like /sys/class/gpio/gpio42/ (for GPIO #42) and have the following read/write attributes:

/sys/class/gpio/gpioN/

**“direction”** ... reads as either “in” or “out” . This value may normally be written. Writing as “out” defaults to initializing the value as low. To ensure glitch free operation, values “low” and “high” may be written to configure the GPIO as an output with that initial value.

Note that this attribute will not exist if the kernel doesn’t support changing the direction of a GPIO, or it was exported by kernel code that didn’t explicitly allow userspace to re-configure this GPIO’s direction.

**“value”** ... reads as either 0 (low) or 1 (high). If the GPIO is configured as an output, this value may be written; any nonzero value is treated as high.

If the pin can be configured as interrupt-generating interrupt and if it has been configured to generate interrupts (see the description of “edge” ), you can poll(2) on that file and poll(2) will return whenever the interrupt was triggered. If you use poll(2), set the events POLLPRI and POLLERR. If you use select(2), set the file descriptor in exceptfds. After poll(2) returns, either lseek(2) to the beginning of the sysfs file and read the new value or close the file and re-open it to read the value.

**“edge”** ... reads as either “none” , “rising” , “falling” , or “both” . Write these strings to select the signal edge(s) that will make poll(2) on the “value” file return.

This file exists only if the pin can be configured as an interrupt generating input pin.

**“active\_low”** ... reads as either 0 (false) or 1 (true). Write any nonzero value to invert the value attribute both for reading and writing. Existing and subsequent poll(2) support configuration via the edge attribute for “rising” and “falling” edges will follow this setting.

GPIO controllers have paths like /sys/class/gpio/gpiochip42/ (for the controller implementing GPIOs starting at #42) and have the following read-only attributes:

/sys/class/gpio/gpiochipN/

“**base**” … same as N, the first GPIO managed by this chip

“**label**” … provided for diagnostics (not always unique)

“**ngpio**” … how many GPIOs this manages (N to N + ngpio - 1)

Board documentation should in most cases cover what GPIOs are used for what purposes. However, those numbers are not always stable; GPIOs on a daughter-card might be different depending on the base board being used, or other cards in the stack. In such cases, you may need to use the `gpiochip` nodes (possibly in conjunction with schematics) to determine the correct GPIO number to use for a given signal.

### 39.2.3 Exporting from Kernel code

Kernel code can explicitly manage exports of GPIOs which have already been requested using `gpio_request()`:

```
/* export the GPIO to userspace */
int gpiod_export(struct gpio_desc *desc, bool direction_may_change);

/* reverse gpio_export() */
void gpiod_unexport(struct gpio_desc *desc);

/* create a sysfs link to an exported GPIO node */
int gpiod_export_link(struct device *dev, const char *name,
                    struct gpio_desc *desc);
```

After a kernel driver requests a GPIO, it may only be made available in the `sysfs` interface by `gpiod_export()`. The driver can control whether the signal direction may change. This helps drivers prevent userspace code from accidentally clobbering important system state.

This explicit exporting can help with debugging (by making some kinds of experiments easier), or can provide an always-there interface that’s suitable for documenting as part of a board support package.

After the GPIO has been exported, `gpiod_export_link()` allows creating symlinks from elsewhere in `sysfs` to the GPIO `sysfs` node. Drivers can use this to provide the interface under their own device in `sysfs` with a descriptive name.



## NOTES ON THE CHANGE FROM 16-BIT UIDS TO 32-BIT UIDS

**Author** Chris Wing <wingc@umich.edu>

**Last updated** January 11, 2000

- kernel code **MUST** take into account `__kernel_uid_t` and `__kernel_uid32_t` when communicating between user and kernel space in an ioctl or data structure.
- kernel code should use `uid_t` and `gid_t` in kernel-private structures and code.

What's left to be done for 32-bit UIDs on all Linux architectures:

- Disk quotas have an interesting limitation that is not related to the maximum UID/GID. They are limited by the maximum file size on the underlying filesystem, because quota records are written at offsets corresponding to the UID in question. Further investigation is needed to see if the quota system can cope properly with huge UIDs. If it can deal with 64-bit file offsets on all architectures, this should not be a problem.
- Decide whether or not to keep backwards compatibility with the system accounting file, or if we should break it as the comments suggest (currently, the old 16-bit UID and GID are still written to disk, and part of the former pad space is used to store separate 32-bit UID and GID)
- Need to validate that OS emulation calls the 16-bit UID compatibility syscalls, if the OS being emulated used 16-bit UIDs, or uses the 32-bit UID system calls properly otherwise.

This affects at least:

- iBCS on Intel
  - sparc32 emulation on sparc64 (need to support whatever new 32-bit UID system calls are added to sparc32)
- Validate that all filesystems behave properly.

At present, 32-bit UIDs should work for:

- ext2
- ufs
- isofs
- nfs

- coda
- udf

Ioctl() fixups have been made for:

- ncpfs
- smbfs

Filesystems with simple fixups to prevent 16-bit UID wraparound:

- minix
- sysv
- qnx4

Other filesystems have not been checked yet.

- The ncpfs and smpfs filesystems cannot presently use 32-bit UIDs in all ioctl(s). Some new ioctl(s) have been added with 32-bit UIDs, but more are needed. (as well as new user<->kernel data structures)
- The ELF core dump format only supports 16-bit UIDs on arm, i386, m68k, sh, and sparc32. Fixing this is probably not that important, but would require adding a new ELF section.
- The ioctl(s) used to control the in-kernel NFS server only support 16-bit UIDs on arm, i386, m68k, sh, and sparc32.
- make sure that the UID mapping feature of AX25 networking works properly (it should be safe because it's always used a 32-bit integer to communicate between user and kernel)

## **LINUX SUPPORT FOR RANDOM NUMBER GENERATOR IN I8XX CHIPSETS**

### **41.1 Introduction**

The `hw_random` framework is software that makes use of a special hardware feature on your CPU or motherboard, a Random Number Generator (RNG). The software has two parts: a core providing the `/dev/hwrng` character device and its `sysfs` support, plus a hardware-specific driver that plugs into that core.

To make the most effective use of these mechanisms, you should download the support software as well. Download the latest version of the “`rng-tools`” package from the `hw_random` driver’s official Web site:

<http://sourceforge.net/projects/gkernel/>

Those tools use `/dev/hwrng` to fill the kernel entropy pool, which is used internally and exported by the `/dev/urandom` and `/dev/random` special files.

### **41.2 Theory of operation**

**CHARACTER DEVICE.** Using the standard `open()` and `read()` system calls, you can read random data from the hardware RNG device. This data is NOT CHECKED by any fitness tests, and could potentially be bogus (if the hardware is faulty or has been tampered with). Data is only output if the hardware “`has-data`” flag is set, but nevertheless a security-conscious person would run fitness tests on the data before assuming it is truly random.

The `rng-tools` package uses such tests in “`rngd`”, and lets you run them by hand with a “`rngtest`” utility.

`/dev/hwrng` is char device major 10, minor 183.

**CLASS DEVICE.** There is a `/sys/class/misc/hw_random` node with two unique attributes, “`rng_available`” and “`rng_current`”. The “`rng_available`” attribute lists the hardware-specific drivers available, while “`rng_current`” lists the one which is currently connected to `/dev/hwrng`. If your system has more than one RNG available, you may change the one used by writing a name from the list in “`rng_available`” into “`rng_current`”.

---

**Hardware driver for Intel/AMD/VIA Random Number Generators (RNG)**

---

- Copyright 2000,2001 Jeff Garzik <jgarzik@pobox.com>
- Copyright 2000,2001 Philipp Rumpf <prumpf@mandrakesoft.com>

### 41.3 About the Intel RNG hardware, from the firmware hub datasheet

The Firmware Hub integrates a Random Number Generator (RNG) using thermal noise generated from inherently random quantum mechanical properties of silicon. When not generating new random bits the RNG circuitry will enter a low power state. Intel will provide a binary software driver to give third party software access to our RNG for use as a security feature. At this time, the RNG is only to be used with a system in an OS-present state.

### 41.4 Intel RNG Driver notes

FIXME: support poll(2)

---

**Note:** request\_mem\_region was removed, for three reasons:

- 1) Only one RNG is supported by this driver;
  - 2) The location used by the RNG is a fixed location in MMIO-addressable memory;
  - 3) users with properly working BIOS e820 handling will always have the region in which the RNG is located reserved, so request\_mem\_region calls always fail for proper setups. However, for people who use mem=XX, BIOS e820 information is **not** in /proc/iomem, and request\_mem\_region(RNG\_ADDR) can succeed.
- 

### 41.5 Driver details

**Based on:** Intel 82802AB/82802AC Firmware Hub (FWH) Datasheet May 1999  
Order Number: 290658-002 R

**Intel 82802 Firmware Hub:** Random Number Generator Programmer's Reference Manual December 1999 Order Number: 298029-001 R

**Intel 82802 Firmware HUB Random Number Generator Driver** Copyright  
(c) 2000 Matt Sottek <msottek@quiknet.com>

Special thanks to Matt Sottek. I did the "guts" , he did the "brains" and all the testing.

## **USING THE INITIAL RAM DISK (INITRD)**

Written 1996,2000 by Werner Almesberger <[werner.almesberger@epfl.ch](mailto:werner.almesberger@epfl.ch)> and Hans Lermen <[lermen@fgan.de](mailto:lermen@fgan.de)>

initrd provides the capability to load a RAM disk by the boot loader. This RAM disk can then be mounted as the root file system and programs can be run from it. Afterwards, a new root file system can be mounted from a different device. The previous root (from initrd) is then moved to a directory and can be subsequently unmounted.

initrd is mainly designed to allow system startup to occur in two phases, where the kernel comes up with a minimum set of compiled-in drivers, and where additional modules are loaded from initrd.

This document gives a brief overview of the use of initrd. A more detailed discussion of the boot process can be found in<sup>1</sup>.

### **42.1 Operation**

When using initrd, the system typically boots as follows:

- 1) the boot loader loads the kernel and the initial RAM disk
- 2) the kernel converts initrd into a “normal” RAM disk and frees the memory used by initrd
- 3) if the root device is not `/dev/ram0`, the old (deprecated) `change_root` procedure is followed. see the “Obsolete root change mechanism” section below.
- 4) root device is mounted. if it is `/dev/ram0`, the initrd image is then mounted as root
- 5) `/sbin/init` is executed (this can be any valid executable, including shell scripts; it is run with uid 0 and can do basically everything init can do).
- 6) init mounts the “real” root file system
- 7) init places the root file system at the root directory using the `pivot_root` system call
- 8) init execs the `/sbin/init` on the new root filesystem, performing the usual boot sequence

---

<sup>1</sup> Almesberger, Werner; “Booting Linux: The History and the Future” <https://www.almesberger.net/cv/papers/ols2k-9.ps.gz>

9) the `initrd` file system is removed

Note that changing the root directory does not involve unmounting it. It is therefore possible to leave processes running on `initrd` during that procedure. Also note that file systems mounted under `initrd` continue to be accessible.

## 42.2 Boot command-line options

`initrd` adds the following new options:

```
initrd=<path>    (e.g. LOADLIN)

Loads the specified file as the initial RAM disk. When using LILO, you
have to specify the RAM disk image file in /etc/lilo.conf, using the
INITRD configuration variable.

noinitrd

initrd data is preserved but it is not converted to a RAM disk and
the "normal" root file system is mounted. initrd data can be read
from /dev/initrd. Note that the data in initrd can have any structure
in this case and doesn't necessarily have to be a file system image.
This option is used mainly for debugging.

Note: /dev/initrd is read-only and it can only be used once. As soon
as the last process has closed it, all data is freed and /dev/initrd
can't be opened anymore.

root=/dev/ram0

initrd is mounted as root, and the normal boot procedure is followed,
with the RAM disk mounted as root.
```

## 42.3 Compressed cpio images

Recent kernels have support for populating a ramdisk from a compressed cpio archive. On such systems, the creation of a ramdisk image doesn't need to involve special block devices or loopbacks; you merely create a directory on disk with the desired `initrd` content, `cd` to that directory, and run (as an example):

```
find . | cpio --quiet -H newc -o | gzip -9 -n > /boot/imagefile.img
```

Examining the contents of an existing image file is just as simple:

```
mkdir /tmp/imagefile
cd /tmp/imagefile
gzip -cd /boot/imagefile.img | cpio -imd --quiet
```

## 42.4 Installation

First, a directory for the `initrd` file system has to be created on the “normal” root file system, e.g.:

```
# mkdir /initrd
```

The name is not relevant. More details can be found on the `pivot_root(2)` man page.

If the root file system is created during the boot procedure (i.e. if you’re building an install floppy), the root file system creation procedure should create the `/initrd` directory.

If `initrd` will not be mounted in some cases, its content is still accessible if the following device has been created:

```
# mknod /dev/initrd b 1 250
# chmod 400 /dev/initrd
```

Second, the kernel has to be compiled with RAM disk support and with support for the initial RAM disk enabled. Also, at least all components needed to execute programs from `initrd` (e.g. executable format and file system) must be compiled into the kernel.

Third, you have to create the RAM disk image. This is done by creating a file system on a block device, copying files to it as needed, and then copying the content of the block device to the `initrd` file. With recent kernels, at least three types of devices are suitable for that:

- a floppy disk (works everywhere but it’s painfully slow)
- a RAM disk (fast, but allocates physical memory)
- a loopback device (the most elegant solution)

We’ll describe the loopback device method:

- 1) make sure loopback block devices are configured into the kernel
- 2) create an empty file system of the appropriate size, e.g.:

```
# dd if=/dev/zero of=initrd bs=300k count=1
# mke2fs -F -m0 initrd
```

(if space is critical, you may want to use the Minix FS instead of Ext2)

- 3) mount the file system, e.g.:

```
# mount -t ext2 -o loop initrd /mnt
```

- 4) create the console device:

```
# mkdir /mnt/dev
# mknod /mnt/dev/console c 5 1
```

- 5) copy all the files that are needed to properly use the `initrd` environment. Don’t forget the most important file, `/sbin/init`

---

**Note:** /sbin/init permissions must include “x” (execute).

---

- 6) correct operation the initrd environment can frequently be tested even without rebooting with the command:

```
# chroot /mnt /sbin/init
```

This is of course limited to initrds that do not interfere with the general system state (e.g. by reconfiguring network interfaces, overwriting mounted devices, trying to start already running demons, etc. Note however that it is usually possible to use pivot\_root in such a chroot’ ed initrd environment.)

- 7) unmount the file system:

```
# umount /mnt
```

- 8) the initrd is now in the file “initrd” . Optionally, it can now be compressed:

```
# gzip -9 initrd
```

For experimenting with initrd, you may want to take a rescue floppy and only add a symbolic link from /sbin/init to /bin/sh. Alternatively, you can try the experimental newlib environment<sup>2</sup> to create a small initrd.

Finally, you have to boot the kernel and load initrd. Almost all Linux boot loaders support initrd. Since the boot process is still compatible with an older mechanism, the following boot command line parameters have to be given:

```
root=/dev/ram0 rw
```

(rw is only necessary if writing to the initrd file system.)

With LOADLIN, you simply execute:

```
LOADLIN <kernel> initrd=<disk_image>
```

e.g.:

```
LOADLIN C:\LINUX\BZIMAGE initrd=C:\LINUX\INITRD.GZ root=/dev/ram0 rw
```

With LILO, you add the option INITRD=<path> to either the global section or to the section of the respective kernel in /etc/lilo.conf, and pass the options using APPEND, e.g.:

```
image = /bzImage
  initrd = /boot/initrd.gz
  append = "root=/dev/ram0 rw"
```

and run /sbin/lilo

For other boot loaders, please refer to the respective documentation.

Now you can boot and enjoy using initrd.

---

<sup>2</sup> newlib package (experimental), with initrd example <https://www.sourceware.org/newlib/>

## 42.5 Changing the root device

When finished with its duties, `init` typically changes the root device and proceeds with starting the Linux system on the “real” root device.

**The procedure involves the following steps:**

- mounting the new root file system
- turning it into the root file system
- removing all accesses to the old (`initrd`) root file system
- unmounting the `initrd` file system and de-allocating the RAM disk

Mounting the new root file system is easy: it just needs to be mounted on a directory under the current root. Example:

```
# mkdir /new-root
# mount -o ro /dev/hda1 /new-root
```

The root change is accomplished with the `pivot_root` system call, which is also available via the `pivot_root` utility (see `pivot_root(8)` man page; `pivot_root` is distributed with `util-linux` version 2.10h or higher<sup>3</sup>). `pivot_root` moves the current root to a directory under the new root, and puts the new root at its place. The directory for the old root must exist before calling `pivot_root`. Example:

```
# cd /new-root
# mkdir initrd
# pivot_root . initrd
```

Now, the `init` process may still access the old root via its executable, shared libraries, standard input/output/error, and its current root directory. All these references are dropped by the following command:

```
# exec chroot . what-follows <dev/console >dev/console 2>&1
```

Where `what-follows` is a program under the new root, e.g. `/sbin/init`. If the new root file system will be used with `udev` and has no valid `/dev` directory, `udev` must be initialized before invoking `chroot` in order to provide `/dev/console`.

Note: implementation details of `pivot_root` may change with time. In order to ensure compatibility, the following points should be observed:

- before calling `pivot_root`, the current directory of the invoking process should point to the new root directory
- use `.` as the first argument, and the `_relative_path` of the directory for the old root as the second argument
- a `chroot` program must be available under the old and the new root
- `chroot` to the new root afterwards
- use relative paths for `dev/console` in the `exec` command

<sup>3</sup> `util-linux`: Miscellaneous utilities for Linux <https://www.kernel.org/pub/linux/utils/util-linux/>

Now, the `initrd` can be unmounted and the memory allocated by the RAM disk can be freed:

```
# umount /initrd
# blockdev --flushbufs /dev/ram0
```

It is also possible to use `initrd` with an NFS-mounted root, see the `pivot_root(8)` man page for details.

## 42.6 Usage scenarios

The main motivation for implementing `initrd` was to allow for modular kernel configuration at system installation. The procedure would work as follows:

- 1) system boots from floppy or other media with a minimal kernel (e.g. support for RAM disks, `initrd`, `a.out`, and the Ext2 FS) and loads `initrd`
- 2) `/sbin/init` determines what is needed to (1) mount the “real” root FS (i.e. device type, device drivers, file system) and (2) the distribution media (e.g. CD-ROM, network, tape, ...). This can be done by asking the user, by auto-probing, or by using a hybrid approach.
- 3) `/sbin/init` loads the necessary kernel modules
- 4) `/sbin/init` creates and populates the root file system (this doesn't have to be a very usable system yet)
- 5) `/sbin/init` invokes `pivot_root` to change the root file system and execs - via `chroot` - a program that continues the installation
- 6) the boot loader is installed
- 7) the boot loader is configured to load an `initrd` with the set of modules that was used to bring up the system (e.g. `/initrd` can be modified, then unmounted, and finally, the image is written from `/dev/ram0` or `/dev/rd/0` to a file)
- 8) now the system is bootable and additional installation tasks can be performed

The key role of `initrd` here is to re-use the configuration data during normal system operation without requiring the use of a bloated “generic” kernel or re-compiling or re-linking the kernel.

A second scenario is for installations where Linux runs on systems with different hardware configurations in a single administrative domain. In such cases, it is desirable to generate only a small set of kernels (ideally only one) and to keep the system-specific part of configuration information as small as possible. In this case, a common `initrd` could be generated with all the necessary modules. Then, only `/sbin/init` or a file read by it would have to be different.

A third scenario is more convenient recovery disks, because information like the location of the root FS partition doesn't have to be provided at boot time, but the system loaded from `initrd` can invoke a user-friendly dialog and it can also perform some sanity checks (or even some form of auto-detection).

Last not least, CD-ROM distributors may use it for better installation from CD, e.g. by using a boot floppy and bootstrapping a bigger RAM disk via `initrd` from CD;

or by booting via a loader like LOADLIN or directly from the CD-ROM, and loading the RAM disk from CD without need of floppies.

## 42.7 Obsolete root change mechanism

The following mechanism was used before the introduction of `pivot_root`. Current kernels still support it, but you should `_not_` rely on its continued availability.

It works by mounting the “real” root device (i.e. the one set with `rdev` in the kernel image or with `root=...` at the boot command line) as the root file system when `linuxrc` exits. The `initrd` file system is then unmounted, or, if it is still busy, moved to a directory `/initrd`, if such a directory exists on the new root file system.

In order to use this mechanism, you do not have to specify the boot command options `root`, `init`, or `rw`. (If specified, they will affect the real root file system, not the `initrd` environment.)

If `/proc` is mounted, the “real” root device can be changed from within `linuxrc` by writing the number of the new root FS device to the special file `/proc/sys/kernel/real-root-dev`, e.g.:

```
# echo 0x301 >/proc/sys/kernel/real-root-dev
```

Note that the mechanism is incompatible with NFS and similar file systems.

This old, deprecated mechanism is commonly called `change_root`, while the new, supported mechanism is called `pivot_root`.

## 42.8 Mixed `change_root` and `pivot_root` mechanism

In case you did not want to use `root=/dev/ram0` to trigger the `pivot_root` mechanism, you may create both `/linuxrc` and `/sbin/init` in your `initrd` image.

`/linuxrc` would contain only the following:

```
#!/bin/sh
mount -n -t proc proc /proc
echo 0x0100 >/proc/sys/kernel/real-root-dev
umount -n /proc
```

Once `linuxrc` exited, the kernel would mount again your `initrd` as root, this time executing `/sbin/init`. Again, it would be the duty of this `init` to build the right environment (maybe using the `root=` device passed on the `cmdline`) before the final execution of the real `/sbin/init`.

## 42.9 Resources

## I/O STATISTICS FIELDS

Since 2.4.20 (and some versions before, with patches), and 2.5.45, more extensive disk statistics have been introduced to help measure disk activity. Tools such as `sar` and `iostat` typically interpret these and do the work for you, but in case you are interested in creating your own tools, the fields are explained here.

In 2.4 now, the information is found as additional fields in `/proc/partitions`. In 2.6 and upper, the same information is found in two places: one is in the file `/proc/diskstats`, and the other is within the `sysfs` file system, which must be mounted in order to obtain the information. Throughout this document we'll assume that `sysfs` is mounted on `/sys`, although of course it may be mounted anywhere. Both `/proc/diskstats` and `sysfs` use the same source for the information and so should not differ.

Here are examples of these different formats:

```
2.4:
 3      0   39082680 hda 446216 784926 9550688 4382310 424847 312726
↳5922052 19310380 0 3376340 23705160
 3      1   9221278 hda1 35486 0 35496 38030 0 0 0 0 0 38030 38030

2.6+ sysfs:
 446216 784926 9550688 4382310 424847 312726 5922052 19310380 0 3376340
↳23705160
 35486      38030      38030      38030

2.6+ diskstats:
 3      0   hda 446216 784926 9550688 4382310 424847 312726 5922052
↳19310380 0 3376340 23705160
 3      1   hda1 35486 38030 38030 38030

4.18+ diskstats:
 3      0   hda 446216 784926 9550688 4382310 424847 312726 5922052
↳19310380 0 3376340 23705160 0 0 0 0
```

On 2.4 you might execute `grep 'hda ' /proc/partitions`. On 2.6+, you have a choice of `cat /sys/block/hda/stat` or `grep 'hda ' /proc/diskstats`.

The advantage of one over the other is that the `sysfs` choice works well if you are watching a known, small set of disks. `/proc/diskstats` may be a better choice if you are watching a large number of disks because you'll avoid the overhead of 50, 100, or 500 or more opens/closes with each snapshot of your disk statistics.

In 2.4, the statistics fields are those after the device name. In the above example, the first field of statistics would be 446216. By contrast, in 2.6+ if you look at

/sys/block/hda/stat, you'll find just the 15 fields, beginning with 446216. If you look at /proc/diskstats, the 15 fields will be preceded by the major and minor device numbers, and device name. Each of these formats provides 15 fields of statistics, each meaning exactly the same things. All fields except field 9 are cumulative since boot. Field 9 should go to zero as I/Os complete; all others only increase (unless they overflow and wrap). Wrapping might eventually occur on a very busy or long-lived system; so applications should be prepared to deal with it. Regarding wrapping, the types of the fields are either unsigned int (32 bit) or unsigned long (32-bit or 64-bit, depending on your machine) as noted per-field below. Unless your observations are very spread in time, these fields should not wrap twice before you notice it.

Each set of stats only applies to the indicated device; if you want system-wide stats you'll have to find all the devices and sum them all up.

**Field 1 - # of reads completed (unsigned long)** This is the total number of reads completed successfully.

**Field 2 - # of reads merged, field 6 - # of writes merged (unsigned long)** Reads and writes which are adjacent to each other may be merged for efficiency. Thus two 4K reads may become one 8K read before it is ultimately handed to the disk, and so it will be counted (and queued) as only one I/O. This field lets you know how often this was done.

**Field 3 - # of sectors read (unsigned long)** This is the total number of sectors read successfully.

**Field 4 - # of milliseconds spent reading (unsigned int)** This is the total number of milliseconds spent by all reads (as measured from `__make_request()` to `end_that_request_last()`).

**Field 5 - # of writes completed (unsigned long)** This is the total number of writes completed successfully.

**Field 6 - # of writes merged (unsigned long)** See the description of field 2.

**Field 7 - # of sectors written (unsigned long)** This is the total number of sectors written successfully.

**Field 8 - # of milliseconds spent writing (unsigned int)** This is the total number of milliseconds spent by all writes (as measured from `__make_request()` to `end_that_request_last()`).

**Field 9 - # of I/Os currently in progress (unsigned int)** The only field that should go to zero. Incremented as requests are given to appropriate struct `request_queue` and decremented as they finish.

**Field 10 - # of milliseconds spent doing I/Os (unsigned int)** This field increases so long as field 9 is nonzero.

Since 5.0 this field counts jiffies when at least one request was started or completed. If request runs more than 2 jiffies then some I/O time might be not accounted in case of concurrent requests.

**Field 11 - weighted # of milliseconds spent doing I/Os (unsigned int)** This field is incremented at each I/O start, I/O completion, I/O merge, or read of these stats by the number of I/Os in progress (field 9) times the number of milliseconds spent doing I/O since the last update of this field. This can

provide an easy measure of both I/O completion time and the backlog that may be accumulating.

**Field 12 - # of discards completed (unsigned long)** This is the total number of discards completed successfully.

**Field 13 - # of discards merged (unsigned long)** See the description of field 2

**Field 14 - # of sectors discarded (unsigned long)** This is the total number of sectors discarded successfully.

**Field 15 - # of milliseconds spent discarding (unsigned int)** This is the total number of milliseconds spent by all discards (as measured from `__make_request()` to `end_that_request_last()`).

**Field 16 - # of flush requests completed** This is the total number of flush requests completed successfully.

Block layer combines flush requests and executes at most one at a time. This counts flush requests executed by disk. Not tracked for partitions.

**Field 17 - # of milliseconds spent flushing** This is the total number of milliseconds spent by all flush requests.

To avoid introducing performance bottlenecks, no locks are held while modifying these counters. This implies that minor inaccuracies may be introduced when changes collide, so (for instance) adding up all the read I/Os issued per partition should equal those made to the disks ...but due to the lack of locking it may only be very close.

In 2.6+, there are counters for each CPU, which make the lack of locking almost a non-issue. When the statistics are read, the per-CPU counters are summed (possibly overflowing the unsigned long variable they are summed to) and the result given to the user. There is no convenient user interface for accessing the per-CPU counters themselves.

Since 4.19 request times are measured with nanoseconds precision and truncated to milliseconds before showing in this interface.

## 43.1 Disks vs Partitions

There were significant changes between 2.4 and 2.6+ in the I/O subsystem. As a result, some statistic information disappeared. The translation from a disk address relative to a partition to the disk address relative to the host disk happens much earlier. All merges and timings now happen at the disk level rather than at both the disk and partition level as in 2.4. Consequently, you'll see a different statistics output on 2.6+ for partitions from that for disks. There are only four fields available for partitions on 2.6+ machines. This is reflected in the examples above.

**Field 1 - # of reads issued** This is the total number of reads issued to this partition.

**Field 2 - # of sectors read** This is the total number of sectors requested to be read from this partition.

**Field 3 - # of writes issued** This is the total number of writes issued to this partition.

**Field 4 - # of sectors written** This is the total number of sectors requested to be written to this partition.

Note that since the address is translated to a disk-relative one, and no record of the partition-relative address is kept, the subsequent success or failure of the read cannot be attributed to the partition. In other words, the number of reads for partitions is counted slightly before time of queuing for partitions, and at completion for whole disks. This is a subtle distinction that is probably uninteresting for most cases.

More significant is the error induced by counting the numbers of reads/writes before merges for partitions and after for disks. Since a typical workload usually contains a lot of successive and adjacent requests, the number of reads/writes issued can be several times higher than the number of reads/writes completed.

In 2.6.25, the full statistic set is again available for partitions and disk and partition statistics are consistent again. Since we still don't keep record of the partition-relative address, an operation is attributed to the partition which contains the first sector of the request after the eventual merges. As requests can be merged across partition, this could lead to some (probably insignificant) inaccuracy.

## 43.2 Additional notes

In 2.6+, `sysfs` is not mounted by default. If your distribution of Linux hasn't added it already, here's the line you'll want to add to your `/etc/fstab`:

```
none /sys sysfs defaults 0 0
```

In 2.6+, all disk statistics were removed from `/proc/stat`. In 2.4, they appear in both `/proc/partitions` and `/proc/stat`, although the ones in `/proc/stat` take a very different format from those in `/proc/partitions` (see `proc(5)`, if your system has it.)

- [ricklind@us.ibm.com](mailto:ricklind@us.ibm.com)

## **JAVA(TM) BINARY KERNEL SUPPORT FOR LINUX V1.03**

Linux beats them ALL! While all other OS' s are TALKING about direct support of Java Binaries in the OS, Linux is doing it!

You can execute Java applications and Java Applets just like any other program after you have done the following:

- 1) You MUST FIRST install the Java Developers Kit for Linux. The Java on Linux HOWTO gives the details on getting and installing this. This HOWTO can be found at:

<ftp://sunsite.unc.edu/pub/Linux/docs/HOWTO/Java-HOWTO>

You should also set up a reasonable CLASSPATH environment variable to use Java applications that make use of any nonstandard classes (not included in the same directory as the application itself).

- 2) You have to compile BINFMT\_MISC either as a module or into the kernel (CONFIG\_BINFMT\_MISC) and set it up properly. If you choose to compile it as a module, you will have to insert it manually with modprobe/insmod, as kmod cannot easily be supported with binfmt\_misc. Read the file 'binfmt\_misc.txt' in this directory to know more about the configuration process.
- 3) Add the following configuration items to binfmt\_misc (you should really have read binfmt\_misc.txt now): support for Java applications:

```
':Java:M::\xca\xfe\xba\xbe::/usr/local/bin/javawrapper:'
```

support for executable Jar files:

```
':ExecutableJAR:E::jar::/usr/local/bin/jarwrapper:'
```

support for Java Applets:

```
':Applet:E::html::/usr/bin/appletviewer:'
```

or the following, if you want to be more selective:

```
':Applet:M::<!--applet::/usr/bin/appletviewer:'
```

Of course you have to fix the path names. The path/file names given in this document match the Debian 2.1 system. (i.e. jdk installed in /usr, custom wrappers from this document in /usr/local)

Note, that for the more selective applet support you have to modify existing html-files to contain `<!--applet-->` in the first line (< has to be the first character!) to let this work!

For the compiled Java programs you need a wrapper script like the following (this is because Java is broken in case of the filename handling), again fix the path names, both in the script and in the above given configuration string.

You, too, need the little program after the script. Compile like:

```
gcc -O2 -o javaclassname javaclassname.c
```

and stick it to `/usr/local/bin`.

Both the javawrapper shellsript and the javaclassname program were supplied by Colin J. Watson <cjw44@cam.ac.uk>.

Javawrapper shell script:

```
#!/bin/bash
# /usr/local/bin/javawrapper - the wrapper for binfmt_misc/java

if [ -z "$1" ]; then
    exec 1>&2
    echo Usage: $0 class-file
    exit 1
fi

CLASS=$1
FQCLASS=`/usr/local/bin/javaclassname $1`
FQCLASSN=`echo $FQCLASS | sed -e 's/^\.*\.\.([^\.]*)$/\1/'`
FQCLASSP=`echo $FQCLASS | sed -e 's-\.-/-g' -e 's-^[^/]*$--' -e 's-/[^\/*]*$-
->'`

# for example:
# CLASS=Test.class
# FQCLASS=foo.bar.Test
# FQCLASSN=Test
# FQCLASSP=foo/bar

unset CLASSBASE

declare -i LINKLEVEL=0

while ;; do
    if [ "`basename $CLASS .class`" == "$FQCLASSN" ]; then
        # See if this directory works straight off
        cd -L `dirname $CLASS`
        CLASSDIR=$PWD
        cd $OLDPWD
        if echo $CLASSDIR | grep -q "$FQCLASSP$"; then
            CLASSBASE=`echo $CLASSDIR | sed -e "s.$FQCLASSP$.."`
            break;
        fi
        # Try dereferencing the directory name
        cd -P `dirname $CLASS`
        CLASSDIR=$PWD
        cd $OLDPWD
    fi
done
```

(continues on next page)

(continued from previous page)

```

        if echo $CLASSDIR | grep -q "$FQCLASSP$"; then
            CLASSBASE=`echo $CLASSDIR | sed -e "s.$FQCLASSP$.."`
            break;
        fi
        # If no other possible filename exists
        if [ ! -L $CLASS ]; then
            exec 1>&2
            echo $0:
            echo " $CLASS should be in a" \
                "directory tree called $FQCLASSP"
            exit 1
        fi
    fi
    if [ ! -L $CLASS ]; then break; fi
    # Go down one more level of symbolic links
    let LINKLEVEL+=1
    if [ $LINKLEVEL -gt 5 ]; then
        exec 1>&2
        echo $0:
        echo " Too many symbolic links encountered"
        exit 1
    fi
    CLASS=`ls --color=no -l $CLASS | sed -e 's/^.* \([^ ]*\)$/\1/'`
done
if [ -z "$CLASSBASE" ]; then
    if [ -z "$FQCLASSP" ]; then
        GOODNAME=$FQCLASSN.class
    else
        GOODNAME=$FQCLASSP/$FQCLASSN.class
    fi
    exec 1>&2
    echo $0:
    echo " $FQCLASS should be in a file called $GOODNAME"
    exit 1
fi
if ! echo $CLASSPATH | grep -q "^\(.*:\)*$CLASSBASE\(:.*\)*"; then
    # class is not in CLASSPATH, so prepend dir of class to CLASSPATH
    if [ -z "${CLASSPATH}" ]; then
        export CLASSPATH=$CLASSBASE
    else
        export CLASSPATH=$CLASSBASE:$CLASSPATH
    fi
fi
shift
/usr/bin/java $FQCLASS "$@"

```

javaclassname.c:

```

/* javaclassname.c
 *
 * Extracts the class name from a Java class file; intended for use in a
↳Java
 * wrapper of the type supported by the binfmt_misc option in the Linux
↳kernel.

```

(continues on next page)

(continued from previous page)

```
*
* Copyright (C) 1999 Colin J. Watson <cjw44@cam.ac.uk>.
*
* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
↳USA
*/

#include <stdlib.h>
#include <stdio.h>
#include <stdarg.h>
#include <sys/types.h>

/* From Sun's Java VM Specification, as tag entries in the constant pool.
↳*/

#define CP_UTF8 1
#define CP_INTEGER 3
#define CP_FLOAT 4
#define CP_LONG 5
#define CP_DOUBLE 6
#define CP_CLASS 7
#define CP_STRING 8
#define CP_FIELDREF 9
#define CP_METHODREF 10
#define CP_INTERFACEMETHODREF 11
#define CP_NAMEANDTYPE 12
#define CP_METHODHANDLE 15
#define CP_METHODTYPE 16
#define CP_INVOKEDYNAMIC 18

/* Define some commonly used error messages */

#define seek_error() error("%s: Cannot seek\n", program)
#define corrupt_error() error("%s: Class file corrupt\n", program)
#define eof_error() error("%s: Unexpected end of file\n", program)
#define utf8_error() error("%s: Only ASCII 1-255 supported\n", program);

char *program;

long *pool;

u_int8_t read_8(FILE *classfile);
u_int16_t read_16(FILE *classfile);
void skip_constant(FILE *classfile, u_int16_t *cur);
```

(continues on next page)

(continued from previous page)

```
void error(const char *format, ...);
int main(int argc, char **argv);

/* Reads in an unsigned 8-bit integer. */
u_int8_t read_8(FILE *classfile)
{
    int b = fgetc(classfile);
    if(b == EOF)
        eof_error();
    return (u_int8_t)b;
}

/* Reads in an unsigned 16-bit integer. */
u_int16_t read_16(FILE *classfile)
{
    int b1, b2;
    b1 = fgetc(classfile);
    if(b1 == EOF)
        eof_error();
    b2 = fgetc(classfile);
    if(b2 == EOF)
        eof_error();
    return (u_int16_t)((b1 << 8) | b2);
}

/* Reads in a value from the constant pool. */
void skip_constant(FILE *classfile, u_int16_t *cur)
{
    u_int16_t len;
    int seekerr = 1;
    pool[*cur] = ftell(classfile);
    switch(read_8(classfile))
    {
        case CP_UTF8:
            len = read_16(classfile);
            seekerr = fseek(classfile, len, SEEK_CUR);
            break;
        case CP_CLASS:
        case CP_STRING:
        case CP_METHODTYPE:
            seekerr = fseek(classfile, 2, SEEK_CUR);
            break;
        case CP_METHODHANDLE:
            seekerr = fseek(classfile, 3, SEEK_CUR);
            break;
        case CP_INTEGER:
        case CP_FLOAT:
        case CP_FIELDREF:
        case CP_METHODREF:
        case CP_INTERFACEMETHODREF:
        case CP_NAMEANDTYPE:
        case CP_INVOKEDYNAMIC:
            seekerr = fseek(classfile, 4, SEEK_CUR);
            break;
        case CP_LONG:
        case CP_DOUBLE:
```

(continues on next page)

(continued from previous page)

```

        seekerr = fseek(classfile, 8, SEEK_CUR);
        ++(*cur);
        break;
    default:
        corrupt_error();
    }
    if(seekerr)
        seek_error();
}

void error(const char *format, ...)
{
    va_list ap;
    va_start(ap, format);
    vfprintf(stderr, format, ap);
    va_end(ap);
    exit(1);
}

int main(int argc, char **argv)
{
    FILE *classfile;
    u_int16_t cp_count, i, this_class, classinfo_ptr;
    u_int8_t length;

    program = argv[0];

    if(!argv[1])
        error("%s: Missing input file\n", program);
    classfile = fopen(argv[1], "rb");
    if(!classfile)
        error("%s: Error opening %s\n", program, argv[1]);

    if(fseek(classfile, 8, SEEK_SET)) /* skip magic and version numbers_
→*/
        seek_error();
    cp_count = read_16(classfile);
    pool = calloc(cp_count, sizeof(long));
    if(!pool)
        error("%s: Out of memory for constant pool\n", program);

    for(i = 1; i < cp_count; ++i)
        skip_constant(classfile, &i);
    if(fseek(classfile, 2, SEEK_CUR)) /* skip access flags */
        seek_error();

    this_class = read_16(classfile);
    if(this_class < 1 || this_class >= cp_count)
        corrupt_error();
    if(!pool[this_class] || pool[this_class] == -1)
        corrupt_error();
    if(fseek(classfile, pool[this_class] + 1, SEEK_SET))
        seek_error();

    classinfo_ptr = read_16(classfile);
    if(classinfo_ptr < 1 || classinfo_ptr >= cp_count)

```

(continues on next page)

(continued from previous page)

```

        corrupt_error();
    if(!pool[classinfo_ptr] || pool[classinfo_ptr] == -1)
        corrupt_error();
    if(fseek(classfile, pool[classinfo_ptr] + 1, SEEK_SET))
        seek_error();

    length = read_16(classfile);
    for(i = 0; i < length; ++i)
    {
        u_int8_t x = read_8(classfile);
        if((x & 0x80) || !x)
        {
            if((x & 0xE0) == 0xC0)
            {
                u_int8_t y = read_8(classfile);
                if((y & 0xC0) == 0x80)
                {
                    int c = ((x & 0x1f) << 6) + (y &
→0x3f);

                    if(c) putchar(c);
                    else utf8_error();
                }
                else utf8_error();
            }
            else if(x == '/') putchar('.');
            else putchar(x);
        }
        putchar('\n');
        free(pool);
        fclose(classfile);
        return 0;
    }
}

```

jarwrapper:

```

#!/bin/bash
# /usr/local/java/bin/jarwrapper - the wrapper for binfmt_misc/jar

java -jar $1

```

Now simply `chmod +x` the `.class`, `.jar` and/or `.html` files you want to execute.

To add a Java program to your path best put a symbolic link to the main `.class` file into `/usr/bin` (or another place you like) omitting the `.class` extension. The directory containing the original `.class` file will be added to your `CLASSPATH` during execution.

To test your new setup, enter in the following simple Java app, and name it “HelloWorld.java” :

```

class HelloWorld {
    public static void main(String args[]) {
        System.out.println("Hello World!");
    }
}

```

(continues on next page)

(continued from previous page)

```
}  
}
```

Now compile the application with:

```
javac HelloWorld.java
```

Set the executable permissions of the binary file, with:

```
chmod 755 HelloWorld.class
```

And then execute it:

```
./HelloWorld.class
```

To execute Java Jar files, simple chmod the \*.jar files to include the execution bit, then just do:

```
./Application.jar
```

To execute Java Applets, simple chmod the \*.html files to include the execution bit, then just do:

```
./Applet.html
```

originally by Brian A. Lantz, [brian@lantz.com](mailto:brian@lantz.com) heavily edited for binfmt\_misc by Richard Günther new scripts by Colin J. Watson <[cjw44@cam.ac.uk](mailto:cjw44@cam.ac.uk)> added executable Jar file support by Kurt Huwig <[kurt@iku-netz.de](mailto:kurt@iku-netz.de)>

## **IBM<sup>®</sup> S JOURNALLED FILE SYSTEM (JFS) FOR LINUX**

JFS Homepage: <http://jfs.sourceforge.net/>

The following mount options are supported:

(\*) == default

**iocharset=name** Character set to use for converting from Unicode to ASCII. The default is to do no conversion. Use `iocharset=utf8` for UTF-8 translations. This requires `CONFIG_NLS_UTF8` to be set in the kernel `.config` file. `iocharset=none` specifies the default behavior explicitly.

**resize=value** Resize the volume to `<value>` blocks. JFS only supports growing a volume, not shrinking it. This option is only valid during a remount, when the volume is mounted read-write. The `resize` keyword with no value will grow the volume to the full size of the partition.

**nointegrity** Do not write to the journal. The primary use of this option is to allow for higher performance when restoring a volume from backup media. The integrity of the volume is not guaranteed if the system abnormally abends.

**integrity(\*)** Commit metadata changes to the journal. Use this option to remount a volume where the `nointegrity` option was previously specified in order to restore normal behavior.

**errors=continue** Keep going on a filesystem error.

**errors=remount-ro(\*)** Remount the filesystem read-only on an error.

**errors=panic** Panic and halt the machine if an error occurs.

**uid=value** Override on-disk uid with specified value

**gid=value** Override on-disk gid with specified value

**umask=value** Override on-disk umask with specified octal value. For directories, the execute bit will be set if the corresponding read bit is set.

**discard=minlen, discard/nodiscard(\*)** This enables/disables the use of `discard/TRIM` commands. The `discard/TRIM` commands are sent to the underlying block device when blocks are freed. This is useful for SSD devices and sparse/thinly-provisioned LUNs. The `FITRIM` `ioctl` command is also available together with the `nodiscard` option. The value of `minlen` specifies the minimum blockcount, when a `TRIM` command to the block device is considered useful. When no value is given to the `discard` option, it defaults to 64 blocks, which means 256KiB in JFS. The `minlen` value of `discard` overrides the `minlen` value given on an `FITRIM` `ioctl()`.

The JFS mailing list can be subscribed to by using the link labeled “Mail list Subscribe” at our web page <http://jfs.sourceforge.net/>

## **REDUCING OS JITTER DUE TO PER-CPU KTHREADS**

This document lists per-CPU kthreads in the Linux kernel and presents options to control their OS jitter. Note that non-per-CPU kthreads are not listed here. To reduce OS jitter from non-per-CPU kthreads, bind them to a “housekeeping” CPU dedicated to such work.

### **46.1 References**

- Documentation/core-api/irq/irq-affinity.rst: Binding interrupts to sets of CPUs.
- Documentation/admin-guide/cgroup-v1: Using cgroups to bind tasks to sets of CPUs.
- man taskset: Using the taskset command to bind tasks to sets of CPUs.
- man sched\_setaffinity: Using the sched\_setaffinity() system call to bind tasks to sets of CPUs.
- /sys/devices/system/cpu/cpuN/online: Control CPU N’s hotplug state, writing “0” to offline and “1” to online.
- In order to locate kernel-generated OS jitter on CPU N:

```
cd /sys/kernel/debug/tracing echo 1 > max_graph_depth # Increase
the “1” for more detail echo function_graph > current_tracer # run
workload cat per_cpu/cpuN/trace
```

### **46.2 kthreads**

**Name:** ehca\_comp/%u

**Purpose:** Periodically process Infiniband-related work.

To reduce its OS jitter, do any of the following:

1. Don’ t use eHCA Infiniband hardware, instead choosing hardware that does not require per-CPU kthreads. This will prevent these kthreads from being created in the first place. (This will work for most people, as this hardware, though important, is relatively old and is produced in relatively low unit volumes.)

2. Do all eHCA-Infiniband-related work on other CPUs, including interrupts.
3. Rework the eHCA driver so that its per-CPU kthreads are provisioned only on selected CPUs.

**Name:** irq/%d-%s

**Purpose:** Handle threaded interrupts.

To reduce its OS jitter, do the following:

1. Use irq affinity to force the irq threads to execute on some other CPU.

**Name:** kcmtpd\_ctr\_%d

**Purpose:** Handle Bluetooth work.

To reduce its OS jitter, do one of the following:

1. Don't use Bluetooth, in which case these kthreads won't be created in the first place.
2. Use irq affinity to force Bluetooth-related interrupts to occur on some other CPU and furthermore initiate all Bluetooth activity on some other CPU.

**Name:** ksoftirqd/%u

**Purpose:** Execute softirq handlers when threaded or when under heavy load.

To reduce its OS jitter, each softirq vector must be handled separately as follows:

### 46.2.1 TIMER\_SOFTIRQ

Do all of the following:

1. To the extent possible, keep the CPU out of the kernel when it is non-idle, for example, by avoiding system calls and by forcing both kernel threads and interrupts to execute elsewhere.
2. Build with CONFIG\_HOTPLUG\_CPU=y. After boot completes, force the CPU offline, then bring it back online. This forces recurring timers to migrate elsewhere. If you are concerned with multiple CPUs, force them all offline before bringing the first one back online. Once you have onlined the CPUs in question, do not offline any other CPUs, because doing so could force the timer back onto one of the CPUs in question.

### 46.2.2 NET\_TX\_SOFTIRQ and NET\_RX\_SOFTIRQ

Do all of the following:

1. Force networking interrupts onto other CPUs.
2. Initiate any network I/O on other CPUs.
3. Once your application has started, prevent CPU-hotplug operations from being initiated from tasks that might run on the CPU to be de-jittered. (It is OK to force this CPU offline and then bring it back online before you start your application.)

### **46.2.3 BLOCK\_SOFTIRQ**

Do all of the following:

1. Force block-device interrupts onto some other CPU.
2. Initiate any block I/O on other CPUs.
3. Once your application has started, prevent CPU-hotplug operations from being initiated from tasks that might run on the CPU to be de-jittered. (It is OK to force this CPU offline and then bring it back online before you start your application.)

### **46.2.4 IRQ\_POLL\_SOFTIRQ**

Do all of the following:

1. Force block-device interrupts onto some other CPU.
2. Initiate any block I/O and block-I/O polling on other CPUs.
3. Once your application has started, prevent CPU-hotplug operations from being initiated from tasks that might run on the CPU to be de-jittered. (It is OK to force this CPU offline and then bring it back online before you start your application.)

### **46.2.5 TASKLET\_SOFTIRQ**

Do one or more of the following:

1. Avoid use of drivers that use tasklets. (Such drivers will contain calls to things like `tasklet_schedule()`.)
2. Convert all drivers that you must use from tasklets to workqueues.
3. Force interrupts for drivers using tasklets onto other CPUs, and also do I/O involving these drivers on other CPUs.

### **46.2.6 SCHED\_SOFTIRQ**

Do all of the following:

1. Avoid sending scheduler IPIs to the CPU to be de-jittered, for example, ensure that at most one runnable kthread is present on that CPU. If a thread that expects to run on the de-jittered CPU awakens, the scheduler will send an IPI that can result in a subsequent `SCHED_SOFTIRQ`.
2. `CONFIG_NO_HZ_FULL=y` and ensure that the CPU to be de-jittered is marked as an adaptive-ticks CPU using the `"nohz_full="` boot parameter. This reduces the number of scheduler-clock interrupts that the de-jittered CPU receives, minimizing its chances of being selected to do the load balancing work that runs in `SCHED_SOFTIRQ` context.

3. To the extent possible, keep the CPU out of the kernel when it is non-idle, for example, by avoiding system calls and by forcing both kernel threads and interrupts to execute elsewhere. This further reduces the number of scheduler-clock interrupts received by the de-jittered CPU.

### 46.2.7 HRTIMER\_SOFTIRQ

Do all of the following:

1. To the extent possible, keep the CPU out of the kernel when it is non-idle. For example, avoid system calls and force both kernel threads and interrupts to execute elsewhere.
2. Build with `CONFIG_HOTPLUG_CPU=y`. Once boot completes, force the CPU offline, then bring it back online. This forces recurring timers to migrate elsewhere. If you are concerned with multiple CPUs, force them all offline before bringing the first one back online. Once you have onlined the CPUs in question, do not offline any other CPUs, because doing so could force the timer back onto one of the CPUs in question.

### 46.2.8 RCU\_SOFTIRQ

Do at least one of the following:

1. Offload callbacks and keep the CPU in either dyntick-idle or adaptive-ticks state by doing all of the following:
  - a. `CONFIG_NO_HZ_FULL=y` and ensure that the CPU to be de-jittered is marked as an adaptive-ticks CPU using the “`nohz_full=`” boot parameter. Bind the rcuo kthreads to housekeeping CPUs, which can tolerate OS jitter.
  - b. To the extent possible, keep the CPU out of the kernel when it is non-idle, for example, by avoiding system calls and by forcing both kernel threads and interrupts to execute elsewhere.
2. Enable RCU to do its processing remotely via dyntick-idle by doing all of the following:
  - a. Build with `CONFIG_NO_HZ=y` and `CONFIG_RCU_FAST_NO_HZ=y`.
  - b. Ensure that the CPU goes idle frequently, allowing other CPUs to detect that it has passed through an RCU quiescent state. If the kernel is built with `CONFIG_NO_HZ_FULL=y`, userspace execution also allows other CPUs to detect that the CPU in question has passed through a quiescent state.
  - c. To the extent possible, keep the CPU out of the kernel when it is non-idle, for example, by avoiding system calls and by forcing both kernel threads and interrupts to execute elsewhere.

**Name:** `kworker/%u:%d%s` (cpu, id, priority)

**Purpose:** Execute workqueue requests

To reduce its OS jitter, do any of the following:

1. Run your workload at a real-time priority, which will allow preempting the kworker daemons.
2. A given workqueue can be made visible in the sysfs filesystem by passing the WQ\_SYSFS to that workqueue's alloc\_workqueue(). Such a workqueue can be confined to a given subset of the CPUs using the /sys/devices/virtual/workqueue/\*/cpumask sysfs files. The set of WQ\_SYSFS workqueues can be displayed using "ls /sys/devices/virtual/workqueue". That said, the workqueues maintainer would like to caution people against indiscriminately sprinkling WQ\_SYSFS across all the workqueues. The reason for caution is that it is easy to add WQ\_SYSFS, but because sysfs is part of the formal user/kernel API, it can be nearly impossible to remove it, even if its addition was a mistake.
3. Do any of the following needed to avoid jitter that your application cannot tolerate:
  - a. Build your kernel with CONFIG\_SLUB=y rather than CONFIG\_SLAB=y, thus avoiding the slab allocator's periodic use of each CPU's workqueues to run its cache\_reap() function.
  - b. Avoid using oprofile, thus avoiding OS jitter from wq\_sync\_buffer().
  - c. Limit your CPU frequency so that a CPU-frequency governor is not required, possibly enlisting the aid of special heatsinks or other cooling technologies. If done correctly, and if your CPU architecture permits, you should be able to build your kernel with CONFIG\_CPU\_FREQ=n to avoid the CPU-frequency governor periodically running on each CPU, including cs\_dbs\_timer() and od\_dbs\_timer().

WARNING: Please check your CPU specifications to make sure that this is safe on your particular system.
  - d. As of v3.18, Christoph Lameter's on-demand vmstat workers commit prevents OS jitter due to vmstat\_update() on CONFIG\_SMP=y systems. Before v3.18, it is not possible to entirely get rid of the OS jitter, but you can decrease its frequency by writing a large value to /proc/sys/vm/stat\_interval. The default value is HZ, for an interval of one second. Of course, larger values will make your virtual-memory statistics update more slowly. Of course, you can also run your workload at a real-time priority, thus preempting vmstat\_update(), but if your workload is CPU-bound, this is a bad idea. However, there is an RFC patch from Christoph Lameter (based on an earlier one from Gilad Ben-Yossef) that reduces or even eliminates vmstat overhead for some workloads at <https://lkml.org/lkml/2013/9/4/379>.
  - e. If running on high-end powerpc servers, build with CONFIG\_PPC\_RTAS\_DAEMON=n. This prevents the RTAS daemon from running on each CPU every second or so. (This will require editing Kconfig files and will defeat this platform's RAS functionality.) This avoids jitter due to the rtas\_event\_scan() function. WARNING: Please check your CPU specifications to make sure that this is safe on your particular system.
  - f. If running on Cell Processor, build your kernel with CBE\_CPUFREQ\_SPU\_GOVERNOR=n to avoid OS jitter from

spu\_gov\_work()). **WARNING:** Please check your CPU specifications to make sure that this is safe on your particular system.

- g. If running on PowerMAC, build your kernel with `CONFIG_PMAC_RACKMETER=n` to disable the CPU-meter, avoiding OS jitter from `rackmeter_do_timer()`.

**Name:** `rcuc/%u`

**Purpose:** Execute RCU callbacks in `CONFIG_RCU_BOOST=y` kernels.

To reduce its OS jitter, do at least one of the following:

1. Build the kernel with `CONFIG_PREEMPT=n`. This prevents these kthreads from being created in the first place, and also obviates the need for RCU priority boosting. This approach is feasible for workloads that do not require high degrees of responsiveness.
2. Build the kernel with `CONFIG_RCU_BOOST=n`. This prevents these kthreads from being created in the first place. This approach is feasible only if your workload never requires RCU priority boosting, for example, if you ensure frequent idle time on all CPUs that might execute within the kernel.
3. Build with `CONFIG_RCU_NOCB_CPU=y` and boot with the `rcu_nocbs= boot` parameter offloading RCU callbacks from all CPUs susceptible to OS jitter. This approach prevents the `rcuc/%u` kthreads from having any work to do, so that they are never awakened.
4. Ensure that the CPU never enters the kernel, and, in particular, avoid initiating any CPU hotplug operations on this CPU. This is another way of preventing any callbacks from being queued on the CPU, again preventing the `rcuc/%u` kthreads from having any work to do.

**Name:** `rcuop/%d` and `rcuos/%d`

**Purpose:** Offload RCU callbacks from the corresponding CPU.

To reduce its OS jitter, do at least one of the following:

1. Use affinity, cgroups, or other mechanism to force these kthreads to execute on some other CPU.
2. Build with `CONFIG_RCU_NOCB_CPU=n`, which will prevent these kthreads from being created in the first place. However, please note that this will not eliminate OS jitter, but will instead shift it to `RCU_SOFTIRQ`.

**Name:** `watchdog/%u`

**Purpose:** Detect software lockups on each CPU.

To reduce its OS jitter, do at least one of the following:

1. Build with `CONFIG_LOCKUP_DETECTOR=n`, which will prevent these kthreads from being created in the first place.
2. Boot with `"nosoftlockup=0"`, which will also prevent these kthreads from being created. Other related watchdog and softlockup boot parameters may be found in `Documentation/admin-guide/kernel-parameters.rst` and `Documentation/watchdog/watchdog-parameters.rst`.
3. Echo a zero to `/proc/sys/kernel/watchdog` to disable the watchdog timer.

4. Echo a large number of `/proc/sys/kernel/watchdog_thresh` in order to reduce the frequency of OS jitter due to the watchdog timer down to a level that is acceptable for your workload.



## **LAPTOP DRIVERS**

### **47.1 Asus Laptop Extras**

Version 0.1

August 6, 2009

Corentin Chary <[corentincj@iksaif.net](mailto:corentincj@iksaif.net)> <http://acpi4asus.sf.net/>

This driver provides support for extra features of ACPI-compatible ASUS laptops. It may also support some MEDION, JVC or VICTOR laptops (such as MEDION 9675 or VICTOR XP7210 for example). It makes all the extra buttons generate input events (like keyboards).

On some models adds support for changing the display brightness and output, switching the LCD backlight on and off, and most importantly, allows you to blink those fancy LEDs intended for reporting mail and wireless status.

This driver supersedes the old `asus_acpi` driver.

#### **47.1.1 Requirements**

Kernel 2.6.X sources, configured for your computer, with ACPI support. You also need `CONFIG_INPUT` and `CONFIG_ACPI`.

#### **47.1.2 Status**

The features currently supported are the following (see below for detailed description):

- Fn key combinations
- Bluetooth enable and disable
- Wlan enable and disable
- GPS enable and disable
- Video output switching
- Ambient Light Sensor on and off
- LED control

- LED Display control
- LCD brightness control
- LCD on and off

A compatibility table by model and feature is maintained on the web site, <http://acpi4asus.sf.net/>.

### 47.1.3 Usage

Try “modprobe asus-laptop” . Check your dmesg (simply type dmesg). You should see some lines like this :

#### **Asus Laptop Extras version 0.42**

- L2D model detected.

If it is not the output you have on your laptop, send it (and the laptop’ s DSDT) to me.

That’ s all, now, all the events generated by the hotkeys of your laptop should be reported via netlink events. You can check with “acpi\_genl monitor” (part of the acpica project).

Hotkeys are also reported as input keys (like keyboards) you can check which key are supported using “xev” under X11.

You can get information on the version of your DSDT table by reading the `/sys/devices/platform/asus-laptop/infos` entry. If you have a question or a bug report to do, please include the output of this entry.

### 47.1.4 LEDs

You can modify LEDs be echoing values to `/sys/class/leds/asus*/brightness`:

```
echo 1 > /sys/class/leds/asus::mail/brightness
```

will switch the mail LED on.

You can also know if they are on/off by reading their content and use kernel triggers like `disk-activity` or `heartbeat`.

### 47.1.5 Backlight

You can control lcd backlight power and brightness with `/sys/class/backlight/asus-laptop/`. Brightness Values are between 0 and 15.

### 47.1.6 Wireless devices

You can turn the internal Bluetooth adapter on/off with the bluetooth entry (only on models with Bluetooth). This usually controls the associated LED. Same for Wlan adapter.

### 47.1.7 Display switching

Note: the display switching code is currently considered EXPERIMENTAL.

Switching works for the following models:

- L3800C
- A2500H
- L5800C
- M5200N
- W1000N (albeit with some glitches)
- M6700R
- A6JC
- F3J

Switching doesn't work for the following:

- M3700N
- L2X00D (locks the laptop under certain conditions)

To switch the displays, echo values from 0 to 15 to `/sys/devices/platform/asus-laptop/display`. The significance of those values is as follows:

Bin	Val	DVI	TV	CRT	LCD
0000	0				
0001	1				X
0010	2			X	
0011	3			X	X
0100	4		X		
0101	5		X		X
0110	6		X	X	
0111	7		X	X	X
1000	8	X			
1001	9	X			X
1010	10	X		X	
1011	11	X		X	X
1100	12	X	X		
1101	13	X	X		X
1110	14	X	X	X	
1111	15	X	X	X	X

In most cases, the appropriate displays must be plugged in for the above combinations to work. TV-Out may need to be initialized at boot time.

Debugging:

- 1) Check whether the Fn+F8 key:
  - a) does not lock the laptop (try a boot with noapic / nolapic if it does)
  - b) generates events (0x6n, where n is the value corresponding to the configuration above)
  - c) actually works

Record the disp value at every configuration.

- 2) Echo values from 0 to 15 to /sys/devices/platform/asus-laptop/display. Record its value, note any change. If nothing changes, try a broader range, up to 65535.
- 3) Send ANY output (both positive and negative reports are needed, unless your machine is already listed above) to the acpi4asus-user mailing list.

Note: on some machines (e.g. L3C), after the module has been loaded, only 0x6n events are generated and no actual switching occurs. In such a case, a line like:

```
echo $((10#$arg-60)) > /sys/devices/platform/asus-laptop/display
```

will usually do the trick (\$arg is the 0000006n-like event passed to acpid).

Note: there is currently no reliable way to read display status on xxN (Centrino) models.

### 47.1.8 LED display

Some models like the W1N have a LED display that can be used to display several items of information.

LED display works for the following models:

- W1000N
- W1J

To control the LED display, use the following:

```
echo 0x0T000DDD > /sys/devices/platform/asus-laptop/
```

where T control the 3 letters display, and DDD the 3 digits display, according to the tables below:

```
DDD (digits)
000 to 999 = display digits
AAA          = ---
```

(continues on next page)

(continued from previous page)

```
BBB to FFF = turn-off
```

```
T (type)
0 = off
1 = dvd
2 = vcd
3 = mp3
4 = cd
5 = tv
6 = cpu
7 = vol
```

For example “echo 0x01000001 >/sys/devices/platform/asus-laptop/ledd” would display “DVD001” .

### 47.1.9 Driver options

Options can be passed to the asus-laptop driver using the standard module argument syntax (<param>=<value> when passing the option to the module or asus-laptop.<param>=<value> on the kernel boot line when asus-laptop is statically linked into the kernel).

**wapf: WAPF defines the behavior of the Fn+Fx wlan key**

The significance of values is yet to be found, but most of the time:

- 0x0 should do nothing
- 0x1 should allow to control the device with Fn+Fx key.
- 0x4 should send an ACPI event (0x88) while pressing the Fn+Fx key
- 0x5 like 0x1 or 0x4

The default value is 0x1.

### 47.1.10 Unsupported models

These models will never be supported by this module, as they use a completely different mechanism to handle LEDs and extra stuff (meaning we have no clue how it works):

- ASUS A1300 (A1B), A1370D
- ASUS L7300G
- ASUS L8400

### 47.1.11 Patches, Errors, Questions

I appreciate any success or failure reports, especially if they add to or correct the compatibility table. Please include the following information in your report:

- Asus model name
- a copy of your ACPI tables, using the “acpidump” utility
- a copy of `/sys/devices/platform/asus-laptop/infos`
- which driver features work and which don’ t
- the observed behavior of non-working features

Any other comments or patches are also more than welcome.

[acpi4asus-user@lists.sourceforge.net](mailto:acpi4asus-user@lists.sourceforge.net)

<http://sourceforge.net/projects/acpi4asus>

## 47.2 Hard disk shock protection

Author: Elias Oltmanns <[eo@nebensachen.de](mailto:eo@nebensachen.de)>

Last modified: 2008-10-03

### 47.2.1 1. Intro

ATA/ATAPI-7 specifies the IDLE IMMEDIATE command with unload feature. Issuing this command should cause the drive to switch to idle mode and unload disk heads. This feature is being used in modern laptops in conjunction with accelerometers and appropriate software to implement a shock protection facility. The idea is to stop all I/O operations on the internal hard drive and park its heads on the ramp when critical situations are anticipated. The desire to have such a feature available on GNU/Linux systems has been the original motivation to implement a generic disk head parking interface in the Linux kernel. Please note, however, that other components have to be set up on your system in order to get disk shock protection working (see section 3. References below for pointers to more information about that).

### 47.2.2 2. The interface

For each ATA device, the kernel exports the file `block/*/device/unload_heads` in `sysfs` (here assumed to be mounted under `/sys`). Access to `/sys/block/*/device/unload_heads` is denied with `-EOPNOTSUPP` if the device does not support the unload feature. Otherwise, writing an integer value to this file will take the heads of the respective drive off the platter and block all I/O operations for the specified number of milliseconds. When the timeout expires and no further disk head park request has been issued in the meantime, normal operation will be resumed. The maximal value accepted for a timeout is 30000 milliseconds. Exceeding this limit will return `-EOVERFLOW`, but heads

will be parked anyway and the timeout will be set to 30 seconds. However, you can always change a timeout to any value between 0 and 30000 by issuing a subsequent head park request before the timeout of the previous one has expired. In particular, the total timeout can exceed 30 seconds and, more importantly, you can cancel a previously set timeout and resume normal operation immediately by specifying a timeout of 0. Values below -2 are rejected with -EINVAL (see below for the special meaning of -1 and -2). If the timeout specified for a recent head park request has not yet expired, reading from `/sys/block/*/device/unload_heads` will report the number of milliseconds remaining until normal operation will be resumed; otherwise, reading the `unload_heads` attribute will return 0.

For example, do the following in order to park the heads of drive `/dev/sda` and stop all I/O operations for five seconds:

```
# echo 5000 > /sys/block/sda/device/unload_heads
```

A simple:

```
# cat /sys/block/sda/device/unload_heads
```

will show you how many milliseconds are left before normal operation will be resumed.

A word of caution: The fact that the interface operates on a basis of milliseconds may raise expectations that cannot be satisfied in reality. In fact, the ATA specs clearly state that the time for an unload operation to complete is vendor specific. The hint in ATA-7 that this will typically be within 500 milliseconds apparently has been dropped in ATA-8.

There is a technical detail of this implementation that may cause some confusion and should be discussed here. When a head park request has been issued to a device successfully, all I/O operations on the controller port this device is attached to will be deferred. That is to say, any other device that may be connected to the same port will be affected too. The only exception is that a subsequent head unload request to that other device will be executed immediately. Further operations on that port will be deferred until the timeout specified for either device on the port has expired. As far as PATA (old style IDE) configurations are concerned, there can only be two devices attached to any single port. In SATA world we have port multipliers which means that a user-issued head parking request to one device may actually result in stopping I/O to a whole bunch of devices. However, since this feature is supposed to be used on laptops and does not seem to be very useful in any other environment, there will be mostly one device per port. Even if the CD/DVD writer happens to be connected to the same port as the hard drive, it generally should recover just fine from the occasional buffer under-run incurred by a head park request to the HD. Actually, when you are using an ide driver rather than its libata counterpart (i.e. your disk is called `/dev/hda` instead of `/dev/sda`), then parking the heads of one drive (drive X) will generally not affect the mode of operation of another drive (drive Y) on the same port as described above. It is only when a port reset is required to recover from an exception on drive Y that further I/O operations on that drive (and the reset itself) will be delayed until drive X is no longer in the parked state.

Finally, there are some hard drives that only comply with an earlier version of the ATA standard than ATA-7, but do support the unload feature nonetheless. Unfor-

Unfortunately, there is no safe way Linux can detect these devices, so you won't be able to write to the `unload_heads` attribute. If you know that your device really does support the unload feature (for instance, because the vendor of your laptop or the hard drive itself told you so), then you can tell the kernel to enable the usage of this feature for that drive by writing the special value `-1` to the `unload_heads` attribute:

```
# echo -1 > /sys/block/sda/device/unload_heads
```

will enable the feature for `/dev/sda`, and giving `-2` instead of `-1` will disable it again.

### 47.2.3 3. References

There are several laptops from different vendors featuring shock protection capabilities. As manufacturers have refused to support open source development of the required software components so far, Linux support for shock protection varies considerably between different hardware implementations. Ideally, this section should contain a list of pointers at different projects aiming at an implementation of shock protection on different systems. Unfortunately, I only know of a single project which, although still considered experimental, is fit for use. Please feel free to add projects that have been the victims of my ignorance.

- <http://www.thinkwiki.org/wiki/HDAPS>

See this page for information about Linux support of the hard disk active protection system as implemented in IBM/Lenovo Thinkpads.

### 47.2.4 4. CREDITS

This implementation of disk head parking has been inspired by a patch originally published by Jon Escombe <[lists@dresco.co.uk](mailto:lists@dresco.co.uk)>. My efforts to develop an implementation of this feature that is fit to be merged into mainline have been aided by various kernel developers, in particular by Tejun Heo and Bartłomiej Zolnierkiewicz.

## 47.3 How to conserve battery power using laptop-mode

Document Author: Bart Samwel ([bart@samwel.tk](mailto:bart@samwel.tk))

Date created: January 2, 2004

Last modified: December 06, 2004

### 47.3.1 Introduction

Laptop mode is used to minimize the time that the hard disk needs to be spun up, to conserve battery power on laptops. It has been reported to cause significant power savings.

### 47.3.2 Installation

To use laptop mode, you don't need to set any kernel configuration options or anything. Simply install all the files included in this document, and laptop mode will automatically be started when you're on battery. For your convenience, a tarball containing an installer can be downloaded at:

[http://www.samwel.tk/laptop\\_mode/laptop\\_mode/](http://www.samwel.tk/laptop_mode/laptop_mode/)

To configure laptop mode, you need to edit the configuration file, which is located in `/etc/default/laptop-mode` on Debian-based systems, or in `/etc/sysconfig/laptop-mode` on other systems.

Unfortunately, automatic enabling of laptop mode does not work for laptops that don't have ACPI. On those laptops, you need to start laptop mode manually. To start laptop mode, run `“laptop_mode start”`, and to stop it, run `“laptop_mode stop”`. (Note: The laptop mode tools package now has experimental support for APM, you might want to try that first.)

### 47.3.3 Caveats

- The downside of laptop mode is that you have a chance of losing up to 10 minutes of work. If you cannot afford this, don't use it! The supplied ACPI scripts automatically turn off laptop mode when the battery almost runs out, so that you won't lose any data at the end of your battery life.
- Most desktop hard drives have a very limited lifetime measured in spindown cycles, typically about 50.000 times (it's usually listed on the spec sheet). Check your drive's rating, and don't wear down your drive's lifetime if you don't need to.
- If you mount some of your ext3/reiserfs filesystems with the `-n` option, then the control script will not be able to remount them correctly. You must set `DO_REMOUNTS=0` in the control script, otherwise it will remount them with the wrong options - or it will fail because it cannot write to `/etc/mstab`.
- If you have your filesystems listed as type `“auto”` in `fstab`, like I did, then the control script will not recognize them as filesystems that need remounting. You must list the filesystems with their true type instead.
- It has been reported that some versions of the mutt mail client use file access times to determine whether a folder contains new mail. If you use mutt and experience this, you must disable the `noatime` remounting by setting the option `DO_REMOUNT_NOATIME` to 0 in the configuration file.

### 47.3.4 The Details

Laptop mode is controlled by the knob `/proc/sys/vm/laptop_mode`. This knob is present for all kernels that have the laptop mode patch, regardless of any configuration options. When the knob is set, any physical disk I/O (that might have caused the hard disk to spin up) causes Linux to flush all dirty blocks. The result of this is that after a disk has spun down, it will not be spun up anymore to write dirty blocks, because those blocks had already been written immediately after the most recent read operation. The value of the `laptop_mode` knob determines the time between the occurrence of disk I/O and when the flush is triggered. A sensible value for the knob is 5 seconds. Setting the knob to 0 disables laptop mode.

To increase the effectiveness of the `laptop_mode` strategy, the `laptop_mode` control script increases `dirty_expire_centisecs` and `dirty_writeback_centisecs` in `/proc/sys/vm` to about 10 minutes (by default), which means that pages that are dirtied are not forced to be written to disk as often. The control script also changes the dirty background ratio, so that background writeback of dirty pages is not done anymore. Combined with a higher commit value (also 10 minutes) for ext3 or ReiserFS filesystems (also done automatically by the control script), this results in concentration of disk activity in a small time interval which occurs only once every 10 minutes, or whenever the disk is forced to spin up by a cache miss. The disk can then be spun down in the periods of inactivity.

If you want to find out which process caused the disk to spin up, you can gather information by setting the flag `/proc/sys/vm/block_dump`. When this flag is set, Linux reports all disk read and write operations that take place, and all block dirtyings done to files. This makes it possible to debug why a disk needs to spin up, and to increase battery life even more. The output of `block_dump` is written to the kernel output, and it can be retrieved using “`dmesg`”. When you use `block_dump` and your kernel logging level also includes kernel debugging messages, you probably want to turn off `klogd`, otherwise the output of `block_dump` will be logged, causing disk activity that is not normally there.

### 47.3.5 Configuration

The laptop mode configuration file is located in `/etc/default/laptop-mode` on Debian-based systems, or in `/etc/sysconfig/laptop-mode` on other systems. It contains the following options:

**MAX\_AGE:**

Maximum time, in seconds, of hard drive spindown time that you are comfortable with. Worst case, it's possible that you could lose this amount of work if your battery fails while you're in laptop mode.

**MINIMUM\_BATTERY\_MINUTES:**

Automatically disable laptop mode if the remaining number of minutes of battery power is less than this value. Default is 10 minutes.

**AC\_HD/BATT\_HD:**

The idle timeout that should be set on your hard drive when laptop mode is active (`BATT_HD`) and when it is not active (`AC_HD`). The defaults are 20 seconds (value

4) for BATT\_HD and 2 hours (value 244) for AC\_HD. The possible values are those listed in the manual page for “hdparm” for the “-S” option.

HD:

The devices for which the spindown timeout should be adjusted by laptop mode. Default is /dev/hda. If you specify multiple devices, separate them by a space.

READAHEAD:

Disk readahead, in 512-byte sectors, while laptop mode is active. A large readahead can prevent disk accesses for things like executable pages (which are loaded on demand while the application executes) and sequentially accessed data (MP3s).

DO\_REMOUNTS:

The control script automatically remounts any mounted journaled filesystems with appropriate commit interval options. When this option is set to 0, this feature is disabled.

DO\_REMOUNT\_NOATIME:

When remounting, should the filesystems be remounted with the noatime option? Normally, this is set to “1” (enabled), but there may be programs that require access time recording.

DIRTY\_RATIO:

The percentage of memory that is allowed to contain “dirty” or unsaved data before a writeback is forced, while laptop mode is active. Corresponds to the /proc/sys/vm/dirty\_ratio sysctl.

DIRTY\_BACKGROUND\_RATIO:

The percentage of memory that is allowed to contain “dirty” or unsaved data after a forced writeback is done due to an exceeding of DIRTY\_RATIO. Set this nice and low. This corresponds to the /proc/sys/vm/dirty\_background\_ratio sysctl.

Note that the behaviour of dirty\_background\_ratio is quite different when laptop mode is active and when it isn't. When laptop mode is inactive, dirty\_background\_ratio is the threshold percentage at which background write-outs start taking place. When laptop mode is active, however, background write-outs are disabled, and the dirty\_background\_ratio only determines how much writeback is done when dirty\_ratio is reached.

DO\_CPU:

Enable CPU frequency scaling when in laptop mode. (Requires CPUFreq to be setup. See Documentation/admin-guide/pm/cpufreq.rst for more info. Disabled by default.)

CPU\_MAXFREQ:

When on battery, what is the maximum CPU speed that the system should use? Legal values are “slowest” for the slowest speed that your CPU is able to operate at, or a value listed in /sys/devices/system/cpu/cpu0/cpufreq/scaling\_available\_frequencies.

### 47.3.6 Tips & Tricks

- Bartek Kania reports getting up to 50 minutes of extra battery life (on top of his regular 3 to 3.5 hours) using a spindown time of 5 seconds (BATT\_HD=1).
- You can spin down the disk while playing MP3, by setting disk readahead to 8MB (READAHEAD=16384). Effectively, the disk will read a complete MP3 at once, and will then spin down while the MP3 is playing. (Thanks to Bartek Kania.)
- Drew Scott Daniels observed: “I don’ t know why, but when I decrease the number of colours that my display uses it consumes less battery power. I’ ve seen this on powerbooks too. I hope that this is a piece of information that might be useful to the Laptop Mode patch or its users.”
- In syslog.conf, you can prefix entries with a dash - to omit syncing the file after every logging. When you’ re using laptop-mode and your disk doesn’ t spin down, this is a likely culprit.
- Richard Atterer observed that laptop mode does not work well with noflushd (<http://noflushd.sourceforge.net/>), it seems that noflushd prevents laptop-mode from doing its thing.
- If you’ re worried about your data, you might want to consider using a USB memory stick or something like that as a “working area” . (Be aware though that flash memory can only handle a limited number of writes, and overuse may wear out your memory stick pretty quickly. Do not use journalling filesystems on flash memory sticks.)

### 47.3.7 Configuration file for control and ACPI battery scripts

This allows the tunables to be changed for the scripts via an external configuration file

It should be installed as /etc/default/laptop-mode on Debian, and as /etc/sysconfig/laptop-mode on Red Hat, SUSE, Mandrake, and other work-alikes.

Config file:

```
# Maximum time, in seconds, of hard drive spindown time that you are
# comfortable with. Worst case, it's possible that you could lose this
# amount of work if your battery fails you while in laptop mode.
#MAX_AGE=600

# Automatically disable laptop mode when the number of minutes of battery
# that you have left goes below this threshold.
MINIMUM_BATTERY_MINUTES=10

# Read-ahead, in 512-byte sectors. You can spin down the disk while
# playing MP3/OGG
# by setting the disk readahead to 8MB (READAHEAD=16384). Effectively, the
# disk
# will read a complete MP3 at once, and will then spin down while the MP3/
# OGG is
# playing.
```

(continues on next page)

(continued from previous page)

```
#READAHEAD=4096

# Shall we remount journaled fs. with appropriate commit interval? (1=yes)
#DO_REMOUNTS=1

# And shall we add the "noatime" option to that as well? (1=yes)
#DO_REMOUNT_NOATIME=1

# Dirty synchronous ratio. At this percentage of dirty pages the process
# which
# calls write() does its own writeback
#DIRTY_RATIO=40

#
# Allowed dirty background ratio, in percent. Once DIRTY_RATIO has been
# exceeded, the kernel will wake flusher threads which will then reduce the
# amount of dirty memory to dirty_background_ratio. Set this nice and low,
# so once some writeout has commenced, we do a lot of it.
#
#DIRTY_BACKGROUND_RATIO=5

# kernel default dirty buffer age
#DEF_AGE=30
#DEF_UPDATE=5
#DEF_DIRTY_BACKGROUND_RATIO=10
#DEF_DIRTY_RATIO=40
#DEF_XFS_AGE_BUFFER=15
#DEF_XFS_SYNC_INTERVAL=30
#DEF_XFS_BUF_INTERVAL=1

# This must be adjusted manually to the value of HZ in the running kernel
# on 2.4, until the XFS people change their 2.4 external interfaces to
↳work in
# centisecs. This can be automated, but it's a work in progress that still
# needs some fixes. On 2.6 kernels, XFS uses USER_HZ instead of HZ for
# external interfaces, and that is currently always set to 100. So you don
↳t
# need to change this on 2.6.
#XFS_HZ=100

# Should the maximum CPU frequency be adjusted down while on battery?
# Requires CPUFreq to be setup.
# See Documentation/admin-guide/pm/cpufreq.rst for more info
#DO_CPU=0

# When on battery what is the maximum CPU speed that the system should
# use? Legal values are "slowest" for the slowest speed that your
# CPU is able to operate at, or a value listed in:
# /sys/devices/system/cpu/cpu0/cpufreq/scaling_available_frequencies
# Only applicable if DO_CPU=1.
#CPU_MAXFREQ=slowest

# Idle timeout for your hard drive (man hdparm for valid values, -S option)
# Default is 2 hours on AC (AC_HD=244) and 20 seconds for battery (BATT_
↳HD=4).
#AC_HD=244
```

(continues on next page)

(continued from previous page)

```
#BATT_HD=4

# The drives for which to adjust the idle timeout. Separate them by a
↳space,
# e.g. HD="/dev/hda /dev/hdb".
#HD="/dev/hda"

# Set the spindown timeout on a hard drive?
#DO_HD=1
```

### 47.3.8 Control script

Please note that this control script works for the Linux 2.4 and 2.6 series (thanks to Kiko Piris).

Control script:

```
#!/bin/bash

# start or stop laptop_mode, best run by a power management daemon when
# ac gets connected/disconnected from a laptop
#
# install as /sbin/laptop_mode
#
# Contributors to this script:  Kiko Piris
#                               Bart Samwel
#                               Micha Feigin
#                               Andrew Morton
#                               Herve Eychenne
#                               Dax Kelson
#
# Original Linux 2.4 version by: Jens Axboe

#####
↳##

# Source config
if [ -f /etc/default/laptop-mode ] ; then
    # Debian
    . /etc/default/laptop-mode
elif [ -f /etc/sysconfig/laptop-mode ] ; then
    # Others
    . /etc/sysconfig/laptop-mode
fi

# Don't raise an error if the config file is incomplete
# set defaults instead:

# Maximum time, in seconds, of hard drive spindown time that you are
# comfortable with. Worst case, it's possible that you could lose this
# amount of work if your battery fails you while in laptop mode.
MAX_AGE=${MAX_AGE:-'600'}

# Read-ahead, in kilobytes
```

(continues on next page)

(continued from previous page)

```

READAHEAD=${READAHEAD:-'4096'}

# Shall we remount journaled fs. with appropriate commit interval? (1=yes)
DO_REMOUNTS=${DO_REMOUNTS:-'1'}

# And shall we add the "noatime" option to that as well? (1=yes)
DO_REMOUNT_NOATIME=${DO_REMOUNT_NOATIME:-'1'}

# Shall we adjust the idle timeout on a hard drive?
DO_HD=${DO_HD:-'1'}

# Adjust idle timeout on which hard drive?
HD="${HD:-'/dev/hda'}"

# spindown time for HD (hdparm -S values)
AC_HD=${AC_HD:-'244'}
BATT_HD=${BATT_HD:-'4'}

# Dirty synchronous ratio. At this percentage of dirty pages the process_
↳which
# calls write() does its own writeback
DIRTY_RATIO=${DIRTY_RATIO:-'40'}

# cpu frequency scaling
# See Documentation/admin-guide/pm/cpufreq.rst for more info
DO_CPU=${CPU_MANAGE:-'0'}
CPU_MAXFREQ=${CPU_MAXFREQ:-'slowest'}

#
# Allowed dirty background ratio, in percent. Once DIRTY_RATIO has been
# exceeded, the kernel will wake flusher threads which will then reduce the
# amount of dirty memory to dirty_background_ratio. Set this nice and low,
# so once some writeout has commenced, we do a lot of it.
#
DIRTY_BACKGROUND_RATIO=${DIRTY_BACKGROUND_RATIO:-'5'}

# kernel default dirty buffer age
DEF_AGE=${DEF_AGE:-'30'}
DEF_UPDATE=${DEF_UPDATE:-'5'}
DEF_DIRTY_BACKGROUND_RATIO=${DEF_DIRTY_BACKGROUND_RATIO:-'10'}
DEF_DIRTY_RATIO=${DEF_DIRTY_RATIO:-'40'}
DEF_XFS_AGE_BUFFER=${DEF_XFS_AGE_BUFFER:-'15'}
DEF_XFS_SYNC_INTERVAL=${DEF_XFS_SYNC_INTERVAL:-'30'}
DEF_XFS_BUFD_INTERVAL=${DEF_XFS_BUFD_INTERVAL:-'1'}

# This must be adjusted manually to the value of HZ in the running kernel
# on 2.4, until the XFS people change their 2.4 external interfaces to_
↳work in
# centisecs. This can be automated, but it's a work in progress that still_
↳needs
# some fixes. On 2.6 kernels, XFS uses USER_HZ instead of HZ for external
# interfaces, and that is currently always set to 100. So you don't need to
# change this on 2.6.
XFS_HZ=${XFS_HZ:-'100'}

```

```

#####
↳##

```

(continues on next page)

(continued from previous page)

```

KLEVEL="$(uname -r |
    {
        IFS='.' read a b c
        echo $a.$b
    }
)"
case "$KLEVEL" in
    "2.4"|"2.6")
        ;;
    *)
        echo "Unhandled kernel version: $KLEVEL ('uname -r' = '
→$(uname -r)')" >&2
        exit 1
        ;;
esac

if [ ! -e /proc/sys/vm/laptop_mode ] ; then
    echo "Kernel is not patched with laptop_mode patch." >&2
    exit 1
fi

if [ ! -w /proc/sys/vm/laptop_mode ] ; then
    echo "You do not have enough privileges to enable laptop_mode." >&2
    exit 1
fi

# Remove an option (the first parameter) of the form option=<number> from
# a mount options string (the rest of the parameters).
parse_mount_opts () {
    OPT="$1"
    shift
    echo ",$*," | sed \
        -e 's/,,"$OPT"=[0-9]*,/,/g' \
        -e 's/,,*/,/g' \
        -e 's/^,/' \
        -e 's/,,$/'
}

# Remove an option (the first parameter) without any arguments from
# a mount option string (the rest of the parameters).
parse_nonumber_mount_opts () {
    OPT="$1"
    shift
    echo ",$*," | sed \
        -e 's/,,"$OPT"',/,/g' \
        -e 's/,,*/,/g' \
        -e 's/^,/' \
        -e 's/,,$/'
}

# Find out the state of a yes/no option (e.g. "atime"/"noatime") in
# fstab for a given filesystem, and use this state to replace the
# value of the option in another mount options string. The device
# is the first argument, the option name the second, and the default
# value the third. The remainder is the mount options string.

```

(continues on next page)

(continued from previous page)

```

#
# Example:
# parse_yesno_opts_wfstab /dev/hda1 atime atime defaults,noatime
#
# If fstab contains, say, "rw" for this filesystem, then the result
# will be "defaults,atime".
parse_yesno_opts_wfstab () {
    L_DEV="$1"
    OPT="$2"
    DEF_OPT="$3"
    shift 3
    L_OPTS="$*"
    PARSEDOPTS1="$(parse_nonumber_mount_opts $OPT $L_OPTS)"
    PARSEDOPTS1="$(parse_nonumber_mount_opts no$OPT $PARSEDOPTS1)"
    # Watch for a default atime in fstab
    FSTAB_OPTS="$(awk '$1 == "'$L_DEV'" { print $4 }' /etc/fstab)"
    if echo "$FSTAB_OPTS" | grep "$OPT" > /dev/null ; then
        # option specified in fstab: extract the value and use it
        if echo "$FSTAB_OPTS" | grep "no$OPT" > /dev/null ; then
            echo "$PARSEDOPTS1,no$OPT"
        else
            # no$OPT not found -- so we must have $OPT.
            echo "$PARSEDOPTS1,$OPT"
        fi
    else
        # option not specified in fstab -- choose the default.
        echo "$PARSEDOPTS1,$DEF_OPT"
    fi
}

# Find out the state of a numbered option (e.g. "commit=NNN") in
# fstab for a given filesystem, and use this state to replace the
# value of the option in another mount options string. The device
# is the first argument, and the option name the second. The
# remainder is the mount options string in which the replacement
# must be done.
#
# Example:
# parse_mount_opts_wfstab /dev/hda1 commit defaults,commit=7
#
# If fstab contains, say, "commit=3,rw" for this filesystem, then the
# result will be "rw,commit=3".
parse_mount_opts_wfstab () {
    L_DEV="$1"
    OPT="$2"
    shift 2
    L_OPTS="$*"
    PARSEDOPTS1="$(parse_mount_opts $OPT $L_OPTS)"
    # Watch for a default commit in fstab
    FSTAB_OPTS="$(awk '$1 == "'$L_DEV'" { print $4 }' /etc/fstab)"
    if echo "$FSTAB_OPTS" | grep "$OPT=" > /dev/null ; then
        # option specified in fstab: extract the value, and use it
        echo -n "$PARSEDOPTS1,$OPT="
        echo ",$FSTAB_OPTS," | sed \
            -e 's/.*,'"$OPT"'=//' \
            -e 's/,.*//'
    fi
}

```

(continues on next page)

(continued from previous page)

```

else
    # option not specified in fstab: set it to 0
    echo "$PARSED_OPTS1,$OPT=0"
fi
}

deduce_fstype () {
    MP="$1"
    # My root filesystem unfortunately has
    # type "unknown" in /etc/mtab. If we encounter
    # "unknown", we try to get the type from fstab.
    cat /etc/fstab |
    grep -v '^#' |
    while read FSTAB_DEV FSTAB_MP FSTAB_FST FSTAB_OPTS FSTAB_DUMP FSTAB_
→DUMP ; do
        if [ "$FSTAB_MP" = "$MP" ]; then
            echo $FSTAB_FST
            exit 0
        fi
    done
}

if [ $DO_REMOUNT_NOATIME -eq 1 ] ; then
    NOATIME_OPT=",noatime"
fi

case "$1" in
    start)
        AGE=$((100*$MAX_AGE))
        XFS_AGE=$((XFS_HZ*$MAX_AGE))
        echo -n "Starting laptop_mode"

        if [ -d /proc/sys/vm/pagebuf ] ; then
            # (For 2.4 and early 2.6.)
            # This only needs to be set, not reset -- it is only_
→used when
            # laptop mode is enabled.
            echo $XFS_AGE > /proc/sys/vm/pagebuf/lm_flush_age
            echo $XFS_AGE > /proc/sys/fs/xfs/lm_sync_interval
        elif [ -f /proc/sys/fs/xfs/lm_age_buffer ] ; then
            # (A couple of early 2.6 laptop mode patches had_
→these.)
            # The same goes for these.
            echo $XFS_AGE > /proc/sys/fs/xfs/lm_age_buffer
            echo $XFS_AGE > /proc/sys/fs/xfs/lm_sync_interval
        elif [ -f /proc/sys/fs/xfs/age_buffer ] ; then
            # (2.6.6)
            # But not for these -- they are also used in normal
            # operation.
            echo $XFS_AGE > /proc/sys/fs/xfs/age_buffer
            echo $XFS_AGE > /proc/sys/fs/xfs/sync_interval
        elif [ -f /proc/sys/fs/xfs/age_buffer_centisecs ] ; then
            # (2.6.7 upwards)
            # And not for these either. These are in centisecs,
            # not USER_HZ, so we have to use $AGE, not $XFS_AGE.
            echo $AGE > /proc/sys/fs/xfs/age_buffer_centisecs

```

(continues on next page)

(continued from previous page)

```

        echo $AGE > /proc/sys/fs/xfs/xfssyncd_centisecs
        echo 3000 > /proc/sys/fs/xfs/xfsbufd_centisecs
    fi

    case "$KLEVEL" in
        "2.4")
            echo 1 > /proc/sys/vm/laptop_mode
            echo "30 500 0 0 $AGE $AGE 60 20 0" > /proc/sys/vm/bdflush
            ;;
        "2.6")
            echo 5 > /proc/sys/vm/laptop_mode
            echo "$AGE" > /proc/sys/vm/dirty_writeback_centisecs
            echo "$AGE" > /proc/sys/vm/dirty_expire_centisecs
            echo "$DIRTY_RATIO" > /proc/sys/vm/dirty_ratio
            echo "$DIRTY_BACKGROUND_RATIO" > /proc/sys/vm/dirty_background_ratio
            ;;
    esac
    if [ $DO_REMOUNTS -eq 1 ]; then
        cat /etc/mtab | while read DEV MP FST OPTS DUMP PASSW
    ; do
        PARSEDOPTS="$(parse_mount_opts "$OPTS")"
        if [ "$FST" = 'unknown' ]; then
            FST=$(deduce_fstype $MP)
        fi
        case "$FST" in
            "ext3"|"reiserfs")
                PARSEDOPTS="$(parse_mount_
    opts commit "$OPTS")"
                mount $DEV -t $FST $MP -o
    remount,$PARSEDOPTS,commit=$MAX_AGE$NOATIME_
    OPT
                ;;
            "xfs")
                mount $DEV -t $FST $MP -o
    remount,$OPTS$NOATIME_
    OPT
                ;;
        esac
        if [ -b $DEV ] ; then
            blockdev --setra $(( $READAHEAD * 2 ))
        fi
    done
fi
if [ $DO_HD -eq 1 ] ; then
    for THISHD in $HD ; do
        /sbin/hdparm -S $BATT_HD $THISHD > /dev/null
    2>&1
        /sbin/hdparm -B 1 $THISHD > /dev/null 2>&1
    done
fi

```

(continues on next page)

(continued from previous page)

```

        if [ $DO_CPU -eq 1 -a -e /sys/devices/system/cpu/cpu0/
→cpufreq/cpuinfo_min_freq ]; then
            if [ $CPU_MAXFREQ = 'slowest' ]; then
                CPU_MAXFREQ=`cat /sys/devices/system/cpu/
→cpu0/cpufreq/cpuinfo_min_freq`
            fi
            echo $CPU_MAXFREQ > /sys/devices/system/cpu/cpu0/
→cpufreq/scaling_max_freq
        fi
        echo "."
        ;;
    stop)
        U_AGE=$((100*$DEF_UPDATE))
        B_AGE=$((100*$DEF_AGE))
        echo -n "Stopping laptop_mode"
        echo 0 > /proc/sys/vm/laptop_mode
        if [ -f /proc/sys/fs/xfs/age_buffer -a ! -f /proc/sys/fs/xfs/
→lm_age_buffer ] ; then
            # These need to be restored, if there are no lm_*.
            echo $((($XFS_HZ*$DEF_XFS_AGE_BUFFER)) > /
→proc/sys/fs/xfs/age_buffer
            echo $((($XFS_HZ*$DEF_XFS_SYNC_INTERVAL)) > /
→proc/sys/fs/xfs/sync_interval
        elif [ -f /proc/sys/fs/xfs/age_buffer_centisecs ] ; then
            # These need to be restored as well.
            echo $((100*$DEF_XFS_AGE_BUFFER)) > /proc/sys/
→fs/xfs/age_buffer_centisecs
            echo $((100*$DEF_XFS_SYNC_INTERVAL)) > /proc/sys/
→fs/xfs/xfssyncd_centisecs
            echo $((100*$DEF_XFS_BUFD_INTERVAL)) > /proc/sys/
→fs/xfs/xfsbufd_centisecs
        fi
        case "$KLEVEL" in
            "2.4")
                echo "30 500 0 0 $U_AGE $B_AGE 60 20 0" > /
→proc/sys/vm/bdflush
                ;;
            "2.6")
                echo "$U_AGE" > /
→proc/sys/vm/dirty_writeback_centisecs
                echo "$B_AGE" > /
→proc/sys/vm/dirty_expire_centisecs
                echo "$DEF_DIRTY_RATIO" > /
→proc/sys/vm/dirty_ratio
                echo "$DEF_DIRTY_BACKGROUND_RATIO" > /
→proc/sys/vm/dirty_background_ratio
                ;;
        esac
        if [ $DO_REMOUNTS -eq 1 ] ; then
            cat /etc/mtab | while read DEV MP FST OPTS DUMP PASS_
→; do
                # Reset commit and atime options to defaults.
                if [ "$FST" = 'unknown' ]; then
                    FST=$(deduce_fstype $MP)
                fi
                case "$FST" in

```

(continues on next page)

(continued from previous page)

```

                                "ext3"|"reiserfs")
                                PARSEDOPTS="$(parse_mount_
↪opts_wfstab $DEV commit $OPTS)"
                                PARSEDOPTS="$(parse_yesno_
↪opts_wfstab $DEV atime atime $PARSEDOPTS)"
                                mount $DEV -t $FST $MP -o_
↪remount,$PARSEDOPTS
                                ;;
                                "xfs")
                                PARSEDOPTS="$(parse_yesno_
↪opts_wfstab $DEV atime atime $OPTS)"
                                mount $DEV -t $FST $MP -o_
↪remount,$PARSEDOPTS
                                ;;
                                esac
                                if [ -b $DEV ] ; then
                                    blockdev --setra 256 $DEV
                                fi
                                done
                                fi
                                if [ $DO_HD -eq 1 ] ; then
                                    for THISHD in $HD ; do
↪&1                                /sbin/hdparm -S $AC_HD $THISHD > /dev/null 2>
                                    /sbin/hdparm -B 255 $THISHD > /dev/null 2>&1
                                done
                                fi
                                if [ $DO_CPU -eq 1 -a -e /sys/devices/system/cpu/cpu0/
↪cpufreq/cpuinfo_min_freq ] ; then
                                    echo `cat /sys/devices/system/cpu/cpu0/cpufreq/
↪cpuinfo_max_freq` > /sys/devices/system/cpu/cpu0/cpufreq/scaling_max_freq
                                fi
                                echo "."
                                ;;
                                *)
                                    echo "Usage: $0 {start|stop}" 2>&1
                                    exit 1
                                ;;
                                esac
                                exit 0

```

### 47.3.9 ACPI integration

Dax Kelson submitted this so that the ACPI acpid daemon will kick off the laptop\_mode script and run hdparm. The part that automatically disables laptop mode when the battery is low was written by Jan Topinski.

/etc/acpi/events/ac\_adapter:

```

event=ac_adapter
action=/etc/acpi/actions/ac.sh %e

```

/etc/acpi/events/battery:

```
event=battery.*
action=/etc/acpi/actions/battery.sh %e
```

/etc/acpi/actions/ac.sh:

```
#!/bin/bash

# ac on/offline event handler

status=`awk '/^state: / { print $2 }' /proc/acpi/ac_adapter/$2/state`

case $status in
    "on-line")
        /sbin/laptop_mode stop
        exit 0
    ;;
    "off-line")
        /sbin/laptop_mode start
        exit 0
    ;;
esac
```

/etc/acpi/actions/battery.sh:

```
#!/bin/bash

# Automatically disable laptop mode when the battery almost runs out.

BATT_INFO=/proc/acpi/battery/$2/state

if [[ -f /proc/sys/vm/laptop_mode ]]
then
    LM=`cat /proc/sys/vm/laptop_mode`
    if [[ $LM -gt 0 ]]
    then
        if [[ -f $BATT_INFO ]]
        then
            # Source the config file only now that we know we need
            if [ -f /etc/default/laptop-mode ] ; then
                # Debian
                . /etc/default/laptop-mode
            elif [ -f /etc/sysconfig/laptop-mode ] ; then
                # Others
                . /etc/sysconfig/laptop-mode
            fi
            MINIMUM_BATTERY_MINUTES=${MINIMUM_BATTERY_MINUTES:-'10'}

            ACTION=""`cat $BATT_INFO | grep charging | cut -c 26-`"
            if [[ ACTION -eq "discharging" ]]
            then
                PRESENT_RATE=`cat $BATT_INFO | grep "present rate:" | sed "s/
→* \([0-9][0-9]* \)*/\1/"`
                REMAINING=`cat $BATT_INFO | grep "remaining capacity:" | sed
→"s/.* \([0-9][0-9]* \)*/\1/"`
                fi
                if (($REMAINING * 60 / $PRESENT_RATE < $MINIMUM_BATTERY_MINUTES))
```

(continues on next page)

(continued from previous page)

```
        then
            /sbin/laptop_mode stop
        fi
    else
        logger -p daemon.warning "You are using laptop mode and your
↳battery interface $BATT_INFO is missing. This may lead to loss of data
↳when the battery runs out. Check kernel ACPI support and /proc/acpi/
↳battery folder, and edit /etc/acpi/battery.sh to set BATT_INFO to the
↳correct path."
    fi
fi
fi
```

### 47.3.10 Monitoring tool

Bartek Kania submitted this, it can be used to measure how much time your disk spends spun up/down. See `tools/laptop/dslm/dslm.c`

## 47.4 LG Gram laptop extra features

By Matan Ziv-Av <[matan@svgalib.org](mailto:matan@svgalib.org)>

### 47.4.1 Hotkeys

The following FN keys are ignored by the kernel without this driver:

- FN-F1 (LG control panel) - Generates F15
- FN-F5 (Touchpad toggle) - Generates F13
- FN-F6 (Airplane mode) - Generates RFKILL
- FN-F8 (Keyboard backlight) - Generates F16. This key also changes keyboard backlight mode.
- FN-F9 (Reader mode) - Generates F14

The rest of the FN keys work without a need for a special driver.

### 47.4.2 Reader mode

Writing 0/1 to `/sys/devices/platform/lg-laptop/reader_mode` disables/enables reader mode. In this mode the screen colors change (blue color reduced), and the reader mode indicator LED (on F9 key) turns on.

### 47.4.3 FN Lock

Writing 0/1 to `/sys/devices/platform/lg-laptop/fn_lock` disables/enables FN lock.

### 47.4.4 Battery care limit

Writing 80/100 to `/sys/devices/platform/lg-laptop/battery_care_limit` sets the maximum capacity to charge the battery. Limiting the charge reduces battery capacity loss over time.

This value is reset to 100 when the kernel boots.

### 47.4.5 Fan mode

Writing 1/0 to `/sys/devices/platform/lg-laptop/fan_mode` disables/enables the fan silent mode.

### 47.4.6 USB charge

Writing 0/1 to `/sys/devices/platform/lg-laptop/usb_charge` disables/enables charging another device from the USB port while the device is turned off.

This value is reset to 0 when the kernel boots.

## LEDs

There are two LED devices supported by the driver:

### 47.4.7 Keyboard backlight

A led device named `kbd_led` controls the keyboard backlight. There are three lighting levels: off (0), low (127) and high (255).

The keyboard backlight is also controlled by the key combination FN-F8 which cycles through those levels.

### 47.4.8 Touchpad indicator LED

On the F5 key. Controlled by led device name `tpad_led`.

## 47.5 Sony Notebook Control Driver (SNC) Readme

- Copyright (C) 2004- 2005 Stelian Pop <stelian@popies.net>
- Copyright (C) 2007 Mattia Dongili <malattia@linux.it>

This mini-driver drives the SNC and SPIC device present in the ACPI BIOS of the Sony Vaio laptops. This driver mixes both devices functions under the same (hopefully consistent) interface. This also means that the sonypi driver is obsoleted by sony-laptop now.

### 47.5.1 Fn keys (hotkeys):

Some models report hotkeys through the SNC or SPIC devices, such events are reported both through the ACPI subsystem as acpi events and through the INPUT subsystem. See the logs of `/proc/bus/input/devices` to find out what those events are and which input devices are created by the driver. Additionally, loading the driver with the debug option will report all events in the kernel log.

The “scancodes” passed to the input system (that can be remapped with udev) are indexes to the table “`sony_laptop_input_keycode_map`” in the `sony-laptop.c` module. For example the “FN/E” key combination (EJECTCD on some models) generates the scancode 20 (0x14).

### 47.5.2 Backlight control:

If your laptop model supports it, you will find sysfs files in the `/sys/class/backlight/sony/` directory. You will be able to query and set the current screen brightness:

<code>brightness</code>	get/set screen brightness (an integer between 0 and 7)
<code>actual_brightness</code>	reading from this file will query the HW to get real brightness value
<code>max_brightness</code>	the maximum brightness value

### 47.5.3 Platform specific:

Loading the `sony-laptop` module will create a `/sys/devices/platform/sony-laptop/` directory populated with some files.

You then read/write integer values from/to those files by using standard UNIX tools.

The files are:

bright-ness_default	screen brightness which will be set when the laptop will be rebooted
cdpower	power on/off the internal CD drive
audiopower	power on/off the internal sound card
lanpower	power on/off the internal ethernet card (only in debug mode)
bluetooth-power	power on/off the internal bluetooth device
fanspeed	get/set the fan speed

Note that some files may be missing if they are not supported by your particular laptop model.

Example usage:

```
# echo "1" > /sys/devices/platform/sony-laptop/brightness_default
```

sets the lowest screen brightness for the next and later reboots

```
# echo "8" > /sys/devices/platform/sony-laptop/brightness_default
```

sets the highest screen brightness for the next and later reboots

```
# cat /sys/devices/platform/sony-laptop/brightness_default
```

retrieves the value

```
# echo "0" > /sys/devices/platform/sony-laptop/audiopower
```

powers off the sound card

```
# echo "1" > /sys/devices/platform/sony-laptop/audiopower
```

powers on the sound card.

### 47.5.4 RFkill control:

More recent Vaio models expose a consistent set of ACPI methods to control radio frequency emitting devices. If you are a lucky owner of such a laptop you will find the necessary rfkill devices under `/sys/class/rfkill`. Check those starting with `sony-*` in:

```
# grep . /sys/class/rfkill/*/{state,name}
```

### 47.5.5 Development:

If you want to help with the development of this driver (and you are not afraid of any side effects doing strange things with your ACPI BIOS could have on your laptop), load the driver and pass the option `'debug=1'` .

**REPEAT: DON' T DO THIS IF YOU DON' T LIKE RISKY BUSINESS.**

In your kernel logs you will find the list of all ACPI methods the SNC device has on your laptop.

- For new models you will see a long list of meaningless method names, reading the DSDT table source should reveal that:

- (1) the SNC device uses an internal capability lookup table
- (2) SN00 is used to find values in the lookup table
- (3) SN06 and SN07 are used to call into the real methods based on offsets you can obtain iterating the table using SN00
- (4) SN02 used to enable events.

Some values in the capability lookup table are more or less known, see the code for all `sony_call_snc_handle` calls, others are more obscure.

- For old models you can see the GCDP/GCDP methods used to pwer on/off the CD drive, but there are others and they are usually different from model to model.

**I HAVE NO IDEA WHAT THOSE METHODS DO.**

The sony-laptop driver creates, for some of those methods (the most current ones found on several Vaio models), an entry under `/sys/devices/platform/sony-laptop`, just like the `'cdpower'` one. You can create other entries corresponding to your own laptop methods by further editing the source (see the `'sony_nc_values'` table, and add a new entry to this table with your get/set method names using the `SNC_HANDLE_NAMES` macro).

Your mission, should you accept it, is to try finding out what those entries are for, by reading/writing random values from/to those files and find out what is the impact on your laptop.

Should you find anything interesting, please report it back to me, I will not disavow all knowledge of your actions :)

See also [http://www.linux.it/~malattia/wiki/index.php/Sony\\_drivers](http://www.linux.it/~malattia/wiki/index.php/Sony_drivers) for other useful info.

### 47.5.6 Bugs/Limitations:

- This driver is not based on official documentation from Sony (because there is none), so there is no guarantee this driver will work at all, or do the right thing. Although this hasn't happened to me, this driver could do very bad things to your laptop, including permanent damage.
- The sony-laptop and sonypi drivers do not interact at all. In the future, sonypi will be removed and replaced by sony-laptop.
- spicctrl, which is the userspace tool used to communicate with the sonypi driver (through /dev/sonypi) is deprecated as well since all its features are now available under the sysfs tree via sony-laptop.

## 47.6 Sony Programmable I/O Control Device Driver Readme

- Copyright (C) 2001-2004 Stelian Pop <[stelian@popies.net](mailto:stelian@popies.net)>
- Copyright (C) 2001-2002 Alcôve <[www.alcove.com](http://www.alcove.com)>
- Copyright (C) 2001 Michael Ashley <[m.ashley@unsw.edu.au](mailto:m.ashley@unsw.edu.au)>
- Copyright (C) 2001 Junichi Morita <[jun1m@mars.dti.ne.jp](mailto:jun1m@mars.dti.ne.jp)>
- Copyright (C) 2000 Takaya Kinjo <[t-kinjo@tc4.so-net.ne.jp](mailto:t-kinjo@tc4.so-net.ne.jp)>
- Copyright (C) 2000 Andrew Tridgell <[tridge@samba.org](mailto:tridge@samba.org)>

This driver enables access to the Sony Programmable I/O Control Device which can be found in many Sony Vaio laptops. Some newer Sony laptops (seems to be limited to new FX series laptops, at least the FX501 and the FX702) lack a sonypi device and are not supported at all by this driver.

It will give access (through a user space utility) to some events those laptops generate, like:

- jogdial events (the small wheel on the side of Vaios)
- capture button events (only on Vaio Picturebook series)
- Fn keys
- bluetooth button (only on C1VR model)
- programmable keys, back, help, zoom, thumbphrase buttons, etc. (when available)

Those events (see `linux/sonypi.h`) can be polled using the character device node `/dev/sonypi` (major 10, minor auto allocated or specified as a option). A simple daemon which translates the jogdial movements into mouse wheel events can be downloaded at: <<http://popies.net/sonypi/>>

Another option to intercept the events is to get them directly through the input layer.

This driver supports also some ioctl commands for setting the LCD screen brightness and querying the batteries charge information (some more commands may be added in the future).

This driver can also be used to set the camera controls on Picturebook series (brightness, contrast etc), and is used by the video4linux driver for the Motion Eye camera.

Please note that this driver was created by reverse engineering the Windows driver and the ACPI BIOS, because Sony doesn't agree to release any programming specs for its laptops. If someone convinces them to do so, drop me a note.

### **47.6.1 Driver options:**

Several options can be passed to the sonypi driver using the standard module argument syntax (`<param>=<value>` when passing the option to the module or `sonypi.<param>=<value>` on the kernel boot line when sonypi is statically linked into the kernel). Those options are:

minor:	minor number of the misc device /dev/sonypi, default is -1 (automatic allocation, see /proc/misc or kernel logs)
camera:	if you have a PictureBook series Vaio (with the integrated Motion-Eye camera), set this parameter to 1 in order to let the driver access to the camera
fnkeyinit:	on some Vaios (C1VE, C1VR etc), the Fn key events don't get enabled unless you set this parameter to 1. Do not use this option unless it's actually necessary, some Vaio models don't deal well with this option. This option is available only if the kernel is compiled without ACPI support (since it conflicts with it and it shouldn't be required anyway if ACPI is already enabled).
verbose:	set to 1 to print unknown events received from the sonypi device. set to 2 to print all events received from the sonypi device.
compat:	uses some compatibility code for enabling the sonypi events. If the driver worked for you in the past (prior to version 1.5) and does not work anymore, add this option and report to the author.
mask:	event mask telling the driver what events will be reported to the user. This parameter is required for some Vaio models where the hardware reuses values used in other Vaio models (like the FX series who does not have a jogdial but reuses the jogdial events for programmable keys events). The default event mask is set to 0xffffffff, meaning that all possible events will be tried. You can use the following bits to construct your own event mask (from drivers/char/sonypi.h): SONYPI_JOGGER_MASK                    ↳ ↳ 0x0001 SONYPI_CAPTURE_MASK                   ↳ ↳ 0x0002 SONYPI_FNKEY_MASK                     ↳ ↳ 0x0004 SONYPI_BLUETOOTH_MASK                ↳ ↳ 0x0008 SONYPI_PKEY_MASK                      ↳ ↳ 0x0010 SONYPI_BACK_MASK

### 47.6.2 Module use:

In order to automatically load the sonypi module on use, you can put those lines a configuration file in `/etc/modprobe.d/`:

```
alias char-major-10-250 sonypi
options sonypi minor=250
```

This supposes the use of minor 250 for the sonypi device:

```
# mknod /dev/sonypi c 10 250
```

### 47.6.3 Bugs:

- several users reported that this driver disables the BIOS-managed Fn-keys which put the laptop in sleeping state, or switch the external monitor on/off. There is no workaround yet, since this driver disables all APM management for those keys, by enabling the ACPI management (and the ACPI core stuff is not complete yet). If you have one of those laptops with working Fn keys and want to continue to use them, don't use this driver.
- some users reported that the laptop speed is lower (dhrystone tested) when using the driver with the `fnkeyinit` parameter. I cannot reproduce it on my laptop and not all users have this problem. This happens because the `fnkeyinit` parameter enables the ACPI mode (but without additional ACPI control, like processor speed handling etc). Use ACPI instead of APM if it works on your laptop.
- sonypi lacks the ability to distinguish between certain key events on some models.
- some models with the nvidia card (geforce go 6200 tc) uses a different way to adjust the backlighting of the screen. There is a userspace utility to adjust the brightness on those models, which can be downloaded from <http://www.acc.umu.se/~erikw/program/smarddimmer-0.1.tar.bz2>
- since all development was done by reverse engineering, there is absolutely no guarantee that this driver will not crash your laptop. Permanently.

## 47.7 ThinkPad ACPI Extras Driver

Version 0.25

October 16th, 2013

- Borislav Deianov <[borislav@users.sf.net](mailto:borislav@users.sf.net)>
- Henrique de Moraes Holschuh <[hmh@hmh.eng.br](mailto:hmh@hmh.eng.br)>

<http://ibm-acpi.sf.net/>

This is a Linux driver for the IBM and Lenovo ThinkPad laptops. It supports various features of these laptops which are accessible through the ACPI and ACPI EC framework, but not otherwise fully supported by the generic Linux ACPI drivers.

This driver used to be named `ibm-acpi` until kernel 2.6.21 and release 0.13-20070314. It used to be in the `drivers/acpi` tree, but it was moved to the `drivers/misc` tree and renamed to `thinkpad-acpi` for kernel 2.6.22, and release 0.14. It was moved to `drivers/platform/x86` for kernel 2.6.29 and release 0.22.

The driver is named “`thinkpad-acpi`”. In some places, like module names and log messages, “`thinkpad_acpi`” is used because of userspace issues.

“`tpacpi`” is used as a shorthand where “`thinkpad-acpi`” would be too long due to length limitations on some Linux kernel versions.

### 47.7.1 Status

The features currently supported are the following (see below for detailed description):

- Fn key combinations
- Bluetooth enable and disable
- video output switching, expansion control
- ThinkLight on and off
- CMOS/UCMS control
- LED control
- ACPI sounds
- temperature sensors
- Experimental: embedded controller register dump
- LCD brightness control
- Volume control
- Fan control and monitoring: fan speed, fan enable/disable
- WAN enable and disable
- UWB enable and disable
- LCD Shadow (PrivacyGuard) enable and disable

A compatibility table by model and feature is maintained on the web site, <http://ibm-acpi.sf.net/>. I appreciate any success or failure reports, especially if they add to or correct the compatibility table. Please include the following information in your report:

- ThinkPad model name
- a copy of your ACPI tables, using the “`acpidump`” utility
- a copy of the output of `dmidecode`, with serial numbers and UUIDs masked off
- which driver features work and which don’ t
- the observed behavior of non-working features

Any other comments or patches are also more than welcome.

### 47.7.2 Installation

If you are compiling this driver as included in the Linux kernel sources, look for the CONFIG\_THINKPAD\_ACPI Kconfig option. It is located on the menu path: “Device Drivers” -> “X86 Platform Specific Device Drivers” -> “ThinkPad ACPI Laptop Extras” .

### 47.7.3 Features

The driver exports two different interfaces to userspace, which can be used to access the features it provides. One is a legacy procfs-based interface, which will be removed at some time in the future. The other is a new sysfs-based interface which is not complete yet.

The procfs interface creates the /proc/acpi/ibm directory. There is a file under that directory for each feature it supports. The procfs interface is mostly frozen, and will change very little if at all: it will not be extended to add any new functionality in the driver, instead all new functionality will be implemented on the sysfs interface.

The sysfs interface tries to blend in the generic Linux sysfs subsystems and classes as much as possible. Since some of these subsystems are not yet ready or stabilized, it is expected that this interface will change, and any and all userspace programs must deal with it.

#### Notes about the sysfs interface

Unlike what was done with the procfs interface, correctness when talking to the sysfs interfaces will be enforced, as will correctness in the thinkpad-acpi’ s implementation of sysfs interfaces.

Also, any bugs in the thinkpad-acpi sysfs driver code or in the thinkpad-acpi’ s implementation of the sysfs interfaces will be fixed for maximum correctness, even if that means changing an interface in non-compatible ways. As these interfaces mature both in the kernel and in thinkpad-acpi, such changes should become quite rare.

Applications interfacing to the thinkpad-acpi sysfs interfaces must follow all sysfs guidelines and correctly process all errors (the sysfs interface makes extensive use of errors). File descriptors and open / close operations to the sysfs inodes must also be properly implemented.

The version of thinkpad-acpi’ s sysfs interface is exported by the driver as a driver attribute (see below).

Sysfs driver attributes are on the driver’ s sysfs attribute space, for 2.6.23+ this is /sys/bus/platform/drivers/thinkpad\_acpi/ and /sys/bus/platform/drivers/thinkpad\_hwmon/

Sysfs device attributes are on the thinkpad\_acpi device sysfs attribute space, for 2.6.23+ this is /sys/devices/platform/thinkpad\_acpi/.

Sysfs device attributes for the sensors and fan are on the thinkpad\_hwmon device’ s sysfs attribute space, but you should locate it looking for a hwmon device with the name attribute of “thinkpad” , or better yet,

through libensors. For 4.14+ sysfs attributes were moved to the hwmon device (/sys/bus/platform/devices/thinkpad\_hwmon/hwmon/hwmon? or /sys/class/hwmon/hwmon?).

### 47.7.4 Driver version

procfs: /proc/acpi/ibm/driver

sysfs driver attribute: version

The driver name and version. No commands can be written to this file.

### 47.7.5 Sysfs interface version

sysfs driver attribute: interface\_version

Version of the thinkpad-acpi sysfs interface, as an unsigned long (output in hex format: 0xAAAABBCC), where:

#### **AAAA**

- major revision

#### **BB**

- minor revision

#### **CC**

- bugfix revision

The sysfs interface version changelog for the driver can be found at the end of this document. Changes to the sysfs interface done by the kernel subsystems are not documented here, nor are they tracked by this attribute.

Changes to the thinkpad-acpi sysfs interface are only considered non-experimental when they are submitted to Linux mainline, at which point the changes in this interface are documented and interface\_version may be updated. If you are using any thinkpad-acpi features not yet sent to mainline for merging, you do so on your own risk: these features may disappear, or be implemented in a different and incompatible way by the time they are merged in Linux mainline.

Changes that are backwards-compatible by nature (e.g. the addition of attributes that do not change the way the other attributes work) do not always warrant an update of interface\_version. Therefore, one must expect that an attribute might not be there, and deal with it properly (an attribute not being there is a valid way to make it clear that a feature is not available in sysfs).

### 47.7.6 Hot keys

procfs: /proc/acpi/ibm/hotkey

sysfs device attribute: hotkey\_\*

In a ThinkPad, the ACPI HKEY handler is responsible for communicating some important events and also keyboard hot key presses to the operating system. Enabling the hotkey functionality of thinkpad-acpi signals the firmware that such a driver is present, and modifies how the ThinkPad firmware will behave in many situations.

The driver enables the HKEY ( “hot key” ) event reporting automatically when loaded, and disables it when it is removed.

The driver will report HKEY events in the following format:

```
ibm/hotkey HKEY 00000080 0000xxxx
```

Some of these events refer to hot key presses, but not all of them.

The driver will generate events over the input layer for hot keys and radio switches, and over the ACPI netlink layer for other events. The input layer support accepts the standard IOCTLS to remap the keycodes assigned to each hot key.

The hot key bit mask allows some control over which hot keys generate events. If a key is “masked” (bit set to 0 in the mask), the firmware will handle it. If it is “unmasked” , it signals the firmware that thinkpad-acpi would prefer to handle it, if the firmware would be so kind to allow it (and it often doesn’ t!).

Not all bits in the mask can be modified. Not all bits that can be modified do anything. Not all hot keys can be individually controlled by the mask. Some models do not support the mask at all. The behaviour of the mask is, therefore, highly dependent on the ThinkPad model.

The driver will filter out any unmasked hotkeys, so even if the firmware doesn’ t allow disabling an specific hotkey, the driver will not report events for unmasked hotkeys.

Note that unmasking some keys prevents their default behavior. For example, if Fn+F5 is unmasked, that key will no longer enable/disable Bluetooth by itself in firmware.

Note also that not all Fn key combinations are supported through ACPI depending on the ThinkPad model and firmware version. On those ThinkPads, it is still possible to support some extra hotkeys by polling the “CMOS NVRAM” at least 10 times per second. The driver attempts to enables this functionality automatically when required.

### procfs notes

The following commands can be written to the `/proc/acpi/ibm/hotkey` file:

```
echo 0xffffffff > /proc/acpi/ibm/hotkey -- enable all hot keys
echo 0 > /proc/acpi/ibm/hotkey -- disable all possible hot keys
... any other 8-hex-digit mask ...
echo reset > /proc/acpi/ibm/hotkey -- restore the recommended mask
```

The following commands have been deprecated and will cause the kernel to log a warning:

```
echo enable > /proc/acpi/ibm/hotkey -- does nothing
echo disable > /proc/acpi/ibm/hotkey -- returns an error
```

The procfs interface does not support NVRAM polling control. So as to maintain maximum bug-to-bug compatibility, it does not report any masks, nor does it allow one to manipulate the hot key mask when the firmware does not support masks at all, even if NVRAM polling is in use.

### sysfs notes

**hotkey\_bios\_enabled:** DEPRECATED, WILL BE REMOVED SOON.

Returns 0.

**hotkey\_bios\_mask:** DEPRECATED, DON' T USE, WILL BE REMOVED IN THE FUTURE.

Returns the hot keys mask when thinkpad-acpi was loaded. Upon module unload, the hot keys mask will be restored to this value. This is always 0x80c, because those are the hotkeys that were supported by ancient firmware without mask support.

**hotkey\_enable:** DEPRECATED, WILL BE REMOVED SOON.

0: returns -EPERM 1: does nothing

**hotkey\_mask:** bit mask to enable reporting (and depending on the firmware, ACPI event generation) for each hot key (see above). Returns the current status of the hot keys mask, and allows one to modify it.

**hotkey\_all\_mask:** bit mask that should enable event reporting for all supported hot keys, when echoed to hotkey\_mask above. Unless you know which events need to be handled passively (because the firmware will handle them anyway), do not use hotkey\_all\_mask. Use hotkey\_recommended\_mask, instead. You have been warned.

**hotkey\_recommended\_mask:** bit mask that should enable event reporting for all supported hot keys, except those which are always handled by the firmware anyway. Echo it to hotkey\_mask above, to use. This is the default mask used by the driver.

**hotkey\_source\_mask:** bit mask that selects which hot keys will the driver poll the NVRAM for. This is auto-detected by the driver based

on the capabilities reported by the ACPI firmware, but it can be overridden at runtime.

Hot keys whose bits are set in `hotkey_source_mask` are polled for in NVRAM, and reported as hotkey events if enabled in `hotkey_mask`. Only a few hot keys are available through CMOS NVRAM polling.

Warning: when in NVRAM mode, the volume up/down/mute keys are synthesized according to changes in the mixer, which uses a single volume up or volume down hotkey press to unmute, as per the ThinkPad volume mixer user interface. When in ACPI event mode, volume up/down/mute events are reported by the firmware and can behave differently (and that behaviour changes with firmware version - not just with firmware models - as well as OSI(Linux) state).

**hotkey\_poll\_freq:** frequency in Hz for hot key polling. It must be between 0 and 25 Hz. Polling is only carried out when strictly needed.

Setting `hotkey_poll_freq` to zero disables polling, and will cause hot key presses that require NVRAM polling to never be reported.

Setting `hotkey_poll_freq` too low may cause repeated pressings of the same hot key to be misreported as a single key press, or to not even be detected at all. The recommended polling frequency is 10Hz.

**hotkey\_radio\_sw:** If the ThinkPad has a hardware radio switch, this attribute will read 0 if the switch is in the “radios disabled” position, and 1 if the switch is in the “radios enabled” position.

This attribute has `poll()/select()` support.

**hotkey\_tablet\_mode:** If the ThinkPad has tablet capabilities, this attribute will read 0 if the ThinkPad is in normal mode, and 1 if the ThinkPad is in tablet mode.

This attribute has `poll()/select()` support.

**wakeup\_reason:** Set to 1 if the system is waking up because the user requested a bay ejection. Set to 2 if the system is waking up because the user requested the system to undock. Set to zero for normal wake-ups or wake-ups due to unknown reasons.

This attribute has `poll()/select()` support.

**wakeup\_hotunplug\_complete:** Set to 1 if the system was waken up because of an undock or bay ejection request, and that request was successfully completed. At this point, it might be useful to send the system back to sleep, at the user’s choice. Refer to HKEY events 0x4003 and 0x3003, below.

This attribute has `poll()/select()` support.

### input layer notes

A Hot key is mapped to a single input layer `EV_KEY` event, possibly followed by an `EV_MSC MSC_SCAN` event that shall contain that key' s scan code. An `EV_SYN` event will always be generated to mark the end of the event block.

Do not use the `EV_MSC MSC_SCAN` events to process keys. They are to be used as a helper to remap keys, only. They are particularly useful when remapping `KEY_UNKNOWN` keys.

The events are available in an input device, with the following id:

Bus	BUS_HOST
ven- dor	0x1014 (PCI_VENDOR_ID_IBM) or 0x17aa (PCI_VENDOR_ID_LENOVO)
prod- uct	0x5054 ( "TP" )
ver- sion	0x4101

The version will have its LSB incremented if the keymap changes in a backwards-compatible way. The MSB shall always be 0x41 for this input device. If the MSB is not 0x41, do not use the device as described in this section, as it is either something else (e.g. another input device exported by a thinkpad driver, such as HDAPS) or its functionality has been changed in a non-backwards compatible way.

Adding other event types for other functionalities shall be considered a backwards-compatible change for this input device.

Thinkpad-acpi Hot Key event map (version 0x4101):

ACPI event	Scan code	Key	Notes
0x1001	0x00	FN+F1	•
0x1002	0x01	FN+F2	IBM: battery (rare) Lenovo: Screen lock
0x1003	0x02	FN+F3	Many IBM models always report this hot key, even with hot keys disabled or with Fn+F3 masked off IBM: screen lock, often turns off the ThinkLight as side-effect Lenovo: battery
0x1004	0x03	FN+F4	Sleep button (ACPI sleep button semantics, i.e. sleep-to-RAM). It always generates some kind of event, either the hot key event or an ACPI sleep button event. The firmware may refuse to generate further FN+F4 key presses until a S3 or S4 ACPI sleep cycle is performed or some time passes.
0x1005	0x04	FN+F5	Radio. Enables/disables the internal Bluetooth hardware and W-WAN card if left in control of the firmware. Does not affect the WLAN card. Should be used to turn on/off all radios (Bluetooth+W-WAN+WLAN), really.
0x1006	0x05	FN+F6	•
<b>47.7. ThinkPad ACPI Extras Driver</b>			
0x1007	0x06	FN+F7	Video output cycle. Do you feel

The ThinkPad firmware does not allow one to differentiate when most hot keys are pressed or released (either that, or we don't know how to, yet). For these keys, the driver generates a set of events for a key press and immediately issues the same set of events for a key release. It is unknown by the driver if the ThinkPad firmware triggered these events on hot key press or release, but the firmware will do it for either one, not both.

If a key is mapped to `KEY_RESERVED`, it generates no input events at all. If a key is mapped to `KEY_UNKNOWN`, it generates an input event that includes an scan code. If a key is mapped to anything else, it will generate input device `EV_KEY` events.

In addition to the `EV_KEY` events, `thinkpad-acpi` may also issue `EV_SW` events for switches:

<code>SW_RFKILL_ALL</code>	T60 and later hardware rfkill rocker switch
<code>SW_TABLET_MODE</code>	Tablet ThinkPads HKEY events 0x5009 and 0x500A

### 47.7.7 Non hotkey ACPI HKEY event map

Events that are never propagated by the driver:

0x2304	System is waking up from suspend to undock
0x2305	System is waking up from suspend to eject bay
0x2404	System is waking up from hibernation to undock
0x2405	System is waking up from hibernation to eject bay
0x5001	Lid closed
0x5002	Lid opened
0x5009	Tablet swivel: switched to tablet mode
0x500A	Tablet swivel: switched to normal mode
0x5010	Brightness level changed/control event
0x6000	KEYBOARD: Numlock key pressed
0x6005	KEYBOARD: Fn key pressed (TO BE VERIFIED)
0x7000	Radio Switch may have changed state

Events that are propagated by the driver to userspace:

0x2313	ALARM: System is waking up from suspend because the battery is nearly empty
0x2413	ALARM: System is waking up from hibernation because the battery is nearly empty
0x3003	Bay ejection (see 0x2x05) complete, can sleep again
0x3006	Bay hotplug request (hint to power up SATA link when the optical drive tray is ejected)
0x4003	Undocked (see 0x2x04), can sleep again
0x4010	Docked into hotplug port replicator (non-ACPI dock)
0x4011	Undocked from hotplug port replicator (non-ACPI dock)
0x500B	Tablet pen inserted into its storage bay
0x500C	Tablet pen removed from its storage bay
0x6011	ALARM: battery is too hot
0x6012	ALARM: battery is extremely hot
0x6021	ALARM: a sensor is too hot
0x6022	ALARM: a sensor is extremely hot
0x6030	System thermal table changed
0x6032	Thermal Control command set completion (DYTC, Windows)
0x6040	Nvidia Optimus/AC adapter related (TO BE VERIFIED)
0x60C0	X1 Yoga 2016, Tablet mode status changed
0x60F0	Thermal Transformation changed (GMTS, Windows)

Battery nearly empty alarms are a last resort attempt to get the operating system to hibernate or shutdown cleanly (0x2313), or shutdown cleanly (0x2413) before power is lost. They must be acted upon, as the wake up caused by the firmware will have negated most safety nets...

When any of the “too hot” alarms happen, according to Lenovo the user should suspend or hibernate the laptop (and in the case of battery alarms, unplug the AC adapter) to let it cool down. These alarms do signal that something is wrong, they should never happen on normal operating conditions.

The “extremely hot” alarms are emergencies. According to Lenovo, the operating system is to force either an immediate suspend or hibernate cycle, or a system shutdown. Obviously, something is very wrong if this happens.

### Brightness hotkey notes

Don’ t mess with the brightness hotkeys in a Thinkpad. If you want notifications for OSD, use the sysfs backlight class event support.

The driver will issue KEY\_BRIGHTNESS\_UP and KEY\_BRIGHTNESS\_DOWN events automatically for the cases where userspace has to do something to implement brightness changes. When you override these events, you will either fail to handle properly the ThinkPads that require explicit action to change backlight brightness, or the ThinkPads that require that no action be taken to work properly.

### 47.7.8 Bluetooth

procfs: /proc/acpi/ibm/bluetooth

sysfs device attribute: bluetooth\_enable (deprecated)

sysfs rfkill class: switch “tpacpi\_bluetooth\_sw”

This feature shows the presence and current state of a ThinkPad Bluetooth device in the internal ThinkPad CDC slot.

If the ThinkPad supports it, the Bluetooth state is stored in NVRAM, so it is kept across reboots and power-off.

#### Procfs notes

If Bluetooth is installed, the following commands can be used:

```
echo enable > /proc/acpi/ibm/bluetooth
echo disable > /proc/acpi/ibm/bluetooth
```

#### Sysfs notes

If the Bluetooth CDC card is installed, it can be enabled / disabled through the “bluetooth\_enable” thinkpad-acpi device attribute, and its current status can also be queried.

enable:

- 0: disables Bluetooth / Bluetooth is disabled
- 1: enables Bluetooth / Bluetooth is enabled.

Note: this interface has been superseded by the generic rfkill class. It has been deprecated, and it will be removed in year 2010.

rfkill controller switch “tpacpi\_bluetooth\_sw” : refer to Documentation/driver-api/rfkill.rst for details.

### 47.7.9 Video output control - /proc/acpi/ibm/video

This feature allows control over the devices used for video output - LCD, CRT or DVI (if available). The following commands are available:

```
echo lcd_enable > /proc/acpi/ibm/video
echo lcd_disable > /proc/acpi/ibm/video
echo crt_enable > /proc/acpi/ibm/video
echo crt_disable > /proc/acpi/ibm/video
echo dvi_enable > /proc/acpi/ibm/video
echo dvi_disable > /proc/acpi/ibm/video
echo auto_enable > /proc/acpi/ibm/video
echo auto_disable > /proc/acpi/ibm/video
echo expand_toggle > /proc/acpi/ibm/video
echo video_switch > /proc/acpi/ibm/video
```

**NOTE:** Access to this feature is restricted to processes owning the `CAP_SYS_ADMIN` capability for safety reasons, as it can interact badly enough with some versions of X.org to crash it.

Each video output device can be enabled or disabled individually. Reading `/proc/acpi/ibm/video` shows the status of each device.

Automatic video switching can be enabled or disabled. When automatic video switching is enabled, certain events (e.g. opening the lid, docking or undocking) cause the video output device to change automatically. While this can be useful, it also causes flickering and, on the X40, video corruption. By disabling automatic switching, the flickering or video corruption can be avoided.

The `video_switch` command cycles through the available video outputs (it simulates the behavior of Fn-F7).

Video expansion can be toggled through this feature. This controls whether the display is expanded to fill the entire LCD screen when a mode with less than full resolution is used. Note that the current video expansion status cannot be determined through this feature.

Note that on many models (particularly those using Radeon graphics chips) the X driver configures the video card in a way which prevents Fn-F7 from working. This also disables the video output switching features of this driver, as it uses the same ACPI methods as Fn-F7. Video switching on the console should still work.

UPDATE: refer to [https://bugs.freedesktop.org/show\\_bug.cgi?id=2000](https://bugs.freedesktop.org/show_bug.cgi?id=2000)

### 47.7.10 ThinkLight control

procfs: `/proc/acpi/ibm/light`

sysfs attributes: as per LED class, for the “`tpacpi::thinklight`” LED

#### procfs notes

The ThinkLight status can be read and set through the procfs interface. A few models which do not make the status available will show the ThinkLight status as “unknown” . The available commands are:

```
echo on > /proc/acpi/ibm/light
echo off > /proc/acpi/ibm/light
```

#### sysfs notes

The ThinkLight sysfs interface is documented by the LED class documentation, in Documentation/leds/leds-class.rst. The ThinkLight LED name is “`tpacpi::thinklight`” .

Due to limitations in the sysfs LED class, if the status of the ThinkLight cannot be read or if it is unknown, thinkpad-acpi will report it as “off” . It is impossible to know if the status returned through sysfs is valid.

### 47.7.11 CMOS/UCMS control

procfs: /proc/acpi/ibm/cmos

sysfs device attribute: cmos\_command

This feature is mostly used internally by the ACPI firmware to keep the legacy CMOS NVRAM bits in sync with the current machine state, and to record this state so that the ThinkPad will retain such settings across reboots.

Some of these commands actually perform actions in some ThinkPad models, but this is expected to disappear more and more in newer models. As an example, in a T43 and in a X40, commands 12 and 13 still control the ThinkLight state for real, but commands 0 to 2 don't control the mixer anymore (they have been phased out) and just update the NVRAM.

The range of valid cmos command numbers is 0 to 21, but not all have an effect and the behavior varies from model to model. Here is the behavior on the X40 (tpb is the ThinkPad Buttons utility):

- 0 - Related to "Volume down" key press
- 1 - Related to "Volume up" key press
- 2 - Related to "Mute on" key press
- 3 - Related to "Access IBM" key press
- 4 - Related to "LCD brightness up" key press
- 5 - Related to "LCD brightness down" key press
- 11 - Related to "toggle screen expansion" key press/function
- 12 - Related to "ThinkLight on"
- 13 - Related to "ThinkLight off"
- 14 - Related to "ThinkLight" key press (toggle ThinkLight)

The cmos command interface is prone to firmware split-brain problems, as in newer ThinkPads it is just a compatibility layer. Do not use it, it is exported just as a debug tool.

### 47.7.12 LED control

procfs: /proc/acpi/ibm/led sysfs attributes: as per LED class, see below for names

Some of the LED indicators can be controlled through this feature. On some older ThinkPad models, it is possible to query the status of the LED indicators as well. Newer ThinkPads cannot query the real status of the LED indicators.

Because misuse of the LEDs could induce an unaware user to perform dangerous actions (like undocking or ejecting a bay device while the buses are still active), or mask an important alarm (such as a nearly empty battery, or a broken battery), access to most LEDs is restricted.

Unrestricted access to all LEDs requires that thinkpad-acpi be compiled with the CONFIG\_THINKPAD\_ACPI\_UNSAFE\_LEDS option enabled. Distributions must

never enable this option. Individual users that are aware of the consequences are welcome to enabling it.

Audio mute and microphone mute LEDs are supported, but currently not visible to userspace. They are used by the `snd-hda-intel` audio driver.

### procfs notes

The available commands are:

```
echo '<LED number> on' >/proc/acpi/ibm/led
echo '<LED number> off' >/proc/acpi/ibm/led
echo '<LED number> blink' >/proc/acpi/ibm/led
```

The `<LED number>` range is 0 to 15. The set of LEDs that can be controlled varies from model to model. Here is the common ThinkPad mapping:

- 0 - power
- 1 - battery (orange)
- 2 - battery (green)
- 3 - UltraBase/dock
- 4 - UltraBay
- 5 - UltraBase battery slot
- 6 - (unknown)
- 7 - standby
- 8 - dock status 1
- 9 - dock status 2
- 10, 11 - (unknown)
- 12 - thinkvantage
- 13, 14, 15 - (unknown)

All of the above can be turned on and off and can be made to blink.

### sysfs notes

The ThinkPad LED sysfs interface is described in detail by the LED class documentation, in `Documentation/leds/leds-class.rst`.

The LEDs are named (in LED ID order, from 0 to 12): “`tpacpi::power`”, “`tpacpi:orange:batt`”, “`tpacpi:green:batt`”, “`tpacpi::dock_active`”, “`tpacpi::bay_active`”, “`tpacpi::dock_batt`”, “`tpacpi::unknown_led`”, “`tpacpi::standby`”, “`tpacpi::dock_status1`”, “`tpacpi::dock_status2`”, “`tpacpi::unknown_led2`”, “`tpacpi::unknown_led3`”, “`tpacpi::thinkvantage`” .

Due to limitations in the sysfs LED class, if the status of the LED indicators cannot be read due to an error, `thinkpad-acpi` will report it as a brightness of zero (same as LED off).

If the thinkpad firmware doesn't support reading the current status, trying to read the current LED brightness will just return whatever brightness was last written to that attribute.

These LEDs can blink using hardware acceleration. To request that a ThinkPad indicator LED should blink in hardware accelerated mode, use the "timer" trigger, and leave the `delay_on` and `delay_off` parameters set to zero (to request hardware acceleration autodetection).

LEDs that are known not to exist in a given ThinkPad model are not made available through the sysfs interface. If you have a dock and you notice there are LEDs listed for your ThinkPad that do not exist (and are not in the dock), or if you notice that there are missing LEDs, a report to [ibm-acpi-devel@lists.sourceforge.net](mailto:ibm-acpi-devel@lists.sourceforge.net) is appreciated.

### 47.7.13 ACPI sounds - `/proc/acpi/ibm/beep`

The BEEP method is used internally by the ACPI firmware to provide audible alerts in various situations. This feature allows the same sounds to be triggered manually.

The commands are non-negative integer numbers:

```
echo <number> >/proc/acpi/ibm/beep
```

The valid `<number>` range is 0 to 17. Not all numbers trigger sounds and the sounds vary from model to model. Here is the behavior on the X40:

- 0 - stop a sound in progress (but use 17 to stop 16)
- 2 - two beeps, pause, third beep ( "low battery" )
- 3 - single beep
- 4 - high, followed by low-pitched beep ( "unable" )
- 5 - single beep
- 6 - very high, followed by high-pitched beep ( "AC/DC" )
- 7 - high-pitched beep
- 9 - three short beeps
- 10 - very long beep
- 12 - low-pitched beep
- 15 - three high-pitched beeps repeating constantly, stop with 0
- 16 - one medium-pitched beep repeating constantly, stop with 17
- 17 - stop 16

### 47.7.14 Temperature sensors

procfs: /proc/acpi/ibm/thermal

sysfs device attributes: (hwmon "thinkpad" ) temp\*\_input

Most ThinkPads include six or more separate temperature sensors but only expose the CPU temperature through the standard ACPI methods. This feature shows readings from up to eight different sensors on older ThinkPads, and up to sixteen different sensors on newer ThinkPads.

For example, on the X40, a typical output may be:

**temperatures:** 42 42 45 41 36 -128 33 -128

On the T43/p, a typical output may be:

**temperatures:** 48 48 36 52 38 -128 31 -128 48 52 48 -128 -128 -128 -128 -128

The mapping of thermal sensors to physical locations varies depending on system-board model (and thus, on ThinkPad model).

[http://thinkwiki.org/wiki/Thermal\\_Sensors](http://thinkwiki.org/wiki/Thermal_Sensors) is a public wiki page that tries to track down these locations for various models.

Most (newer?) models seem to follow this pattern:

- 1: CPU
- 2: (depends on model)
- 3: (depends on model)
- 4: GPU
- 5: Main battery: main sensor
- 6: Bay battery: main sensor
- 7: Main battery: secondary sensor
- 8: Bay battery: secondary sensor
- 9-15: (depends on model)

For the R51 (source: Thomas Gruber):

- 2: Mini-PCI
- 3: Internal HDD

For the T43, T43/p (source: Shmidoax/Thinkwiki.org) [http://thinkwiki.org/wiki/Thermal\\_Sensors#ThinkPad\\_T43.2C\\_T43p](http://thinkwiki.org/wiki/Thermal_Sensors#ThinkPad_T43.2C_T43p)

- 2: System board, left side (near PCMCIA slot), reported as HDAPS temp
- 3: PCMCIA slot
- 9: MCH (northbridge) to DRAM Bus
- **10: Clock-generator, mini-pci card and ICH (southbridge), under Mini-PCI card, under touchpad**
- 11: Power regulator, underside of system board, below F2 key

The A31 has a very atypical layout for the thermal sensors (source: Milos Popovic, [http://thinkwiki.org/wiki/Thermal\\_Sensors#ThinkPad\\_A31](http://thinkwiki.org/wiki/Thermal_Sensors#ThinkPad_A31))

- 1: CPU
- 2: Main Battery: main sensor
- 3: Power Converter
- 4: Bay Battery: main sensor
- 5: MCH (northbridge)
- 6: PCMCIA/ambient
- 7: Main Battery: secondary sensor
- 8: Bay Battery: secondary sensor

### Procfs notes

Readings from sensors that are not available return -128. No commands can be written to this file.

### Sysfs notes

Sensors that are not available return the ENXIO error. This status may change at runtime, as there are hotplug thermal sensors, like those inside the batteries and docks.

thinkpad-acpi thermal sensors are reported through the hwmon subsystem, and follow all of the hwmon guidelines at Documentation/hwmon.

### 47.7.15 EXPERIMENTAL: Embedded controller register dump

This feature is not included in the thinkpad driver anymore. Instead the EC can be accessed through `/sys/kernel/debug/ec` with a userspace tool which can be found here: <ftp://ftp.suse.com/pub/people/trenn/sources/ec>

Use it to determine the register holding the fan speed on some models. To do that, do the following:

- make sure the battery is fully charged
- make sure the fan is running
- use above mentioned tool to read out the EC

Often fan and temperature values vary between readings. Since temperatures don't change vary fast, you can take several quick dumps to eliminate them.

You can use a similar method to figure out the meaning of other embedded controller registers - e.g. make sure nothing else changes except the charging or discharging battery to determine which registers contain the current battery capacity, etc. If you experiment with this, do send me your results (including some complete dumps with a description of the conditions when they were taken.)

### 47.7.16 LCD brightness control

procfs: /proc/acpi/ibm/brightness

sysfs backlight device “thinkpad\_screen”

This feature allows software control of the LCD brightness on ThinkPad models which don't have a hardware brightness slider.

It has some limitations: the LCD backlight cannot be actually turned on or off by this interface, it just controls the backlight brightness level.

On IBM (and some of the earlier Lenovo) ThinkPads, the backlight control has eight brightness levels, ranging from 0 to 7. Some of the levels may not be distinct. Later Lenovo models that implement the ACPI display backlight brightness control methods have 16 levels, ranging from 0 to 15.

For IBM ThinkPads, there are two interfaces to the firmware for direct brightness control, EC and UCMS (or CMOS). To select which one should be used, use the `brightness_mode` module parameter: `brightness_mode=1` selects EC mode, `brightness_mode=2` selects UCMS mode, `brightness_mode=3` selects EC mode with NVRAM backing (so that brightness changes are remembered across shutdown/reboot).

The driver tries to select which interface to use from a table of defaults for each ThinkPad model. If it makes a wrong choice, please report this as a bug, so that we can fix it.

Lenovo ThinkPads only support `brightness_mode=2` (UCMS).

When display backlight brightness controls are available through the standard ACPI interface, it is best to use it instead of this direct ThinkPad-specific interface. The driver will disable its native backlight brightness control interface if it detects that the standard ACPI interface is available in the ThinkPad.

If you want to use the `thinkpad-acpi` backlight brightness control instead of the generic ACPI video backlight brightness control for some reason, you should use the `acpi_backlight=vendor` kernel parameter.

The `brightness_enable` module parameter can be used to control whether the LCD brightness control feature will be enabled when available. `brightness_enable=0` forces it to be disabled. `brightness_enable=1` forces it to be enabled when available, even if the standard ACPI interface is also available.

#### Procfs notes

The available commands are:

```
echo up >/proc/acpi/ibm/brightness
echo down >/proc/acpi/ibm/brightness
echo 'level <level>' >/proc/acpi/ibm/brightness
```

### Sysfs notes

The interface is implemented through the backlight sysfs class, which is poorly documented at this time.

Locate the `thinkpad_screen` device under `/sys/class/backlight`, and inside it there will be the following attributes:

**max\_brightness:** Reads the maximum brightness the hardware can be set to. The minimum is always zero.

**actual\_brightness:** Reads what brightness the screen is set to at this instant.

**brightness:** Writes request the driver to change brightness to the given value. Reads will tell you what brightness the driver is trying to set the display to when “power” is set to zero and the display has not been dimmed by a kernel power management event.

**power:** power management mode, where 0 is “display on”, and 1 to 3 will dim the display backlight to brightness level 0 because `thinkpad-acpi` cannot really turn the backlight off. Kernel power management events can temporarily increase the current power management level, i.e. they can dim the display.

#### WARNING:

Whatever you do, do NOT ever call `thinkpad-acpi` backlight-level change interface and the ACPI-based backlight level change interface (available on newer BIOSes, and driven by the Linux ACPI video driver) at the same time. The two will interact in bad ways, do funny things, and maybe reduce the life of the backlight lamps by needlessly kicking its level up and down at every change.

### 47.7.17 Volume control (Console Audio control)

procfs: `/proc/acpi/ibm/volume`

ALSA: “ThinkPad Console Audio Control” , default ID: “ThinkPadEC”

NOTE: by default, the volume control interface operates in read-only mode, as it is supposed to be used for on-screen-display purposes. The read/write mode can be enabled through the use of the “`volume_control=1`” module parameter.

NOTE: distros are urged to not enable `volume_control` by default, this should be done by the local admin only. The ThinkPad UI is for the console audio control to be done through the volume keys only, and for the desktop environment to just provide on-screen-display feedback. Software volume control should be done only in the main AC97/HDA mixer.

## About the ThinkPad Console Audio control

ThinkPads have a built-in amplifier and muting circuit that drives the console headphone and speakers. This circuit is after the main AC97 or HDA mixer in the audio path, and under exclusive control of the firmware.

ThinkPads have three special hotkeys to interact with the console audio control: volume up, volume down and mute.

It is worth noting that the normal way the mute function works (on ThinkPads that do not have a “mute LED” ) is:

1. Press mute to mute. It will always mute, you can press it as many times as you want, and the sound will remain mute.
2. Press either volume key to unmute the ThinkPad (it will *not* change the volume, it will just unmute).

This is a very superior design when compared to the cheap software-only mute-toggle solution found on normal consumer laptops: you can be absolutely sure the ThinkPad will not make noise if you press the mute button, no matter the previous state.

The IBM ThinkPads, and the earlier Lenovo ThinkPads have variable-gain amplifiers driving the speakers and headphone output, and the firmware also handles volume control for the headphone and speakers on these ThinkPads without any help from the operating system (this volume control stage exists after the main AC97 or HDA mixer in the audio path).

The newer Lenovo models only have firmware mute control, and depend on the main HDA mixer to do volume control (which is done by the operating system). In this case, the volume keys are filtered out for unmute key press (there are some firmware bugs in this area) and delivered as normal key presses to the operating system (thinkpad-acpi is not involved).

## The ThinkPad-ACPI volume control

The preferred way to interact with the Console Audio control is the ALSA interface.

The legacy procfs interface allows one to read the current state, and if volume control is enabled, accepts the following commands:

```
echo up >/proc/acpi/ibm/volume
echo down >/proc/acpi/ibm/volume
echo mute >/proc/acpi/ibm/volume
echo unmute >/proc/acpi/ibm/volume
echo 'level <level>' >/proc/acpi/ibm/volume
```

The <level> number range is 0 to 14 although not all of them may be distinct. To unmute the volume after the mute command, use either the up or down command (the level command will not unmute the volume), or the unmute command.

You can use the `volume_capabilities` parameter to tell the driver whether your thinkpad has volume control or mute-only control: `volume_capabilities=1` for mixers with mute and volume control, `volume_capabilities=2` for mixers with only mute control.

If the driver misdetects the capabilities for your ThinkPad model, please report this to [ibm-acpi-devel@lists.sourceforge.net](mailto:ibm-acpi-devel@lists.sourceforge.net), so that we can update the driver.

There are two strategies for volume control. To select which one should be used, use the `volume_mode` module parameter: `volume_mode=1` selects EC mode, and `volume_mode=3` selects EC mode with NVRAM backing (so that volume/mute changes are remembered across shutdown/reboot).

The driver will operate in `volume_mode=3` by default. If that does not work well on your ThinkPad model, please report this to [ibm-acpi-devel@lists.sourceforge.net](mailto:ibm-acpi-devel@lists.sourceforge.net).

The driver supports the standard ALSA module parameters. If the ALSA mixer is disabled, the driver will disable all volume functionality.

### 47.7.18 Fan control and monitoring: fan speed, fan enable/disable

procfs: `/proc/acpi/ibm/fan`

sysfs device attributes: (hwmon “thinkpad” ) `fan1_input`, `pwm1`, `pwm1_enable`, `fan2_input`

sysfs hwmon driver attributes: `fan_watchdog`

**NOTE NOTE NOTE:** fan control operations are disabled by default for safety reasons. To enable them, the module parameter “`fan_control=1`” must be given to `thinkpad-acpi`.

This feature attempts to show the current fan speed, control mode and other fan data that might be available. The speed is read directly from the hardware registers of the embedded controller. This is known to work on later R, T, X and Z series ThinkPads but may show a bogus value on other models.

Some Lenovo ThinkPads support a secondary fan. This fan cannot be controlled separately, it shares the main fan control.

#### Fan levels

Most ThinkPad fans work in “levels” at the firmware interface. Level 0 stops the fan. The higher the level, the higher the fan speed, although adjacent levels often map to the same fan speed. 7 is the highest level, where the fan reaches the maximum recommended speed.

Level “auto” means the EC changes the fan level according to some internal algorithm, usually based on readings from the thermal sensors.

There is also a “full-speed” level, also known as “disengaged” level. In this level, the EC disables the speed-locked closed-loop fan control, and drives the fan as fast as it can go, which might exceed hardware limits, so use this level with caution.

The fan usually ramps up or down slowly from one speed to another, and it is normal for the EC to take several seconds to react to fan commands. The full-speed level may take up to two minutes to ramp up to maximum speed, and in some ThinkPads, the tachometer readings go stale while the EC is transitioning to the full-speed level.

WARNING WARNING WARNING: do not leave the fan disabled unless you are monitoring all of the temperature sensor readings and you are ready to enable it if necessary to avoid overheating.

An enabled fan in level “auto” may stop spinning if the EC decides the ThinkPad is cool enough and doesn’t need the extra airflow. This is normal, and the EC will spin the fan up if the various thermal readings rise too much.

On the X40, this seems to depend on the CPU and HDD temperatures. Specifically, the fan is turned on when either the CPU temperature climbs to 56 degrees or the HDD temperature climbs to 46 degrees. The fan is turned off when the CPU temperature drops to 49 degrees and the HDD temperature drops to 41 degrees. These thresholds cannot currently be controlled.

The ThinkPad’s ACPI DSDT code will reprogram the fan on its own when certain conditions are met. It will override any fan programming done through thinkpad-acpi.

The thinkpad-acpi kernel driver can be programmed to revert the fan level to a safe setting if userspace does not issue one of the procfs fan commands: “enable”, “disable”, “level” or “watchdog”, or if there are no writes to pwm1\_enable (or to pwm1 if and only if pwm1\_enable is set to 1, manual mode) within a configurable amount of time of up to 120 seconds. This functionality is called fan safety watchdog.

Note that the watchdog timer stops after it enables the fan. It will be rearmed again automatically (using the same interval) when one of the above mentioned fan commands is received. The fan watchdog is, therefore, not suitable to protect against fan mode changes made through means other than the “enable”, “disable”, and “level” procfs fan commands, or the hwmon fan control sysfs interface.

## Procfs notes

The fan may be enabled or disabled with the following commands:

```
echo enable >/proc/acpi/ibm/fan
echo disable >/proc/acpi/ibm/fan
```

Placing a fan on level 0 is the same as disabling it. Enabling a fan will try to place it in a safe level if it is too slow or disabled.

The fan level can be controlled with the command:

```
echo 'level <level>' > /proc/acpi/ibm/fan
```

Where <level> is an integer from 0 to 7, or one of the words “auto” or “full-speed” (without the quotes). Not all ThinkPads support the “auto” and “full-speed” levels. The driver accepts “disengaged” as an alias for “full-speed”, and reports it as “disengaged” for backwards compatibility.

On the X31 and X40 (and ONLY on those models), the fan speed can be controlled to a certain degree. Once the fan is running, it can be forced to run faster or slower with the following command:

```
echo 'speed <speed>' > /proc/acpi/ibm/fan
```

The sustainable range of fan speeds on the X40 appears to be from about 3700 to about 7350. Values outside this range either do not have any effect or the fan speed eventually settles somewhere in that range. The fan cannot be stopped or started with this command. This functionality is incomplete, and not available through the sysfs interface.

To program the safety watchdog, use the “watchdog” command:

```
echo 'watchdog <interval in seconds>' > /proc/acpi/ibm/fan
```

If you want to disable the watchdog, use 0 as the interval.

### Sysfs notes

The sysfs interface follows the hwmon subsystem guidelines for the most part, and the exception is the fan safety watchdog.

Writes to any of the sysfs attributes may return the EINVAL error if that operation is not supported in a given ThinkPad or if the parameter is out-of-bounds, and EPERM if it is forbidden. They may also return EINTR (interrupted system call), and EIO (I/O error while trying to talk to the firmware).

Features not yet implemented by the driver return ENOSYS.

#### hwmon device attribute pwm1\_enable:

- 0: PWM offline (fan is set to full-speed mode)
- 1: Manual PWM control (use pwm1 to set fan level)
- 2: Hardware PWM control (EC “auto” mode)
- 3: reserved (Software PWM control, not implemented yet)

Modes 0 and 2 are not supported by all ThinkPads, and the driver is not always able to detect this. If it does know a mode is unsupported, it will return -EINVAL.

**hwmon device attribute pwm1:** Fan level, scaled from the firmware values of 0-7 to the hwmon scale of 0-255. 0 means fan stopped, 255 means highest normal speed (level 7).

This attribute only commands the fan if pmw1\_enable is set to 1 (manual PWM control).

**hwmon device attribute fan1\_input:** Fan tachometer reading, in RPM. May go stale on certain ThinkPads while the EC transitions the PWM to offline mode, which can take up to two minutes. May return rubbish on older ThinkPads.

**hwmon device attribute fan2\_input:** Fan tachometer reading, in RPM, for the secondary fan. Available only on some ThinkPads. If the secondary fan is not installed, will always read 0.

**hwmon driver attribute fan\_watchdog:** Fan safety watchdog timer interval, in seconds. Minimum is 1 second, maximum is 120 seconds. 0 disables the watchdog.

To stop the fan: set `pwm1` to zero, and `pwm1_enable` to 1.

To start the fan in a safe mode: set `pwm1_enable` to 2. If that fails with `EINVAL`, try to set `pwm1_enable` to 1 and `pwm1` to at least 128 (255 would be the safest choice, though).

### 47.7.19 WAN

procfs: `/proc/acpi/ibm/wan`

sysfs device attribute: `wwan_enable` (deprecated)

sysfs rfkill class: `switch "tpacpi_wwan_sw"`

This feature shows the presence and current state of the built-in Wireless WAN device.

If the ThinkPad supports it, the WWAN state is stored in NVRAM, so it is kept across reboots and power-off.

It was tested on a Lenovo ThinkPad X60. It should probably work on other ThinkPad models which come with this module installed.

#### Procfs notes

If the W-WAN card is installed, the following commands can be used:

```
echo enable > /proc/acpi/ibm/wan
echo disable > /proc/acpi/ibm/wan
```

#### Sysfs notes

If the W-WAN card is installed, it can be enabled / disabled through the `"wwan_enable"` thinkpad-acpi device attribute, and its current status can also be queried.

##### **enable:**

- 0: disables WWAN card / WWAN card is disabled
- 1: enables WWAN card / WWAN card is enabled.

Note: this interface has been superseded by the generic rfkill class. It has been deprecated, and it will be removed in year 2010.

rfkill controller switch `"tpacpi_wwan_sw"` : refer to [Documentation/driver-api/rfkill.rst](#) for details.

### 47.7.20 LCD Shadow control

procfs: /proc/acpi/ibm/lcdshadow

Some newer T480s and T490s ThinkPads provide a feature called PrivacyGuard. By turning this feature on, the usable vertical and horizontal viewing angles of the LCD can be limited (as if some privacy screen was applied manually in front of the display).

#### procfs notes

The available commands are:

```
echo '0' >/proc/acpi/ibm/lcdshadow
echo '1' >/proc/acpi/ibm/lcdshadow
```

The first command ensures the best viewing angle and the latter one turns on the feature, restricting the viewing angles.

### 47.7.21 EXPERIMENTAL: UWB

This feature is considered EXPERIMENTAL because it has not been extensively tested and validated in various ThinkPad models yet. The feature may not work as expected. USE WITH CAUTION! To use this feature, you need to supply the `experimental=1` parameter when loading the module.

sysfs rkill class: switch “tpacpi\_uwb\_sw”

This feature exports an rkill controller for the UWB device, if one is present and enabled in the BIOS.

#### Sysfs notes

rkill controller switch “tpacpi\_uwb\_sw”: refer to Documentation/driver-api/rkill.rst for details.

### 47.7.22 Adaptive keyboard

sysfs device attribute: `adaptive_kbd_mode`

This sysfs attribute controls the keyboard “face” that will be shown on the Lenovo X1 Carbon 2nd gen (2014)’ s adaptive keyboard. The value can be read and set.

- 1 = Home mode
- 2 = Web-browser mode
- 3 = Web-conference mode
- 4 = Function mode
- 5 = Layflat mode

For more details about which buttons will appear depending on the mode, please review the laptop's user guide: [http://www.lenovo.com/shop/americas/content/user\\_guides/x1carbon\\_2\\_ug\\_en.pdf](http://www.lenovo.com/shop/americas/content/user_guides/x1carbon_2_ug_en.pdf)

### 47.7.23 Multiple Commands, Module Parameters

Multiple commands can be written to the proc files in one shot by separating them with commas, for example:

```
echo enable,0xffff > /proc/acpi/ibm/hotkey
echo lcd_disable,crt_enable > /proc/acpi/ibm/video
```

Commands can also be specified when loading the thinkpad-acpi module, for example:

```
modprobe thinkpad_acpi hotkey=enable,0xffff video=auto_disable
```

### 47.7.24 Enabling debugging output

The module takes a debug parameter which can be used to selectively enable various classes of debugging output, for example:

```
modprobe thinkpad_acpi debug=0xffff
```

will enable all debugging output classes. It takes a bitmask, so to enable more than one output class, just add their values.

Debug bit-mask	Description
0x8000	Disclose PID of userspace programs accessing some functions of the driver
0x0001	Initialization and probing
0x0002	Removal
0x0004	RF Transmitter control (RFKILL) (bluetooth, WWAN, UWB...)
0x0008	HKEY event interface, hotkeys
0x0010	Fan control
0x0020	Backlight brightness
0x0040	Audio mixer/volume control

There is also a kernel build option to enable more debugging information, which may be necessary to debug driver problems.

The level of debugging information output by the driver can be changed at runtime through sysfs, using the driver attribute `debug_level`. The attribute takes the same bitmask as the debug module parameter above.

### 47.7.25 Force loading of module

If thinkpad-acpi refuses to detect your ThinkPad, you can try to specify the module parameter `force_load=1`. Regardless of whether this works or not, please contact [ibm-acpi-devel@lists.sourceforge.net](mailto:ibm-acpi-devel@lists.sourceforge.net) with a report.

### Sysfs interface changelog

0x00011000	Initial sysfs support, as a single platform driver and device.
0x00012000	Key support for 32 hot keys, and radio slider switch support.
0x01000000	Keys are now handled by default over the input layer, the radio switch generates input event EV_RADIO, and the driver enables hot key handling by default in the firmware.
0x02000000	Fix: added a separate hwmon platform device and driver, which must be located by name (thinkpad) and the hwmon class for libsensors4 (lm-sensors 3) compatibility. Moved all hwmon attributes to this new platform device.
0x02010000	Marker for thinkpad-acpi with hot key NVRAM polling support. If you must, use it to know you should not start a userspace NVRAM poller (allows to detect when NVRAM is compiled out by the user because it is unneeded/undesired in the first place).
0x02011000	Marker for thinkpad-acpi with hot key NVRAM polling and proper hotkey_mask semantics (version 8 of the NVRAM polling patch). Some development snapshots of 0.18 had an earlier version that did strange things to hotkey_mask.
0x02020000	poll()/select() support to the following attributes: hotkey_radio_sw, wakeup_hotunplug_complete, wakeup_reason
0x02030000	Key enable/disable support removed, attributes hotkey_bios_enabled and hotkey_enable deprecated and marked for removal.
0x02040000	Marker for 16 LEDs support. Also, LEDs that are known to not exist in a given model are not registered with the LED sysfs class anymore.
0x02050000	Updated hotkey driver, hotkey_mask is always available and it is always able to disable hot keys. Very old thinkpads are properly supported. hotkey_bios_mask is deprecated and marked for removal.
0x02060000	Marker for backlight change event support.
0x02070000	Support for mute-only mixers. Volume control in read-only mode by default. Marker for ALSA mixer support.
0x03000000	Thermal and fan sysfs attributes were moved to the hwmon device instead of being attached to the backing platform device.

## 47.8 Toshiba HDD Active Protection Sensor

Kernel driver: toshiba\_haps

Author: Azael Avalos <coproscefalo@gmail.com>

### 47.8.1 1. Description

This driver provides support for the accelerometer found in various Toshiba laptops, being called “Toshiba HDD Protection - Shock Sensor” officially, and detects laptops automatically with this device. On Windows, Toshiba provided software monitors this device and provides automatic HDD protection (head unload) on sudden moves or harsh vibrations, however, this driver only provides a notification via a sysfs file to let userspace tools or daemons act accordingly, as well as providing a sysfs file to set the desired protection level or sensor sensibility.

### 47.8.2 2. Interface

This device comes with 3 methods:

_STA	Checks existence of the device, returning Zero if the device does not exists or is not supported.
PTLV	Sets the desired protection level.
RSSS	Shuts down the HDD protection interface for a few seconds, then restores normal operation.

**Note:** The presence of Solid State Drives (SSD) can make this driver to fail loading, given the fact that such drives have no movable parts, and thus, not requiring any “protection” as well as failing during the evaluation of the \_STA method found under this device.

### 47.8.3 3. Accelerometer axes

This device does not report any axes, however, to query the sensor position a couple HCI (Hardware Configuration Interface) calls (0x6D and 0xA6) are provided to query such information, handled by the kernel module toshiba\_acpi since kernel version 3.15.

### 47.8.4 4. Supported devices

This driver binds itself to the ACPI device TOS620A, and any Toshiba laptop with this device is supported, given the fact that they have the presence of conventional HDD and not only SSD, or a combination of both HDD and SSD.

### 47.8.5 5. Usage

The sysfs files under /sys/devices/LNXSYSTM:00/LNXSYBUS:00/TOS620A:00/ are:

<p>protection_level</p>	<p>The protection_level is readable and writeable, and provides a way to let userspace query the current protection level, as well as set the desired protection level, the available protection levels are:</p> <pre> =====      =====      ␣ ↪=====      ===== 0 - Disabled  1 - Low   2 - ␣ ↪Medium     3 - High =====      =====      ␣ ↪=====      =====                     </pre>
<p>reset_protection</p>	<p>The reset_protection entry is writeable only, being “1” the only parameter it accepts, it is used to trigger a reset of the protection interface.</p>

## **PARALLEL PORT LCD/KEYPAD PANEL SUPPORT**

Some LCDs allow you to define up to 8 characters, mapped to ASCII characters 0 to 7. The escape code to define a new character is ‘e[LG’ followed by one digit from 0 to 7, representing the character number, and up to 8 couples of hex digits terminated by a semi-colon ( ‘;’ ). Each couple of digits represents a line, with 1-bits for each illuminated pixel with LSB on the right. Lines are numbered from the top of the character to the bottom. On a 5x7 matrix, only the 5 lower bits of the 7 first bytes are used for each character. If the string is incomplete, only complete lines will be redefined. Here are some examples:

```
printf "\e[LG0010101050D1F0C04;" => 0 = [enter]
printf "\e[LG1040E1F0000000000;" => 1 = [up]
printf "\e[LG2000000001F0E0400;" => 2 = [down]
printf "\e[LG3040E1F001F0E0400;" => 3 = [up-down]
printf "\e[LG40002060E1E0E0602;" => 4 = [left]
printf "\e[LG500080C0E0F0E0C08;" => 5 = [right]
printf "\e[LG60016051516141400;" => 6 = "IP"

printf "\e[LG00103071F1F070301;" => big speaker
printf "\e[LG00002061E1E060200;" => small speaker
```

Willy



## **LDM - LOGICAL DISK MANAGER (DYNAMIC DISKS)**

**Author** Originally Written by FlatCap - Richard Russon  
<[ldm@flatcap.org](mailto:ldm@flatcap.org)>.

**Last Updated** Anton Altaparmakov on 30 March 2007 for Windows Vista.

### **49.1 Overview**

Windows 2000, XP, and Vista use a new partitioning scheme. It is a complete replacement for the MSDOS style partitions. It stores its information in a 1MiB journalled database at the end of the physical disk. The size of partitions is limited only by disk space. The maximum number of partitions is nearly 2000.

Any partitions created under the LDM are called “Dynamic Disks” . There are no longer any primary or extended partitions. Normal MSDOS style partitions are now known as Basic Disks.

If you wish to use Spanned, Striped, Mirrored or RAID 5 Volumes, you must use Dynamic Disks. The journalling allows Windows to make changes to these partitions and filesystems without the need to reboot.

Once the LDM driver has divided up the disk, you can use the MD driver to assemble any multi-partition volumes, e.g. Stripes, RAID5.

To prevent legacy applications from repartitioning the disk, the LDM creates a dummy MSDOS partition containing one disk-sized partition. This is what is supported with the Linux LDM driver.

A newer approach that has been implemented with Vista is to put LDM on top of a GPT label disk. This is not supported by the Linux LDM driver yet.

## 49.2 Example

Below we have a 50MiB disk, divided into seven partitions.

---

**Note:** The missing 1MiB at the end of the disk is where the LDM database is stored.

---

Device	Offset Bytes	Sectors	MiB	Size Bytes	Sectors	MiB
hda	0	0	0	52428800	102400	50
hda1	51380224	100352	49	1048576	2048	1
hda2	16384	32	0	6979584	13632	6
hda3	6995968	13664	6	10485760	20480	10
hda4	17481728	34144	16	4194304	8192	4
hda5	21676032	42336	20	5242880	10240	5
hda6	26918912	52576	25	10485760	20480	10
hda7	37404672	73056	35	13959168	27264	13

The LDM Database may not store the partitions in the order that they appear on disk, but the driver will sort them.

When Linux boots, you will see something like:

```
hda: 102400 sectors w/32KiB Cache, CHS=50/64/32
hda: [LDM] hda1 hda2 hda3 hda4 hda5 hda6 hda7
```

## 49.3 Compiling LDM Support

To enable LDM, choose the following two options:

- “Advanced partition selection” CONFIG\_PARTITION\_ADVANCED
- “Windows Logical Disk Manager (Dynamic Disk) support” CONFIG\_LDM\_PARTITION

If you believe the driver isn't working as it should, you can enable the extra debugging code. This will produce a LOT of output. The option is:

- “Windows LDM extra logging” CONFIG\_LDM\_DEBUG

N.B. The partition code cannot be compiled as a module.

As with all the partition code, if the driver doesn't see signs of its type of partition, it will pass control to another driver, so there is no harm in enabling it.

If you have Dynamic Disks but don't enable the driver, then all you will see is a dummy MSDOS partition filling the whole disk. You won't be able to mount any of the volumes on the disk.

## 49.4 Booting

If you enable LDM support, then lilo is capable of booting from any of the discovered partitions. However, grub does not understand the LDM partitioning and cannot boot from a Dynamic Disk.

## 49.5 More Documentation

There is an Overview of the LDM together with complete Technical Documentation. It is available for download.

<http://www.linux-ntfs.org/>

If you have any LDM questions that aren't answered in the documentation, email me.

**Cheers,** FlatCap - Richard Russon [ldm@flatcap.org](mailto:ldm@flatcap.org)



## SOFTLOCKUP DETECTOR AND HARDLOCKUP DETECTOR (AKA NMI\_WATCHDOG)

The Linux kernel can act as a watchdog to detect both soft and hard lockups.

A ‘softlockup’ is defined as a bug that causes the kernel to loop in kernel mode for more than 20 seconds (see “Implementation” below for details), without giving other tasks a chance to run. The current stack trace is displayed upon detection and, by default, the system will stay locked up. Alternatively, the kernel can be configured to panic; a `sysctl`, “`kernel.softlockup_panic`”, a kernel parameter, “`softlockup_panic`” (see “Documentation/admin-guide/kernel-parameters.rst” for details), and a compile option, “`BOOTPARAM_SOFTLOCKUP_PANIC`”, are provided for this.

A ‘hardlockup’ is defined as a bug that causes the CPU to loop in kernel mode for more than 10 seconds (see “Implementation” below for details), without letting other interrupts have a chance to run. Similarly to the softlockup case, the current stack trace is displayed upon detection and the system will stay locked up unless the default behavior is changed, which can be done through a `sysctl`, ‘`hardlockup_panic`’, a compile time knob, “`BOOTPARAM_HARDLOCKUP_PANIC`”, and a kernel parameter, “`nmi_watchdog`” (see “Documentation/admin-guide/kernel-parameters.rst” for details).

The panic option can be used in combination with `panic_timeout` (this timeout is set through the confusingly named “`kernel.panic`” `sysctl`), to cause the system to reboot automatically after a specified amount of time.

### 50.1 Implementation

The soft and hard lockup detectors are built on top of the `hrtimer` and `perf` subsystems, respectively. A direct consequence of this is that, in principle, they should work in any architecture where these subsystems are present.

A periodic `hrtimer` runs to generate interrupts and kick the watchdog task. An NMI `perf` event is generated every “`watchdog_thresh`” (compile-time initialized to 10 and configurable through `sysctl` of the same name) seconds to check for hardlockups. If any CPU in the system does not receive any `hrtimer` interrupt during that time the ‘hardlockup detector’ (the handler for the NMI `perf` event) will generate a kernel warning or call `panic`, depending on the configuration.

The watchdog task is a high priority kernel thread that updates a timestamp every time it is scheduled. If that timestamp is not updated for  $2 * \text{watchdog\_thresh}$

seconds (the softlockup threshold) the ‘softlockup detector’ (coded inside the hrtimer callback function) will dump useful debug information to the system log, after which it will call panic if it was instructed to do so or resume execution of other kernel code.

The period of the hrtimer is  $2 * \text{watchdog\_thresh} / 5$ , which means it has two or three chances to generate an interrupt before the hardlockup detector kicks in.

As explained above, a kernel knob is provided that allows administrators to configure the period of the hrtimer and the perf event. The right value for a particular environment is a trade-off between fast response to lockups and detection overhead.

By default, the watchdog runs on all online cores. However, on a kernel configured with `NO_HZ_FULL`, by default the watchdog runs only on the housekeeping cores, not the cores specified in the “nohz\_full” boot argument. If we allowed the watchdog to run by default on the “nohz\_full” cores, we would have to run timer ticks to activate the scheduler, which would prevent the “nohz\_full” functionality from protecting the user code on those cores from the kernel. Of course, disabling it by default on the nohz\_full cores means that when those cores do enter the kernel, by default we will not be able to detect if they lock up. However, allowing the watchdog to continue to run on the housekeeping (non-tickless) cores means that we will continue to detect lockups properly on those cores.

In either case, the set of cores excluded from running the watchdog may be adjusted via the `kernel.watchdog_cpumask` sysctl. For nohz\_full cores, this may be useful for debugging a case where the kernel seems to be hanging on the nohz\_full cores.

## **LINUX SECURITY MODULE USAGE**

The Linux Security Module (LSM) framework provides a mechanism for various security checks to be hooked by new kernel extensions. The name “module” is a bit of a misnomer since these extensions are not actually loadable kernel modules. Instead, they are selectable at build-time via `CONFIG_DEFAULT_SECURITY` and can be overridden at boot-time via the `"security=..."` kernel command line argument, in the case where multiple LSMs were built into a given kernel.

The primary users of the LSM interface are Mandatory Access Control (MAC) extensions which provide a comprehensive security policy. Examples include SELinux, Smack, Tomoyo, and AppArmor. In addition to the larger MAC extensions, other extensions can be built using the LSM to provide specific changes to system operation when these tweaks are not available in the core functionality of Linux itself.

The Linux capabilities modules will always be included. This may be followed by any number of “minor” modules and at most one “major” module. For more details on capabilities, see `capabilities(7)` in the Linux man-pages project.

A list of the active security modules can be found by reading `/sys/kernel/security/lsm`. This is a comma separated list, and will always include the capability module. The list reflects the order in which checks are made. The capability module will always be first, followed by any “minor” modules (e.g. Yama) and then the one “major” module (e.g. SELinux) if there is one configured.

Process attributes associated with “major” security modules should be accessed and maintained using the special files in `/proc/.../attr`. A security module may maintain a module specific subdirectory there, named after the module. `/proc/.../attr/smack` is provided by the Smack security module and contains all its special files. The files directly in `/proc/.../attr` remain as legacy interfaces for modules that provide subdirectories.

### **51.1 AppArmor**

#### **51.1.1 What is AppArmor?**

AppArmor is MAC style security extension for the Linux kernel. It implements a task centered policy, with task “profiles” being created and loaded from user space. Tasks on the system that do not have a profile defined for them run in an unconfined state which is equivalent to standard Linux DAC permissions.

### 51.1.2 How to enable/disable

set `CONFIG_SECURITY_APPARMOR=y`

If AppArmor should be selected as the default security module then set:

```
CONFIG_DEFAULT_SECURITY="apparmor"  
CONFIG_SECURITY_APPARMOR_BOOTPARAM_VALUE=1
```

Build the kernel

If AppArmor is not the default security module it can be enabled by passing `security=apparmor` on the kernel's command line.

If AppArmor is the default security module it can be disabled by passing `apparmor=0, security=XXXX` (where XXXX is valid security module), on the kernel's command line.

For AppArmor to enforce any restrictions beyond standard Linux DAC permissions policy must be loaded into the kernel from user space (see the Documentation and tools links).

### 51.1.3 Documentation

Documentation can be found on the wiki, linked below.

### 51.1.4 Links

Mailing List - [apparmor@lists.ubuntu.com](mailto:apparmor@lists.ubuntu.com)

Wiki - <http://wiki.apparmor.net>

User space tools - <https://gitlab.com/apparmor>

Kernel module - [git://git.kernel.org/pub/scm/linux/kernel/git/jj/linux-apparmor](https://git.kernel.org/pub/scm/linux/kernel/git/jj/linux-apparmor)

## 51.2 LoadPin

LoadPin is a Linux Security Module that ensures all kernel-loaded files (modules, firmware, etc) all originate from the same filesystem, with the expectation that such a filesystem is backed by a read-only device such as dm-verity or CDROM. This allows systems that have a verified and/or unchangeable filesystem to enforce module and firmware loading restrictions without needing to sign the files individually.

The LSM is selectable at build-time with `CONFIG_SECURITY_LOADPIN`, and can be controlled at boot-time with the kernel command line option `"loadpin.enabled"`. By default, it is enabled, but can be disabled at boot (`"loadpin.enabled=0"`).

LoadPin starts pinning when it sees the first file loaded. If the block device backing the filesystem is not read-only, a `sysctl` is created to toggle pinning: `/proc/sys/kernel/loadpin/enabled`. (Having a mutable filesystem means pinning is mutable too, but having the `sysctl` allows for easy testing on systems with a mutable filesystem.)

It's also possible to exclude specific file types from LoadPin using kernel command line option "loadpin.exclude". By default, all files are included, but they can be excluded using kernel command line option such as "loadpin.exclude=kernel-module,kexec-image". This allows to use different mechanisms such as CONFIG\_MODULE\_SIG and CONFIG\_KEXEC\_VERIFY\_SIG to verify kernel module and kernel image while still use LoadPin to protect the integrity of other files kernel loads. The full list of valid file types can be found in kernel\_read\_file\_str defined in include/linux/fs.h.

## 51.3 SELinux

If you want to use SELinux, chances are you will want to use the distro-provided policies, or install the latest reference policy release from

<https://github.com/SELinuxProject/refpolicy>

However, if you want to install a dummy policy for testing, you can do using mdp provided under scripts/selinux. Note that this requires the selinux userspace to be installed - in particular you will need checkpolicy to compile a kernel, and setfiles and fixfiles to label the filesystem.

1. Compile the kernel with selinux enabled.
2. Type make to compile mdp.
3. Make sure that you are not running with SELinux enabled and a real policy. If you are, reboot with selinux disabled before continuing.
4. Run install\_policy.sh:

```
cd scripts/selinux
sh install_policy.sh
```

Step 4 will create a new dummy policy valid for your kernel, with a single selinux user, role, and type. It will compile the policy, will set your SELINUXTYPE to dummy in /etc/selinux/config, install the compiled policy as dummy, and relabel your filesystem.

## 51.4 Smack

"Good for you, you've decided to clean the elevator!" - The Elevator,  
from Dark Star

Smack is the Simplified Mandatory Access Control Kernel. Smack is a kernel based implementation of mandatory access control that includes simplicity in its primary design goals.

Smack is not the only Mandatory Access Control scheme available for Linux. Those new to Mandatory Access Control are encouraged to compare Smack with the other mechanisms available to determine which is best suited to the problem at hand.

Smack consists of three major components:

- The kernel
- Basic utilities, which are helpful but not required
- Configuration data

The kernel component of Smack is implemented as a Linux Security Modules (LSM) module. It requires netlabel and works best with file systems that support extended attributes, although xattr support is not strictly required. It is safe to run a Smack kernel under a “vanilla” distribution.

Smack kernels use the CIPSO IP option. Some network configurations are intolerant of IP options and can impede access to systems that use them as Smack does.

Smack is used in the Tizen operating system. Please go to <http://wiki.tizen.org> for information about how Smack is used in Tizen.

The current git repository for Smack user space is:

```
git://github.com/smack-team/smack.git
```

This should make and install on most modern distributions. There are five commands included in smackutil:

**chsmack:** display or set Smack extended attribute values

**smackctl:** load the Smack access rules

**smackaccess:** report if a process with one label has access to an object with another

These two commands are obsolete with the introduction of the smackfs/load2 and smackfs/cipso2 interfaces.

**smackload:** properly formats data for writing to smackfs/load

**smackcipso:** properly formats data for writing to smackfs/cipso

In keeping with the intent of Smack, configuration data is minimal and not strictly required. The most important configuration step is mounting the smackfs pseudo filesystem. If smackutil is installed the startup script will take care of this, but it can be manually as well.

Add this line to /etc/fstab:

```
smackfs /sys/fs/smackfs smackfs defaults 0 0
```

The /sys/fs/smackfs directory is created by the kernel.

Smack uses extended attributes (xattrs) to store labels on filesystem objects. The attributes are stored in the extended attribute security name space. A process must have CAP\_MAC\_ADMIN to change any of these attributes.

The extended attributes that Smack uses are:

**SMACK64** Used to make access control decisions. In almost all cases the label given to a new filesystem object will be the label of the process that created it.

**SMACK64EXEC** The Smack label of a process that execs a program file with this attribute set will run with this attribute's value.

**SMACK64MMAP** Don't allow the file to be mmaped by a process whose Smack label does not allow all of the access permitted to a process with the label contained in this attribute. This is a very specific use case for shared libraries.

**SMACK64TRANSMUTE** Can only have the value "TRUE" . If this attribute is present on a directory when an object is created in the directory and the Smack rule (more below) that permitted the write access to the directory includes the transmute ( "t" ) mode the object gets the label of the directory instead of the label of the creating process. If the object being created is a directory the SMACK64TRANSMUTE attribute is set as well.

**SMACK64IPIN** This attribute is only available on file descriptors for sockets. Use the Smack label in this attribute for access control decisions on packets being delivered to this socket.

**SMACK64IPOUT** This attribute is only available on file descriptors for sockets. Use the Smack label in this attribute for access control decisions on packets coming from this socket.

There are multiple ways to set a Smack label on a file:

```
# attr -S -s SMACK64 -V "value" path
# chsmack -a value path
```

A process can see the Smack label it is running with by reading `/proc/self/attr/current`. A process with `CAP_MAC_ADMIN` can set the process Smack by writing there.

Most Smack configuration is accomplished by writing to files in the `smackfs` filesystem. This pseudo-filesystem is mounted on `/sys/fs/smackfs`.

**access** Provided for backward compatibility. The `access2` interface is preferred and should be used instead. This interface reports whether a subject with the specified Smack label has a particular access to an object with a specified Smack label. Write a fixed format access rule to this file. The next read will indicate whether the access would be permitted. The text will be either "1" indicating access, or "0" indicating denial.

**access2** This interface reports whether a subject with the specified Smack label has a particular access to an object with a specified Smack label. Write a long format access rule to this file. The next read will indicate whether the access would be permitted. The text will be either "1" indicating access, or "0" indicating denial.

**ambient** This contains the Smack label applied to unlabeled network packets.

**change-rule** This interface allows modification of existing access control rules. The format accepted on write is:

```
"%s %s %s %s"
```

where the first string is the subject label, the second the object label, the third the access to allow and the fourth the access to deny. The access strings may contain only the characters "rwxat-". If a rule for a given subject and object exists it will be modified by enabling the permissions in the third string and disabling those in the fourth string. If there is no such rule it will be created using the access specified in the third and the fourth strings.

**cipso** Provided for backward compatibility. The cipso2 interface is preferred and should be used instead. This interface allows a specific CIPSO header to be assigned to a Smack label. The format accepted on write is:

```
"%24s%4d%4d" ["%4d"] . . .
```

The first string is a fixed Smack label. The first number is the level to use. The second number is the number of categories. The following numbers are the categories:

```
"level-3-cats-5-19      3  2  5  19"
```

**cipso2** This interface allows a specific CIPSO header to be assigned to a Smack label. The format accepted on write is:

```
"%s%4d%4d" ["%4d"] . . .
```

The first string is a long Smack label. The first number is the level to use. The second number is the number of categories. The following numbers are the categories:

```
"level-3-cats-5-19  3  2  5  19"
```

**direct** This contains the CIPSO level used for Smack direct label representation in network packets.

**doi** This contains the CIPSO domain of interpretation used in network packets.

**ipv6host** This interface allows specific IPv6 internet addresses to be treated as single label hosts. Packets are sent to single label hosts only from processes that have Smack write access to the host label. All packets received from single label hosts are given the specified label. The format accepted on write is:

```
"%h:%h:%h:%h:%h:%h:%h:%h label" or  
"%h:%h:%h:%h:%h:%h:%h:%h/%d label".
```

The “:” address shortcut is not supported. If label is “-DELETE” a matched entry will be deleted.

**load** Provided for backward compatibility. The load2 interface is preferred and should be used instead. This interface allows access control rules in addition to the system defined rules to be specified. The format accepted on write is:

```
"%24s%24s%5s"
```

where the first string is the subject label, the second the object label, and the third the requested access. The access string may contain only the characters “rwxat-”, and specifies which sort of access is allowed. The “-” is a placeholder for permissions that are not allowed. The string “r-x-” would specify read and execute access. Labels are limited to 23 characters in length.

**load2** This interface allows access control rules in addition to the system defined rules to be specified. The format accepted on write is:

```
"%s %s %s"
```

where the first string is the subject label, the second the object label, and the third the requested access. The access string may contain only the characters “`rwxt-`”, and specifies which sort of access is allowed. The “`-`” is a placeholder for permissions that are not allowed. The string “`r-x-`” would specify read and execute access.

**load-self** Provided for backward compatibility. The `load-self2` interface is preferred and should be used instead. This interface allows process specific access rules to be defined. These rules are only consulted if access would otherwise be permitted, and are intended to provide additional restrictions on the process. The format is the same as for the `load` interface.

**load-self2** This interface allows process specific access rules to be defined. These rules are only consulted if access would otherwise be permitted, and are intended to provide additional restrictions on the process. The format is the same as for the `load2` interface.

**logging** This contains the Smack logging state.

**mapped** This contains the CIPSO level used for Smack mapped label representation in network packets.

**netlabel** This interface allows specific internet addresses to be treated as single label hosts. Packets are sent to single label hosts without CIPSO headers, but only from processes that have Smack write access to the host label. All packets received from single label hosts are given the specified label. The format accepted on write is:

```
"%d.%d.%d.%d label" or "%d.%d.%d.%d/%d label".
```

If the label specified is “`-CIPSO`” the address is treated as a host that supports CIPSO headers.

**onlycap** This contains labels processes must have for `CAP_MAC_ADMIN` and `CAP_MAC_OVERRIDE` to be effective. If this file is empty these capabilities are effective at for processes with any label. The values are set by writing the desired labels, separated by spaces, to the file or cleared by writing “`-`” to the file.

**ptrace** This is used to define the current ptrace policy

**0 - default:** this is the policy that relies on Smack access rules. For the `PTRACE_READ` a subject needs to have a read access on object. For the `PTRACE_ATTACH` a read-write access is required.

**1 - exact:** this is the policy that limits `PTRACE_ATTACH`. Attach is only allowed when subject’s and object’s labels are equal. `PTRACE_READ` is not affected. Can be overridden with `CAP_SYS_PTRACE`.

**2 - draconian:** this policy behaves like the ‘exact’ above with an exception that it can’t be overridden with `CAP_SYS_PTRACE`.

**revoke-subject** Writing a Smack label here sets the access to ‘-’ for all access rules with that subject label.

**unconfined** If the kernel is configured with `CONFIG_SECURITY_SMACK_BRINGUP` a process with `CAP_MAC_ADMIN` can write a label into this interface. Thereafter, accesses that involve that label will be logged and the access permitted if it wouldn't be otherwise. Note that this is dangerous and can ruin the proper labeling of your system. It should never be used in production.

**relabel-self** This interface contains a list of labels to which the process can transition to, by writing to `/proc/self/attr/current`. Normally a process can change its own label to any legal value, but only if it has `CAP_MAC_ADMIN`. This interface allows a process without `CAP_MAC_ADMIN` to relabel itself to one of labels from predefined list. A process without `CAP_MAC_ADMIN` can change its label only once. When it does, this list will be cleared. The values are set by writing the desired labels, separated by spaces, to the file or cleared by writing `"-"` to the file.

If you are using the `smackload` utility you can add access rules in `/etc/smack/accesses`. They take the form:

```
subjectlabel objectlabel access
```

`access` is a combination of the letters `rwxtab` which specify the kind of access permitted a subject with `subjectlabel` on an object with `objectlabel`. If there is no rule no access is allowed.

Look for additional programs on <http://schaufler-ca.com>

### 51.4.1 The Simplified Mandatory Access Control Kernel (Whitepaper)

Casey Schaufler [casey@schaufler-ca.com](mailto:casey@schaufler-ca.com)

#### Mandatory Access Control

Computer systems employ a variety of schemes to constrain how information is shared among the people and services using the machine. Some of these schemes allow the program or user to decide what other programs or users are allowed access to pieces of data. These schemes are called discretionary access control mechanisms because the access control is specified at the discretion of the user. Other schemes do not leave the decision regarding what a user or program can access up to users or programs. These schemes are called mandatory access control mechanisms because you don't have a choice regarding the users or programs that have access to pieces of data.

## Bell & LaPadula

From the middle of the 1980's until the turn of the century Mandatory Access Control (MAC) was very closely associated with the Bell & LaPadula security model, a mathematical description of the United States Department of Defense policy for marking paper documents. MAC in this form enjoyed a following within the Capital Beltway and Scandinavian supercomputer centers but was often sited as failing to address general needs.

## Domain Type Enforcement

Around the turn of the century Domain Type Enforcement (DTE) became popular. This scheme organizes users, programs, and data into domains that are protected from each other. This scheme has been widely deployed as a component of popular Linux distributions. The administrative overhead required to maintain this scheme and the detailed understanding of the whole system necessary to provide a secure domain mapping leads to the scheme being disabled or used in limited ways in the majority of cases.

## Smack

Smack is a Mandatory Access Control mechanism designed to provide useful MAC while avoiding the pitfalls of its predecessors. The limitations of Bell & LaPadula are addressed by providing a scheme whereby access can be controlled according to the requirements of the system and its purpose rather than those imposed by an arcane government policy. The complexity of Domain Type Enforcement and avoided by defining access controls in terms of the access modes already in use.

## Smack Terminology

The jargon used to talk about Smack will be familiar to those who have dealt with other MAC systems and shouldn't be too difficult for the uninitiated to pick up. There are four terms that are used in a specific way and that are especially important:

**Subject:** A subject is an active entity on the computer system. On Smack a subject is a task, which is in turn the basic unit of execution.

**Object:** An object is a passive entity on the computer system. On Smack files of all types, IPC, and tasks can be objects.

**Access:** Any attempt by a subject to put information into or get information from an object is an access.

**Label:** Data that identifies the Mandatory Access Control characteristics of a subject or an object.

These definitions are consistent with the traditional use in the security community. There are also some terms from Linux that are likely to crop up:

**Capability:** A task that possesses a capability has permission to violate an aspect of the system security policy, as identified by the specific capability. A task that possesses one or more capabilities is a privileged task, whereas a task with no capabilities is an unprivileged task.

**Privilege:** A task that is allowed to violate the system security policy is said to have privilege. As of this writing a task can have privilege either by possessing capabilities or by having an effective user of root.

### Smack Basics

Smack is an extension to a Linux system. It enforces additional restrictions on what subjects can access which objects, based on the labels attached to each of the subject and the object.

### Labels

Smack labels are ASCII character strings. They can be up to 255 characters long, but keeping them to twenty-three characters is recommended. Single character labels using special characters, that being anything other than a letter or digit, are reserved for use by the Smack development team. Smack labels are unstructured, case sensitive, and the only operation ever performed on them is comparison for equality. Smack labels cannot contain unprintable characters, the “/” (slash), the “\” (backslash), the “” (quote) and “” (double-quote) characters. Smack labels cannot begin with a ‘-’. This is reserved for special options.

There are some predefined labels:

<code>_</code>	Pronounced "floor", a single underscore character.
<code>^</code>	Pronounced "hat", a single circumflex character.
<code>*</code>	Pronounced "star", a single asterisk character.
<code>?</code>	Pronounced "huh", a single question mark character.
<code>@</code>	Pronounced "web", a single at sign character.

Every task on a Smack system is assigned a label. The Smack label of a process will usually be assigned by the system initialization mechanism.

### Access Rules

Smack uses the traditional access modes of Linux. These modes are read, execute, write, and occasionally append. There are a few cases where the access mode may not be obvious. These include:

**Signals:** A signal is a write operation from the subject task to the object task.

**Internet Domain IPC:** Transmission of a packet is considered a write operation from the source task to the destination task.

Smack restricts access based on the label attached to a subject and the label attached to the object it is trying to access. The rules enforced are, in order:

1. Any access requested by a task labeled “\*” is denied.
2. A read or execute access requested by a task labeled “^” is permitted.
3. A read or execute access requested on an object labeled “\_” is permitted.
4. Any access requested on an object labeled “\*” is permitted.
5. Any access requested by a task on an object with the same label is permitted.
6. Any access requested that is explicitly defined in the loaded rule set is permitted.
7. Any other access is denied.

## Smack Access Rules

With the isolation provided by Smack access separation is simple. There are many interesting cases where limited access by subjects to objects with different labels is desired. One example is the familiar spy model of sensitivity, where a scientist working on a highly classified project would be able to read documents of lower classifications and anything she writes will be “born” highly classified. To accommodate such schemes Smack includes a mechanism for specifying rules allowing access between labels.

## Access Rule Format

The format of an access rule is:

```
subject-label object-label access
```

Where subject-label is the Smack label of the task, object-label is the Smack label of the thing being accessed, and access is a string specifying the sort of access allowed. The access specification is searched for letters that describe access modes:

a: indicates that append access should be granted. r: indicates that read access should be granted. w: indicates that write access should be granted. x: indicates that execute access should be granted. t: indicates that the rule requests transmutation. b: indicates that the rule should be reported for bring-up.

Uppercase values for the specification letters are allowed as well. Access mode specifications can be in any order. Examples of acceptable rules are:

```
TopSecret Secret rx
Secret Unclass R
Manager Game x
User HR w
Snap Crackle rwxatb
New Old rRrRr
Closed Off -
```

Examples of unacceptable rules are:

Top Secret	Secret	rx
Ace	Ace	r
Odd	spells	waxbeans

Spaces are not allowed in labels. Since a subject always has access to files with the same label specifying a rule for that case is pointless. Only valid letters (rwx-atbRWXATB) and the dash ( ‘ - ’ ) character are allowed in access specifications. The dash is a placeholder, so “a-r” is the same as “ar” . A lone dash is used to specify that no access should be allowed.

### Applying Access Rules

The developers of Linux rarely define new sorts of things, usually importing schemes and concepts from other systems. Most often, the other systems are variants of Unix. Unix has many endearing properties, but consistency of access control models is not one of them. Smack strives to treat accesses as uniformly as is sensible while keeping with the spirit of the underlying mechanism.

File system objects including files, directories, named pipes, symbolic links, and devices require access permissions that closely match those used by mode bit access. To open a file for reading read access is required on the file. To search a directory requires execute access. Creating a file with write access requires both read and write access on the containing directory. Deleting a file requires read and write access to the file and to the containing directory. It is possible that a user may be able to see that a file exists but not any of its attributes by the circumstance of having read access to the containing directory but not to the differently labeled file. This is an artifact of the file name being data in the directory, not a part of the file.

If a directory is marked as transmuting (SMACK64TRANSMUTE=TRUE) and the access rule that allows a process to create an object in that directory includes ‘t’ access the label assigned to the new object will be that of the directory, not the creating process. This makes it much easier for two processes with different labels to share data without granting access to all of their files.

IPC objects, message queues, semaphore sets, and memory segments exist in flat namespaces and access requests are only required to match the object in question.

Process objects reflect tasks on the system and the Smack label used to access them is the same Smack label that the task would use for its own access attempts. Sending a signal via the kill() system call is a write operation from the signaler to the recipient. Debugging a process requires both reading and writing. Creating a new task is an internal operation that results in two tasks with identical Smack labels and requires no access checks.

Sockets are data structures attached to processes and sending a packet from one process to another requires that the sender have write access to the receiver. The receiver is not required to have read access to the sender.

## Setting Access Rules

The configuration file `/etc/smack/accesses` contains the rules to be set at system startup. The contents are written to the special file `/sys/fs/smackfs/load2`. Rules can be added at any time and take effect immediately. For any pair of subject and object labels there can be only one rule, with the most recently specified overriding any earlier specification.

## Task Attribute

The Smack label of a process can be read from `/proc/<pid>/attr/current`. A process can read its own Smack label from `/proc/self/attr/current`. A privileged process can change its own Smack label by writing to `/proc/self/attr/current` but not the label of another process.

## File Attribute

The Smack label of a filesystem object is stored as an extended attribute named `SMACK64` on the file. This attribute is in the security namespace. It can only be changed by a process with privilege.

## Privilege

A process with `CAP_MAC_OVERRIDE` or `CAP_MAC_ADMIN` is privileged. `CAP_MAC_OVERRIDE` allows the process access to objects it would be denied otherwise. `CAP_MAC_ADMIN` allows a process to change Smack data, including rules and attributes.

## Smack Networking

As mentioned before, Smack enforces access control on network protocol transmissions. Every packet sent by a Smack process is tagged with its Smack label. This is done by adding a CIPSO tag to the header of the IP packet. Each packet received is expected to have a CIPSO tag that identifies the label and if it lacks such a tag the network ambient label is assumed. Before the packet is delivered a check is made to determine that a subject with the label on the packet has write access to the receiving process and if that is not the case the packet is dropped.

### CIPSO Configuration

It is normally unnecessary to specify the CIPSO configuration. The default values used by the system handle all internal cases. Smack will compose CIPSO label values to match the Smack labels being used without administrative intervention. Unlabeled packets that come into the system will be given the ambient label.

Smack requires configuration in the case where packets from a system that is not Smack that speaks CIPSO may be encountered. Usually this will be a Trusted Solaris system, but there are other, less widely deployed systems out there. CIPSO provides 3 important values, a Domain Of Interpretation (DOI), a level, and a category set with each packet. The DOI is intended to identify a group of systems that use compatible labeling schemes, and the DOI specified on the Smack system must match that of the remote system or packets will be discarded. The DOI is 3 by default. The value can be read from `/sys/fs/smackfs/doi` and can be changed by writing to `/sys/fs/smackfs/doi`.

The label and category set are mapped to a Smack label as defined in `/etc/smack/cipso`.

A Smack/CIPSO mapping has the form:

```
smack level [category [category]*]
```

Smack does not expect the level or category sets to be related in any particular way and does not assume or assign accesses based on them. Some examples of mappings:

```
TopSecret 7
TS:A,B    7 1 2
SecBDE    5 2 4 6
RAFTERS   7 12 26
```

The “:” and “,” characters are permitted in a Smack label but have no special meaning.

The mapping of Smack labels to CIPSO values is defined by writing to `/sys/fs/smackfs/cipso2`.

In addition to explicit mappings Smack supports direct CIPSO mappings. One CIPSO level is used to indicate that the category set passed in the packet is in fact an encoding of the Smack label. The level used is 250 by default. The value can be read from `/sys/fs/smackfs/direct` and changed by writing to `/sys/fs/smackfs/direct`.

### Socket Attributes

There are two attributes that are associated with sockets. These attributes can only be set by privileged tasks, but any task can read them for their own sockets.

**SMACK64IPIN:** The Smack label of the task object. A privileged program that will enforce policy may set this to the star label.

**SMACK64IPOUT:** The Smack label transmitted with outgoing packets. A privileged program may set this to match the label of another task with which it hopes to communicate.

## Smack Netlabel Exceptions

You will often find that your labeled application has to talk to the outside, unlabeled world. To do this there's a special file `/sys/fs/smackfs/netlabel` where you can add some exceptions in the form of:

```
@IP1 LABEL1 or
@IP2/MASK LABEL2
```

It means that your application will have unlabeled access to `@IP1` if it has write access on `LABEL1`, and access to the subnet `@IP2/MASK` if it has write access on `LABEL2`.

Entries in the `/sys/fs/smackfs/netlabel` file are matched by longest mask first, like in classless IPv4 routing.

A special label '@' and an option '-CIPSO' can be used there:

```
@ means Internet, any application with any label has access to it
-CIPSO means standard CIPSO networking
```

If you don't know what CIPSO is and don't plan to use it, you can just do:

```
echo 127.0.0.1 -CIPSO > /sys/fs/smackfs/netlabel
echo 0.0.0.0/0 @ > /sys/fs/smackfs/netlabel
```

If you use CIPSO on your 192.168.0.0/16 local network and need also unlabeled Internet access, you can have:

```
echo 127.0.0.1 -CIPSO > /sys/fs/smackfs/netlabel
echo 192.168.0.0/16 -CIPSO > /sys/fs/smackfs/netlabel
echo 0.0.0.0/0 @ > /sys/fs/smackfs/netlabel
```

## Writing Applications for Smack

There are three sorts of applications that will run on a Smack system. How an application interacts with Smack will determine what it will have to do to work properly under Smack.

### Smack Ignorant Applications

By far the majority of applications have no reason whatever to care about the unique properties of Smack. Since invoking a program has no impact on the Smack label associated with the process the only concern likely to arise is whether the process has execute access to the program.

### Smack Relevant Applications

Some programs can be improved by teaching them about Smack, but do not make any security decisions themselves. The utility `ls(1)` is one example of such a program.

### Smack Enforcing Applications

These are special programs that not only know about Smack, but participate in the enforcement of system policy. In most cases these are the programs that set up user sessions. There are also network services that provide information to processes running with various labels.

### File System Interfaces

Smack maintains labels on file system objects using extended attributes. The Smack label of a file, directory, or other file system object can be obtained using `getxattr(2)`:

```
len = getxattr("/", "security.SMACK64", value, sizeof (value));
```

will put the Smack label of the root directory into `value`. A privileged process can set the Smack label of a file system object with `setxattr(2)`:

```
len = strlen("Rubble");  
rc = setxattr("/foo", "security.SMACK64", "Rubble", len, 0);
```

will set the Smack label of `/foo` to “Rubble” if the program has appropriate privilege.

### Socket Interfaces

The socket attributes can be read using `fgetxattr(2)`.

A privileged process can set the Smack label of outgoing packets with `fsetxattr(2)`:

```
len = strlen("Rubble");  
rc = fsetxattr(fd, "security.SMACK64IPOUT", "Rubble", len, 0);
```

will set the Smack label “Rubble” on packets going out from the socket if the program has appropriate privilege:

```
rc = fsetxattr(fd, "security.SMACK64IPIN, "*", strlen("*"), 0);
```

will set the Smack label “\*” as the object label against which incoming packets will be checked if the program has appropriate privilege.

## Administration

Smack supports some mount options:

**smackfsdef=label:** specifies the label to give files that lack the Smack label extended attribute.

**smackfsroot=label:** specifies the label to assign the root of the file system if it lacks the Smack extended attribute.

**smackfshat=label:** specifies a label that must have read access to all labels set on the filesystem. Not yet enforced.

**smackfsfloor=label:** specifies a label to which all labels set on the filesystem must have read access. Not yet enforced.

**smackfstransmute=label:** behaves exactly like smackfsroot except that it also sets the transmute flag on the root of the mount

These mount options apply to all file system types.

## Smack auditing

If you want Smack auditing of security events, you need to set `CONFIG_AUDIT` in your kernel configuration. By default, all denied events will be audited. You can change this behavior by writing a single character to the `/sys/fs/smackfs/logging` file:

```
0 : no logging
1 : log denied (default)
2 : log accepted
3 : log denied & accepted
```

Events are logged as ‘key=value’ pairs, for each event you at least will get the subject, the object, the rights requested, the action, the kernel function that triggered the event, plus other pairs depending on the type of event audited.

## Bringup Mode

Bringup mode provides logging features that can make application configuration and system bringup easier. Configure the kernel with `CONFIG_SECURITY_SMACK_BRINGUP` to enable these features. When bringup mode is enabled accesses that succeed due to rules marked with the “b” access mode will be logged. When a new label is introduced for processes rules can be added aggressively, marked with the “b”. The logging allows tracking of which rules actually get used for that label.

Another feature of bringup mode is the “unconfined” option. Writing a label to `/sys/fs/smackfs/unconfined` makes subjects with that label able to access any object, and objects with that label accessible to all subjects. Any access that is granted because a label is unconfined is logged. This feature is dangerous, as files and directories may be created in places they couldn’t if the policy were being enforced.

## 51.5 TOMOYO

### 51.5.1 What is TOMOYO?

TOMOYO is a name-based MAC extension (LSM module) for the Linux kernel.

LiveCD-based tutorials are available at

<http://tomoyo.sourceforge.jp/1.8/ubuntu12.04-live.html>                      <http://tomoyo.sourceforge.jp/1.8/centos6-live.html>

Though these tutorials use non-LSM version of TOMOYO, they are useful for you to know what TOMOYO is.

### 51.5.2 How to enable TOMOYO?

Build the kernel with `CONFIG_SECURITY_TOMOYO=y` and pass `security=tomoyo` on kernel' s command line.

Please see <http://tomoyo.osdn.jp/2.5/> for details.

### 51.5.3 Where is documentation?

User <-> Kernel interface documentation is available at <https://tomoyo.osdn.jp/2.5/policy-specification/index.html> .

Materials we prepared for seminars and symposiums are available at [https://osdn.jp/projects/tomoyo/docs/?category\\_id=532&language\\_id=1](https://osdn.jp/projects/tomoyo/docs/?category_id=532&language_id=1) . Below lists are chosen from three aspects.

#### What is TOMOYO?

**TOMOYO Linux Overview** <https://osdn.jp/projects/tomoyo/docs/lca2009-takeda.pdf>

**TOMOYO Linux: pragmatic and manageable security for Linux**  
<https://osdn.jp/projects/tomoyo/docs/freedomhetapei-tomoyo.pdf>

**TOMOYO Linux: A Practical Method to Understand and Protect Your Own Linux F**  
<https://osdn.jp/projects/tomoyo/docs/PacSec2007-en-no-demo.pdf>

#### What can TOMOYO do?

**Deep inside TOMOYO Linux** <https://osdn.jp/projects/tomoyo/docs/lca2009-kumaneko.pdf>

**The role of “pathname based access control” in security.** <https://osdn.jp/projects/tomoyo/docs/lfj2008-bof.pdf>

#### History of TOMOYO?

**Realities of Mainlining** <https://osdn.jp/projects/tomoyo/docs/lfj2008.pdf>

### 51.5.4 What is future plan?

We believe that inode based security and name based security are complementary and both should be used together. But unfortunately, so far, we cannot enable multiple LSM modules at the same time. We feel sorry that you have to give up SELinux/SMACK/AppArmor etc. when you want to use TOMOYO.

We hope that LSM becomes stackable in future. Meanwhile, you can use non-LSM version of TOMOYO, available at <http://tomoyo.osdn.jp/1.8/> . LSM version of TOMOYO is a subset of non-LSM version of TOMOYO. We are planning to port non-LSM version' s functionalities to LSM versions.

## 51.6 Yama

Yama is a Linux Security Module that collects system-wide DAC security protections that are not handled by the core kernel itself. This is selectable at build-time with `CONFIG_SECURITY_YAMA`, and can be controlled at run-time through sysctls in `/proc/sys/kernel/yama`:

### 51.6.1 ptrace\_scope

As Linux grows in popularity, it will become a larger target for malware. One particularly troubling weakness of the Linux process interfaces is that a single user is able to examine the memory and running state of any of their processes. For example, if one application (e.g. Pidgin) was compromised, it would be possible for an attacker to attach to other running processes (e.g. Firefox, SSH sessions, GPG agent, etc) to extract additional credentials and continue to expand the scope of their attack without resorting to user-assisted phishing.

This is not a theoretical problem. SSH session hijacking (<http://www.storm.net.nz/projects/7>) and arbitrary code injection (<http://c-skills.blogspot.com/2007/05/injectso.html>) attacks already exist and remain possible if ptrace is allowed to operate as before. Since ptrace is not commonly used by non-developers and non-admins, system builders should be allowed the option to disable this debugging system.

For a solution, some applications use `prctl(PR_SET_DUMPABLE, ...)` to specifically disallow such ptrace attachment (e.g. ssh-agent), but many do not. A more general solution is to only allow ptrace directly from a parent to a child process (i.e. direct “gdb EXE” and “strace EXE” still work), or with `CAP_SYS_PTRACE` (i.e. “gdb -pid=PID” , and “strace -p PID” still work as root).

In mode 1, software that has defined application-specific relationships between a debugging process and its inferior (crash handlers, etc), `prctl(PR_SET_PTRACER, pid, ...)` can be used. An inferior can declare which other process (and its descendants) are allowed to call `PTRACE_ATTACH` against it. Only one such declared debugging process can exist for each inferior at a time. For example, this is used by KDE, Chromium, and Firefox' s crash handlers, and by Wine for allowing only Wine processes to ptrace each other. If a process wishes to entirely disable these ptrace restrictions, it can call `prctl(PR_SET_PTRACER, PR_SET_PTRACER_ANY, .`

. . ) so that any otherwise allowed process (even those in external pid namespaces) may attach.

The sysctl settings (writable only with CAP\_SYS\_PTRACE) are:

- 0 - classic ptrace permissions:** a process can PTRACE\_ATTACH to any other process running under the same uid, as long as it is dumpable (i.e. did not transition uids, start privileged, or have called `prctl(PR_SET_DUMPABLE...)` already). Similarly, PTRACE\_TRACEME is unchanged.
- 1 - restricted ptrace:** a process must have a predefined relationship with the inferior it wants to call PTRACE\_ATTACH on. By default, this relationship is that of only its descendants when the above classic criteria is also met. To change the relationship, an inferior can call `prctl(PR_SET_PTRACER, debugger, . . .)` to declare an allowed debugger PID to call PTRACE\_ATTACH on the inferior. Using PTRACE\_TRACEME is unchanged.
- 2 - admin-only attach:** only processes with CAP\_SYS\_PTRACE may use ptrace, either with PTRACE\_ATTACH or through children calling PTRACE\_TRACEME.
- 3 - no attach:** no processes may use ptrace with PTRACE\_ATTACH nor via PTRACE\_TRACEME. Once set, this sysctl value cannot be changed.

The original children-only logic was based on the restrictions in grsecurity.

## 51.7 SafeSetID

SafeSetID is an LSM module that gates the setid family of syscalls to restrict UID/GID transitions from a given UID/GID to only those approved by a system-wide whitelist. These restrictions also prohibit the given UIDs/GIDs from obtaining auxiliary privileges associated with CAP\_SET{U/G}ID, such as allowing a user to set up user namespace UID mappings.

### 51.7.1 Background

In absence of file capabilities, processes spawned on a Linux system that need to switch to a different user must be spawned with CAP\_SETUID privileges. CAP\_SETUID is granted to programs running as root or those running as a non-root user that have been explicitly given the CAP\_SETUID runtime capability. It is often preferable to use Linux runtime capabilities rather than file capabilities, since using file capabilities to run a program with elevated privileges opens up possible security holes since any user with access to the file can `exec()` that program to gain the elevated privileges.

While it is possible to implement a tree of processes by giving full CAP\_SET{U/G}ID capabilities, this is often at odds with the goals of running a tree of processes under non-root user(s) in the first place. Specifically, since CAP\_SETUID allows changing to any user on the system, including the root user, it is an overpowered capability for what is needed in this scenario, especially since programs often only call `setuid()` to drop privileges to a lesser-privileged user – not elevate privileges. Unfortunately, there is no generally feasible way in Linux to restrict the potential UIDs that a user can switch to through `setuid()` beyond

allowing a switch to any user on the system. This SafeSetID LSM seeks to provide a solution for restricting setid capabilities in such a way.

The main use case for this LSM is to allow a non-root program to transition to other untrusted uids without full blown CAP\_SETUID capabilities. The non-root program would still need CAP\_SETUID to do any kind of transition, but the additional restrictions imposed by this LSM would mean it is a “safer” version of CAP\_SETUID since the non-root program cannot take advantage of CAP\_SETUID to do any unapproved actions (e.g. setuid to uid 0 or create/enter new user namespace). The higher level goal is to allow for uid-based sandboxing of system services without having to give out CAP\_SETUID all over the place just so that non-root programs can drop to even-lesser-privileged uids. This is especially relevant when one non-root daemon on the system should be allowed to spawn other processes as different uids, but its undesirable to give the daemon a basically-root-equivalent CAP\_SETUID.

## 51.7.2 Other Approaches Considered

### Solve this problem in userspace

For candidate applications that would like to have restricted setid capabilities as implemented in this LSM, an alternative option would be to simply take away setid capabilities from the application completely and refactor the process spawning semantics in the application (e.g. by using a privileged helper program to do process spawning and UID/GID transitions). Unfortunately, there are a number of semantics around process spawning that would be affected by this, such as fork() calls where the program doesn't immediately call exec() after the fork(), parent processes specifying custom environment variables or command line args for spawned child processes, or inheritance of file handles across a fork()/exec(). Because of this, as solution that uses a privileged helper in userspace would likely be less appealing to incorporate into existing projects that rely on certain process-spawning semantics in Linux.

### Use user namespaces

Another possible approach would be to run a given process tree in its own user namespace and give programs in the tree setid capabilities. In this way, programs in the tree could change to any desired UID/GID in the context of their own user namespace, and only approved UIDs/GIDs could be mapped back to the initial system user namespace, affectively preventing privilege escalation. Unfortunately, it is not generally feasible to use user namespaces in isolation, without pairing them with other namespace types, which is not always an option. Linux checks for capabilities based off of the user namespace that “owns” some entity. For example, Linux has the notion that network namespaces are owned by the user namespace in which they were created. A consequence of this is that capability checks for access to a given network namespace are done by checking whether a task has the given capability in the context of the user namespace that owns the network namespace - not necessarily the user namespace under which the given task runs. Therefore spawning a process in a new user namespace effectively prevents it from

accessing the network namespace owned by the initial namespace. This is a deal-breaker for any application that expects to retain the `CAP_NET_ADMIN` capability for the purpose of adjusting network configurations. Using user namespaces in isolation causes problems regarding other system interactions, including use of pid namespaces and device creation.

### Use an existing LSM

None of the other in-tree LSMs have the capability to gate `setuid` transitions, or even employ the `security_task_fix_setuid` hook at all. SELinux says of that hook: “Since `setuid` only affects the current process, and since the SELinux controls are not based on the Linux identity attributes, SELinux does not need to control this operation.”

### 51.7.3 Directions for use

This LSM hooks the `setid` syscalls to make sure transitions are allowed if an applicable restriction policy is in place. Policies are configured through `securityfs` by writing to the `safesetid/add_whitelist_policy` and `safesetid/flush_whitelist_policies` files at the location where `securityfs` is mounted. The format for adding a policy is ‘<UID>:<UID>’, using literal numbers, such as ‘123:456’. To flush the policies, any write to the file is sufficient. Again, configuring a policy for a UID will prevent that UID from obtaining auxiliary `setid` privileges, such as allowing a user to set up user namespace UID mappings.

## RAID ARRAYS

### 52.1 Boot time assembly of RAID arrays

Tools that manage md devices can be found at <https://www.kernel.org/pub/linux/utils/raid/>

You can boot with your md device with the following kernel command lines:

for old raid arrays without persistent superblocks:

```
md=<md device no.>,<raid level>,<chunk size factor>,<fault level>,dev0,  
↔dev1,...,devn
```

for raid arrays with persistent superblocks:

```
md=<md device no.>,dev0,dev1,...,devn
```

or, to assemble a partitionable array:

```
md=d<md device no.>,dev0,dev1,...,devn
```

#### 52.1.1 md device no.

The number of the md device

md device no.	device
0	md0
1	md1
2	md2
3	md3
4	md4

### 52.1.2 raid level

level of the RAID array

raid level	level
-1	linear mode
0	striped mode

other modes are only supported with persistent super blocks

### 52.1.3 chunk size factor

(raid-0 and raid-1 only)

Set the chunk size as  $4k \ll n$ .

### 52.1.4 fault level

Totally ignored

### 52.1.5 dev0 to devn

e.g. /dev/hda1, /dev/hdc1, /dev/sda1, /dev/sdb1

A possible loadlin line (Harald Hoyer <[HarryH@Royal.Net](mailto:HarryH@Royal.Net)>) looks like this:

```
e:\loadlin\loadlin e:\zimage root=/dev/md0 md=0,0,4,0,/dev/hdb2,/dev/hdc3  
→ ro
```

## 52.2 Boot time autodetection of RAID arrays

When md is compiled into the kernel (not as module), partitions of type 0xfd are scanned and automatically assembled into RAID arrays. This autodetection may be suppressed with the kernel parameter `raid=noautodetect`. As of kernel 2.6.9, only drives with a type 0 superblock can be autodetected and run at boot time.

The kernel parameter `raid=partitionable` (or `raid=part`) means that all auto-detected arrays are assembled as partitionable.

## 52.3 Boot time assembly of degraded/dirty arrays

If a raid5 or raid6 array is both dirty and degraded, it could have undetectable data corruption. This is because the fact that it is dirty means that the parity cannot be trusted, and the fact that it is degraded means that some datablocks are missing and cannot reliably be reconstructed (due to no parity).

For this reason, md will normally refuse to start such an array. This requires the sysadmin to take action to explicitly start the array despite possible corruption. This is normally done with:

```
mdadm --assemble --force ....
```

This option is not really available if the array has the root filesystem on it. In order to support this booting from such an array, md supports a module parameter `start_dirty_degraded` which, when set to 1, bypasses the checks and will allow dirty degraded arrays to be started.

So, to boot with a root filesystem of a dirty degraded raid 5 or 6, use:

```
md-mod.start_dirty_degraded=1
```

## 52.4 Superblock formats

The md driver can support a variety of different superblock formats. Currently, it supports superblock formats 0.90.0 and the md-1 format introduced in the 2.5 development series.

The kernel will autodetect which format superblock is being used.

Superblock format 0 is treated differently to others for legacy reasons - it is the original superblock format.

## 52.5 General Rules - apply for all superblock formats

An array is created by writing appropriate superblocks to all devices.

It is assembled by associating each of these devices with an particular md virtual device. Once it is completely assembled, it can be accessed.

An array should be created by a user-space tool. This will write superblocks to all devices. It will usually mark the array as `unclean`, or with some devices missing so that the kernel md driver can create appropriate redundancy (copying in raid 1, parity calculation in raid 4/5).

When an array is assembled, it is first initialized with the `SET_ARRAY_INFO` ioctl. This contains, in particular, a major and minor version number. The major version number selects which superblock format is to be used. The minor number might be used to tune handling of the format, such as suggesting where on each device to look for the superblock.

Then each device is added using the `ADD_NEW_DISK` ioctl. This provides, in particular, a major and minor number identifying the device to add.

The array is started with the `RUN_ARRAY` ioctl.

Once started, new devices can be added. They should have an appropriate superblock written to them, and then be passed in with `ADD_NEW_DISK`.

Devices that have failed or are not yet active can be detached from an array using `HOT_REMOVE_DISK`.

## 52.6 Specific Rules that apply to format-0 super block arrays, and arrays with no superblock (non-persistent)

An array can be created by describing the array (level, chunksize etc) in a SET\_ARRAY\_INFO ioctl. This must have major\_version==0 and raid\_disks != 0.

Then uninitialized devices can be added with ADD\_NEW\_DISK. The structure passed to ADD\_NEW\_DISK must specify the state of the device and its role in the array.

Once started with RUN\_ARRAY, uninitialized spares can be added with HOT\_ADD\_DISK.

## 52.7 MD devices in sysfs

md devices appear in sysfs (/sys) as regular block devices, e.g.:

```
/sys/block/md0
```

Each md device will contain a subdirectory called md which contains further md-specific information about the device.

All md devices contain:

**level** a text file indicating the raid level. e.g. raid0, raid1, raid5, linear, multipath, faulty. If no raid level has been set yet (array is still being assembled), the value will reflect whatever has been written to it, which may be a name like the above, or may be a number such as 0, 5, etc.

**raid\_disks** a text file with a simple number indicating the number of devices in a fully functional array. If this is not yet known, the file will be empty. If an array is being resized this will contain the new number of devices. Some raid levels allow this value to be set while the array is active. This will reconfigure the array. Otherwise it can only be set while assembling an array. A change to this attribute will not be permitted if it would reduce the size of the array. To reduce the number of drives in an e.g. raid5, the array size must first be reduced by setting the array\_size attribute.

**chunk\_size** This is the size in bytes for chunks and is only relevant to raid levels that involve striping (0,4,5,6,10). The address space of the array is conceptually divided into chunks and consecutive chunks are striped onto neighbouring devices. The size should be at least PAGE\_SIZE (4k) and should be a power of 2. This can only be set while assembling an array

**layout** The layout for the array for the particular level. This is simply a number that is interpreted differently by different levels. It can be written while assembling an array.

**array\_size** This can be used to artificially constrain the available space in the array to be less than is actually available on the combined devices. Writing a number (in Kilobytes) which is less than the available size will set the size. Any reconfiguration of the array (e.g. adding devices) will not cause the size to change. Writing the word `default` will cause the effective size of the array to be whatever size is actually available based on `level`, `chunk_size` and `component_size`.

This can be used to reduce the size of the array before reducing the number of devices in a raid4/5/6, or to support external metadata formats which mandate such clipping.

**reshape\_position** This is either none or a sector number within the devices of the array where reshape is up to. If this is set, the three attributes mentioned above (`raid_disks`, `chunk_size`, `layout`) can potentially have 2 values, an old and a new value. If these values differ, reading the attribute returns:

```
new (old)
```

and writing will effect the new value, leaving the old unchanged.

**component\_size** For arrays with data redundancy (i.e. not raid0, linear, faulty, multipath), all components must be the same size - or at least there must a size that they all provide space for. This is a key part of the geometry of the array. It is measured in sectors and can be read from here. Writing to this value may resize the array if the personality supports it (raid1, raid5, raid6), and if the component drives are large enough.

**metadata\_version** This indicates the format that is being used to record metadata about the array. It can be 0.90 (traditional format), 1.0, 1.1, 1.2 (newer format in varying locations) or none indicating that the kernel isn't managing metadata at all. Alternately it can be `external:` followed by a string which is set by user-space. This indicates that metadata is managed by a user-space program. Any device failure or other event that requires a metadata update will cause array activity to be suspended until the event is acknowledged.

**resync\_start** The point at which resync should start. If no resync is needed, this will be a very large number (or none since 2.6.30-rc1). At array creation it will default to 0, though starting the array as `clean` will set it much larger.

**new\_dev** This file can be written but not read. The value written should be a block device number as major:minor. e.g. 8:0 This will cause that device to be attached to the array, if it is available. It will then appear at `md/dev-XXX` (depending on the name of the device) and further configuration is then possible.

**safe\_mode\_delay** When an md array has seen no write requests for a certain period of time, it will be marked as `clean`. When another write request arrives, the array is marked as `dirty` before the write

commences. This is known as `safe_mode`. The certain period is controlled by this file which stores the period as a number of seconds. The default is 200msec (0.200). Writing a value of 0 disables `safemode`.

**array\_state** This file contains a single word which describes the current state of the array. In many cases, the state can be set by writing the word for the desired state, however some states cannot be explicitly set, and some transitions are not allowed.

Select/poll works on this file. All changes except between `Active_idle` and `active` (which can be frequent and are not very interesting) are notified. `active->active_idle` is reported if the metadata is externally managed.

**clear** No devices, no size, no level

Writing is equivalent to `STOP_ARRAY` ioctl

**inactive** May have some settings, but array is not active all IO results in error

When written, doesn't tear down array, but just stops it

**suspended (not supported yet)** All IO requests will block. The array can be reconfigured.

Writing this, if accepted, will block until array is quiescent

**readonly** no resync can happen. no superblocks get written.

Write requests fail

**read-auto** like `readonly`, but behaves like `clean` on a write request.

**clean** no pending writes, but otherwise active.

When written to inactive array, starts without resync

If a write request arrives then if metadata is known, mark dirty and switch to `active`. if not known, block and switch to `write-pending`

If written to an active array that has pending writes, then fails.

**active** fully active: IO and resync can be happening. When written to inactive array, starts with resync

**write-pending** `clean`, but writes are blocked waiting for `active` to be written.

**active-idle** like `active`, but no writes have been seen for a while (`safe_mode_delay`).

**bitmap/location** This indicates where the write-intent bitmap for the array is stored.

It can be one of `none`, `file` or `[+-]N`. `file` may later be extended to `file:/file/name` `[+-]N` means that many sectors from the start of the metadata.

This is replicated on all devices. For arrays with externally managed metadata, the offset is from the beginning of the device.

**bitmap/chunksize** The size, in bytes, of the chunk which will be represented by a single bit. For RAID456, it is a portion of an individual device. For RAID10, it is a portion of the array. For RAID1, it is both (they come to the same thing).

**bitmap/time\_base** The time, in seconds, between looking for bits in the bitmap to be cleared. In the current implementation, a bit will be cleared between 2 and 3 times `time_base` after all the covered blocks are known to be in-sync.

**bitmap/backlog** When write-mostly devices are active in a RAID1, write requests to those devices proceed in the background - the filesystem (or other user of the device) does not have to wait for them. `backlog` sets a limit on the number of concurrent background writes. If there are more than this, new writes will be synchronous.

**bitmap/metadata** This can be either `internal` or `external`.

**internal** is the default and means the metadata for the bitmap is stored in the first 256 bytes of the allocated space and is managed by the md module.

**external** means that bitmap metadata is managed externally to the kernel (i.e. by some userspace program)

**bitmap/can\_clear** This is either `true` or `false`. If `true`, then bits in the bitmap will be cleared when the corresponding blocks are thought to be in-sync. If `false`, bits will never be cleared. This is automatically set to `false` if a write happens on a degraded array, or if the array becomes degraded during a write. When metadata is managed externally, it should be set to `true` once the array becomes non-degraded, and this fact has been recorded in the metadata.

**consistency\_policy** This indicates how the array maintains consistency in case of unexpected shutdown. It can be:

**none** Array has no redundancy information, e.g. `raid0`, `linear`.

**resync** Full resync is performed and all redundancy is regenerated when the array is started after unclean shutdown.

**bitmap** Resync assisted by a write-intent bitmap.

**journal** For `raid4/5/6`, journal device is used to log transactions and replay after unclean shutdown.

**ppl** For `raid5` only, Partial Parity Log is used to close the write hole and eliminate resync.

The accepted values when writing to this file are `ppl` and `resync`, used to enable and disable PPL.

As component devices are added to an md array, they appear in the md directory as new directories named:

```
dev-XXX
```

where XXX is a name that the kernel knows for the device, e.g. hdb1. Each directory contains:

**block** a symlink to the block device in /sys/block, e.g.:

```
/sys/block/md0/md/dev-hdb1/block -> ../../../../../../block/hdb/hdb1
```

**super** A file containing an image of the superblock read from, or written to, that device.

**state** A file recording the current state of the device in the array which can be a comma separated list of:

**faulty** device has been kicked from active use due to a detected fault, or it has unacknowledged bad blocks

**in\_sync** device is a fully in-sync member of the array

**writemostly** device will only be subject to read requests if there are no other options.

This applies only to raid1 arrays.

**blocked** device has failed, and the failure hasn't been acknowledged yet by the metadata handler.

Writes that would write to this device if it were not faulty are blocked.

**spare** device is working, but not a full member.

This includes spares that are in the process of being recovered to

**write\_error** device has ever seen a write error.

**want\_replacement** device is (mostly) working but probably should be replaced, either due to errors or due to user request.

**replacement** device is a replacement for another active device with same raid\_disk.

This list may grow in future.

This can be written to.

Writing faulty simulates a failure on the device.

Writing remove removes the device from the array.

Writing writemostly sets the writemostly flag.

Writing -writemostly clears the writemostly flag.

Writing blocked sets the blocked flag.

Writing -blocked clears the blocked flags and allows writes to complete and possibly simulates an error.

Writing `in_sync` sets the `in_sync` flag.

Writing `write_error` sets `writeerrorseen` flag.

Writing `-write_error` clears `writeerrorseen` flag.

Writing `want_replacement` is allowed at any time except to a replacement device or a spare. It sets the flag.

Writing `-want_replacement` is allowed at any time. It clears the flag.

Writing `replacement` or `-replacement` is only allowed before starting the array. It sets or clears the flag.

This file responds to `select/poll`. Any change to `faulty` or `blocked` causes an event.

**errors** An approximate count of read errors that have been detected on this device but have not caused the device to be evicted from the array (either because they were corrected or because they happened while the array was read-only). When using version-1 metadata, this value persists across restarts of the array.

This value can be written while assembling an array thus providing an ongoing count for arrays with metadata managed by userspace.

**slot** This gives the role that the device has in the array. It will either be none if the device is not active in the array (i.e. is a spare or has failed) or an integer less than the `raid_disks` number for the array indicating which position it currently fills. This can only be set while assembling an array. A device for which this is set is assumed to be working.

**offset** This gives the location in the device (in sectors from the start) where data from the array will be stored. Any part of the device before this offset is not touched, unless it is used for storing metadata (Formats 1.1 and 1.2).

**size** The amount of the device, after the offset, that can be used for storage of data. This will normally be the same as the `component_size`. This can be written while assembling an array. If a value less than the current `component_size` is written, it will be rejected.

**recovery\_start** When the device is not `in_sync`, this records the number of sectors from the start of the device which are known to be correct. This is normally zero, but during a recovery operation it will steadily increase, and if the recovery is interrupted, restoring this value can cause recovery to avoid repeating the earlier blocks. With v1.x metadata, this value is saved and restored automatically.

This can be set whenever the device is not an active member of the array, either before the array is activated, or before the `slot` is set.

Setting this to none is equivalent to setting `in_sync`. Setting to any other value also clears the `in_sync` flag.

**bad\_blocks** This gives the list of all known bad blocks in the form of start address and length (in sectors respectively). If output is too

big to fit in a page, it will be truncated. Writing sector length to this file adds new acknowledged (i.e. recorded to disk safely) bad blocks.

**unacknowledged\_bad\_blocks** This gives the list of known-but-not-yet-saved-to-disk bad blocks in the same form of `bad_blocks`. If output is too big to fit in a page, it will be truncated. Writing to this file adds bad blocks without acknowledging them. This is largely for testing.

**ppl\_sector, ppl\_size** Location and size (in sectors) of the space used for Partial Parity Log on this device.

An active md device will also contain an entry for each active device in the array. These are named:

```
rdNN
```

where NN is the position in the array, starting from 0. So for a 3 drive array there will be `rd0`, `rd1`, `rd2`. These are symbolic links to the appropriate `dev-XXX` entry. Thus, for example:

```
cat /sys/block/md*/md/rd*/state
```

will show `in_sync` on every line.

Active md devices for levels that support data redundancy (1,4,5,6,10) also have

**sync\_action** a text file that can be used to monitor and control the rebuild process. It contains one word which can be one of:

**resync** redundancy is being recalculated after unclean shutdown or creation

**recover** a hot spare is being built to replace a failed/missing device

**idle** nothing is happening

**check** A full check of redundancy was requested and is happening. This reads all blocks and checks them. A repair may also happen for some raid levels.

**repair** A full check and repair is happening. This is similar to `resync`, but was requested by the user, and the write-intent bitmap is NOT used to optimise the process.

This file is writable, and each of the strings that could be read are meaningful for writing.

`idle` will stop an active `resync/recovery` etc. There is no guarantee that another `resync/recovery` may not be automatically started again, though some event will be needed to trigger this.

`resync` or `recovery` can be used to restart the corresponding operation if it was stopped with `idle`.

check and repair will start the appropriate process providing the current state is idle.

This file responds to select/poll. Any important change in the value triggers a poll event. Sometimes the value will briefly be recover if a recovery seems to be needed, but cannot be achieved. In that case, the transition to recover isn't notified, but the transition away is.

**degraded** This contains a count of the number of devices by which the arrays is degraded. So an optimal array will show 0. A single failed/missing drive will show 1, etc.

This file responds to select/poll, any increase or decrease in the count of missing devices will trigger an event.

**mismatch\_count** When performing check and repair, and possibly when performing resync, md will count the number of errors that are found. The count in `mismatch_cnt` is the number of sectors that were re-written, or (for check) would have been re-written. As most raid levels work in units of pages rather than sectors, this may be larger than the number of actual errors by a factor of the number of sectors in a page.

**bitmap\_set\_bits** If the array has a write-intent bitmap, then writing to this attribute can set bits in the bitmap, indicating that a resync would need to check the corresponding blocks. Either individual numbers or start-end pairs can be written. Multiple numbers can be separated by a space.

Note that the numbers are bit numbers, not block numbers. They should be scaled by the `bitmap_chunksize`.

**sync\_speed\_min, sync\_speed\_max** This are similar to `/proc/sys/dev/raid/speed_limit_{min,max}` however they only apply to the particular array.

If no value has been written to these, or if the word `system` is written, then the system-wide value is used. If a value, in kibibytes-per-second is written, then it is used.

When the files are read, they show the currently active value followed by `(local)` or `(system)` depending on whether it is a locally set or system-wide value.

**sync\_completed** This shows the number of sectors that have been completed of whatever the current `sync_action` is, followed by the number of sectors in total that could need to be processed. The two numbers are separated by a `/` thus effectively showing one value, a fraction of the process that is complete.

A select on this attribute will return when resync completes, when it reaches the current `sync_max` (below) and possibly at other times.

**sync\_speed** This shows the current actual speed, in K/sec, of the current `sync_action`. It is averaged over the last 30 seconds.

**suspend\_lo, suspend\_hi** The two values, given as numbers of sectors, indicate a range within the array where IO will be blocked. This is currently only supported for raid4/5/6.

**sync\_min, sync\_max** The two values, given as numbers of sectors, indicate a range within the array where check/repair will operate. Must be a multiple of `chunk_size`. When it reaches `sync_max` it will pause, rather than complete. You can use `select` or `poll` on `sync_completed` to wait for that number to reach `sync_max`. Then you can either increase `sync_max`, or can write `idle` to `sync_action`.

The value of `max` for `sync_max` effectively disables the limit. When a resync is active, the value can only ever be increased, never decreased. The value of `0` is the minimum for `sync_min`.

Each active md device may also have attributes specific to the personality module that manages it. These are specific to the implementation of the module and could change substantially if the implementation changes.

These currently include:

**stripe\_cache\_size (currently raid5 only)** number of entries in the stripe cache. This is writable, but there are upper and lower limits (32768, 17). Default is 256.

**strip\_cache\_active (currently raid5 only)** number of active entries in the stripe cache

**preread\_bypass\_threshold (currently raid5 only)** number of times a stripe requiring preread will be bypassed by a stripe that does not require preread. For fairness defaults to 1. Setting this to 0 disables bypass accounting and requires preread stripes to wait until all full-width stripe-writes are complete. Valid values are 0 to `stripe_cache_size`.

**journal\_mode (currently raid5 only)** The cache mode for raid5. raid5 could include an extra disk for caching. The mode can be “write-through” and “write-back”. The default is “write-through”.

**ppl\_write\_hint** NVMe stream ID to be set for each PPL write request.

## **MEDIA SUBSYSTEM ADMIN AND USER GUIDE**

This section contains usage information about media subsystem and its supported drivers.

Please see:

- **/userspace-api/media/index** for the userspace APIs used on media devices.
- **/driver-api/media/index** for driver development information and Kernel APIs used by media devices;

### **53.1 The media subsystem**

#### **53.1.1 Introduction**

The media subsystem consists on Linux support for several different types of devices:

- Audio and video grabbers;
- PC and Laptop Cameras;
- Complex cameras found on Embedded hardware;
- Analog and digital TV;
- HDMI Customer Electronics Control (CEC);
- Multi-touch input devices;
- Remote Controllers;
- Media encoders and decoders.

Due to the diversity of devices, the subsystem provides several different APIs:

- Remote Controller API;
- HDMI CEC API;
- Video4Linux API;
- Media controller API;
- Video4Linux Request API (experimental);
- Digital TV API (also known as DVB API).

### 53.1.2 Building support for a media device

The first step is to download the Kernel' s source code, either via a distribution-specific source file or via the Kernel' s main git tree<sup>1</sup>.

Please notice, however, that, if:

- you' re a braveheart and want to experiment with new stuff;
- if you want to report a bug;
- if you' re developing new patches

you should use the main media development tree master branch:

[https://git.linuxtv.org/media\\_tree.git/](https://git.linuxtv.org/media_tree.git/)

In this case, you may find some useful information at the [LinuxTv wiki pages](#):

[https://linuxtv.org/wiki/index.php/How\\_to\\_Obtain,\\_Build\\_and\\_Install\\_V4L-DVB\\_Device\\_Drivers](https://linuxtv.org/wiki/index.php/How_to_Obtain,_Build_and_Install_V4L-DVB_Device_Drivers)

### Configuring the Linux Kernel

You can access a menu of Kernel building options with:

```
$ make menuconfig
```

Then, select all desired options and exit it, saving the configuration.

The changed configuration will be at the `.config` file. It would look like:

```
...
# CONFIG_RC_CORE is not set
# CONFIG_CEC_CORE is not set
CONFIG_MEDIA_SUPPORT=m
CONFIG_MEDIA_SUPPORT_FILTER=y
...
```

The media subsystem is controlled by those menu configuration options:

```
Device Drivers --->
  <M> Remote Controller support --->
    [ ] HDMI CEC RC integration
    [ ] Enable CEC error injection support
    [*] HDMI CEC drivers --->
  <*> Multimedia support --->
```

The Remote Controller support option enables the core support for remote controllers<sup>2</sup>.

The HDMI CEC RC integration option enables integration of HDMI CEC with Linux, allowing to receive data via HDMI CEC as if it were produced by a remote controller directly connected to the machine.

---

<sup>1</sup> The upstream Linux Kernel development tree is located at <https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/>

<sup>2</sup> Remote Controller support should also be enabled if you want to use some TV card drivers that may depend on the remote controller core support.

The HDMI CEC drivers option allow selecting platform and USB drivers that receives and/or transmits CEC codes via HDMI interfaces<sup>3</sup>.

The last option (Multimedia support) enables support for cameras, audio/video grabbers and TV.

The media subsystem support can either be built together with the main Kernel or as a module. For most use cases, it is preferred to have it built as modules.

---

**Note:** Instead of using a menu, the Kernel provides a script with allows enabling configuration options directly. To enable media support and remote controller support using Kernel modules, you could use:

```
$ scripts/config -m RC_CORE
$ scripts/config -m MEDIA_SUPPORT
```

---

## Media dependencies

It should be noticed that enabling the above from a clean config is usually not enough. The media subsystem depends on several other Linux core support in order to work.

For example, most media devices use a serial communication bus in order to talk with some peripherals. Such bus is called I<sup>2</sup>C (Inter-Integrated Circuit). In order to be able to build support for such hardware, the I<sup>2</sup>C bus support should be enabled, either via menu or with:

```
./scripts/config -m I2C
```

Another example: the remote controller core requires support for input devices, with can be enabled with:

```
./scripts/config -m INPUT
```

Other core functionality may also be needed (like PCI and/or USB support), depending on the specific driver(s) you would like to enable.

---

<sup>3</sup> Please notice that the DRM subsystem also have drivers for GPUs that use the media HDMI CEC support.

Those GPU-specific drivers are selected via the Graphics support menu, under Device Drivers. When a GPU driver supports HDMI CEC, it will automatically enable the CEC core support at the media subsystem.

### Enabling Remote Controller Support

The remote controller menu allows selecting drivers for specific devices. It's menu looks like this:

```
--- Remote Controller support
<M>  Compile Remote Controller keymap modules
[*]  LIRC user interface
[*]   Support for eBPF programs attached to lirc devices
[*]  Remote controller decoders --->
[*]  Remote Controller devices --->
```

The Compile Remote Controller keymap modules option creates key maps for several popular remote controllers.

The LIRC user interface option adds enhanced functionality when using the lirc program, by enabling an API that allows userspace to receive raw data from remote controllers.

The Support for eBPF programs attached to lirc devices option allows the usage of special programs (called eBPF) that would allow applications to add extra remote controller decoding functionality to the Linux Kernel.

The Remote controller decoders option allows selecting the protocols that will be recognized by the Linux Kernel. Except if you want to disable some specific decoder, it is suggested to keep all sub-options enabled.

The Remote Controller devices allows you to select the drivers that would be needed to support your device.

The same configuration can also be set via the script/config script. So, for instance, in order to support the ITE remote controller driver (found on Intel NUCs and on some ASUS x86 desktops), you could do:

```
$ scripts/config -e INPUT
$ scripts/config -e ACPI
$ scripts/config -e MODULES
$ scripts/config -m RC_CORE
$ scripts/config -e RC_DEVICES
$ scripts/config -e RC_DECODERS
$ scripts/config -m IR_RC5_DECODER
$ scripts/config -m IR_ITE_CIR
```

### Enabling HDMI CEC Support

The HDMI CEC support is set automatically when a driver requires it. So, all you need to do is to enable support either for a graphics card that needs it or by one of the existing HDMI drivers.

The HDMI-specific drivers are available at the HDMI CEC drivers menu<sup>4</sup>:

---

<sup>4</sup> The above contents is just an example. The actual options for HDMI devices depends on the system's architecture and may vary on new Kernels.

```

--- HDMI CEC drivers
< > ChromeOS EC CEC driver
< > Amlogic Meson A0 CEC driver
< > Amlogic Meson G12A A0 CEC driver
< > Generic GPIO-based CEC driver
< > Samsung S5P CEC driver
< > STMicroelectronics STiH4xx HDMI CEC driver
< > STMicroelectronics STM32 HDMI CEC driver
< > Tegra HDMI CEC driver
< > SECO Boards HDMI CEC driver
[ ] SECO Boards IR RC5 support
< > Pulse Eight HDMI CEC
< > RainShadow Tech HDMI CEC

```

## Enabling Media Support

The Media menu has a lot more options than the remote controller menu. Once selected, you should see the following options:

```

--- Media support
[ ] Filter media drivers
[*] Autoselect ancillary drivers
  Media device types --->
  Media core support --->
  Video4Linux options --->
  Media controller options --->
  Digital TV options --->
  HDMI CEC options --->
  Media drivers --->
  Media ancillary drivers --->

```

Except if you know exactly what you're doing, or if you want to build a driver for a SoC platform, it is strongly recommended to keep the Autoselect ancillary drivers option turned on, as it will auto-select the needed I<sup>2</sup>C ancillary drivers.

There are now two ways to select media device drivers, as described below.

### Filter media drivers menu

This menu is meant to easy setup for PC and Laptop hardware. It works by letting the user to specify what kind of media drivers are desired, with those options:

```

[ ] Cameras and video grabbers
[ ] Analog TV
[ ] Digital TV
[ ] AM/FM radio receivers/transmitters
[ ] Software defined radio
[ ] Platform-specific devices
[ ] Test drivers

```

So, if you want to add support to a camera or video grabber only, select just the first option. Multiple options are allowed.

Once the options on this menu are selected, the building system will auto-select the needed core drivers in order to support the selected functionality.

---

**Note:** Most TV cards are hybrid: they support both Analog TV and Digital TV.

If you have an hybrid card, you may need to enable both Analog TV and Digital TV at the menu.

---

When using this option, the defaults for the the media support core functionality are usually good enough to provide the basic functionality for the driver. Yet, you could manually enable some desired extra (optional) functionality using the settings under each of the following Media support sub-menus:

```
Media core support --->
Video4Linux options --->
Media controller options --->
Digital TV options --->
HDMI CEC options --->
```

Once you select the desired filters, the drivers that matches the filtering criteria will be available at the Media support->Media drivers sub-menu.

### Media Core Support menu without filtering

If you disable the Filter media drivers menu, all drivers available for your system whose dependencies are met should be shown at the Media drivers menu.

Please notice, however, that you should first ensure that the Media Core Support menu has all the core functionality your drivers would need, as otherwise the corresponding device drivers won't be shown.

### Example

In order to enable modular support for one of the boards listed on this table, with modular media core modules, the .config file should contain those lines:

```
CONFIG_MODULES=y
CONFIG_USB=y
CONFIG_I2C=y
CONFIG_INPUT=y
CONFIG_RC_CORE=m
CONFIG_MEDIA_SUPPORT=m
CONFIG_MEDIA_SUPPORT_FILTER=y
CONFIG_MEDIA_ANALOG_TV_SUPPORT=y
CONFIG_MEDIA_DIGITAL_TV_SUPPORT=y
CONFIG_MEDIA_USB_SUPPORT=y
CONFIG_VIDEO_CX231XX=y
CONFIG_VIDEO_CX231XX_DVB=y
```

## Building and installing a new Kernel

Once the `.config` file has everything needed, all it takes to build is to run the `make` command:

```
$ make
```

And then install the new Kernel and its modules:

```
$ sudo make modules_install  
$ sudo make install
```

## Building just the new media drivers and core

Running a new development Kernel from the development tree is usually risky, because it may have experimental changes that may have bugs. So, there are some ways to build just the new drivers, using alternative trees.

There is the [Linux Kernel backports project](#), with contains newer drivers meant to be compiled against stable Kernels.

The LinuxTV developers, with are responsible for maintaining the media subsystem also maintains a backport tree, with just the media drivers daily updated from the newest kernel. Such tree is available at:

[https://git.linuxtv.org/media\\_build.git/](https://git.linuxtv.org/media_build.git/)

It should be noticed that, while it should be relatively safe to use the `media_build` tree for testing purposes, there are not warranties that it would work (or even build) on a random Kernel. This tree is maintained using a “best-efforts” principle, as time permits us to fix issues there.

If you notice anything wrong on it, feel free to submit patches at the Linux media subsystem’s mailing list: [media@vger.kernel.org](mailto:media@vger.kernel.org). Please add [PATCH media-build] at the e-mail’s subject if you submit a new patch for the `media-build`.

Before using it, you should run:

```
$ ./build
```

---

### Note:

- 1) you may need to run it twice if the `media-build` tree gets updated;
  - 2) you may need to do a `make distclean` if you had built it in the past for a different Kernel version than the one you’re currently using;
  - 3) by default, it will use the same config options for media as the ones defined on the Kernel you’re running.
- 

In order to select different drivers or different config options, use:

```
$ make menuconfig
```

Then, you can build and install the new drivers:

```
$ make && sudo make install
```

This will override the previous media drivers that your Kernel were using.

### 53.1.3 Infrared remote control support in video4linux drivers

Authors: Gerd Hoffmann, Mauro Carvalho Chehab

#### Basics

Most analog and digital TV boards support remote controllers. Several of them have a microprocessor that receives the IR carriers, convert into pulse/space sequences and then to scan codes, returning such codes to userspace ( “scancode mode” ). Other boards return just the pulse/space sequences ( “raw mode” ).

The support for remote controller in scancode mode is provided by the standard Linux input layer. The support for raw mode is provided via LIRC.

In order to check the support and test it, it is suggested to download the [v4l-utils](#). It provides two tools to handle remote controllers:

- `ir-keytable`: provides a way to query the remote controller, list the protocols it supports, enable in-kernel support for IR decoder or switch the protocol and to test the reception of scan codes;
- `ir-ctl`: provide tools to handle remote controllers that support raw mode via LIRC interface.

Usually, the remote controller module is auto-loaded when the TV card is detected. However, for a few devices, you need to manually load the `ir-kbd-i2c` module.

#### How it works

The modules register the remote as keyboard within the linux input layer, i.e. you'll see the keys of the remote as normal key strokes (if `CONFIG_INPUT_KEYBOARD` is enabled).

Using the event devices (`CONFIG_INPUT_EVDEV`) it is possible for applications to access the remote via `/dev/input/event<n>` devices. The `udev/systemd` will automatically create the devices. If you install the [v4l-utils](#), it may also automatically load a different keytable than the default one. Please see [v4l-utils ir-keytable.1](#) man page for details.

The `ir-keytable` tool is nice for trouble shooting, i.e. to check whenever the input device is really present, which of the devices it is, check whenever pressing keys on the remote actually generates events and the like. You can also use any other input utility that changes the keymaps, like the `input kbd` utility.

## Using with lircd

The latest versions of the lircd daemon supports reading events from the linux input layer (via event device). It also supports receiving IR codes in lirc mode.

## Using without lircd

Xorg recognizes several IR keycodes that have its numerical value lower than 247. With the advent of Wayland, the input driver got updated too, and should now accept all keycodes. Yet, you may want to just reassign the keycodes to something that your favorite media application likes.

This can be done by setting `v4l-utils` to load your own keytable in runtime. Please read `ir-keytable.1` man page for details.

## 53.1.4 Digital TV

### Using the Digital TV Framework

#### Introduction

One significant difference between Digital TV and Analogue TV that the unwary (like myself) should consider is that, although the component structure of DVB-T cards are substantially similar to Analogue TV cards, they function in substantially different ways.

The purpose of an Analogue TV is to receive and display an Analogue Television signal. An Analogue TV signal (otherwise known as composite video) is an analogue encoding of a sequence of image frames (25 frames per second in Europe) rasterised using an interlacing technique. Interlacing takes two fields to represent one frame. Therefore, an Analogue TV card for a PC has the following purpose:

- Tune the receiver to receive a broadcast signal
- demodulate the broadcast signal
- demultiplex the analogue video signal and analogue audio signal.

---

**Note:** some countries employ a digital audio signal embedded within the modulated composite analogue signal - using NICAM signaling.)

---

- digitize the analogue video signal and make the resulting datastream available to the data bus.

The digital datastream from an Analogue TV card is generated by circuitry on the card and is often presented uncompressed. For a PAL TV signal encoded at a resolution of 768x576 24-bit color pixels over 25 frames per second - a fair amount of data is generated and must be processed by the PC before it can be displayed on the video monitor screen. Some Analogue TV cards for PCs have onboard MPEG2 encoders which permit the raw digital data stream to be presented to the PC in an encoded and compressed form - similar to the form that is used in Digital TV.

The purpose of a simple budget digital TV card (DVB-T,C or S) is to simply:

- Tune the received to receive a broadcast signal. \* Extract the encoded digital datastream from the broadcast signal.
- Make the encoded digital datastream (MPEG2) available to the data bus.

The significant difference between the two is that the tuner on the analogue TV card spits out an Analogue signal, whereas the tuner on the digital TV card spits out a compressed encoded digital datastream. As the signal is already digitised, it is trivial to pass this datastream to the PC databus with minimal additional processing and then extract the digital video and audio datastreams passing them to the appropriate software or hardware for decoding and viewing.

### Getting the card going

The Device Driver API for DVB under Linux will the following device nodes via the devfs filesystem:

- /dev/dvb/adapter0/demux0
- /dev/dvb/adapter0/dvr0
- /dev/dvb/adapter0/frontend0

The /dev/dvb/adapter0/dvr0 device node is used to read the MPEG2 Data Stream and the /dev/dvb/adapter0/frontend0 device node is used to tune the frontend tuner module. The /dev/dvb/adapter0/demux0 is used to control what programs will be received.

Depending on the card's feature set, the Device Driver API could also expose other device nodes:

- /dev/dvb/adapter0/ca0
- /dev/dvb/adapter0/audio0
- /dev/dvb/adapter0/net0
- /dev/dvb/adapter0/osd0
- /dev/dvb/adapter0/video0

The /dev/dvb/adapter0/ca0 is used to decode encrypted channels. The other device nodes are found only on devices that use the av7110 driver, with is now obsolete, together with the extra API whose such devices use.

### Receiving a digital TV channel

This section attempts to explain how it works and how this affects the configuration of a Digital TV card.

On this example, we' re considering tuning into DVB-T channels in Australia, at the Melbourne region.

The frequencies broadcast by Mount Dandenong transmitters are, currently:

Table 1. Transponder Frequencies Mount Dandenong, Vic, Aus.

Broadcaster	Frequency
Seven	177.500 Mhz
SBS	184.500 Mhz
Nine	191.625 Mhz
Ten	219.500 Mhz
ABC	226.500 Mhz
Channel 31	557.625 Mhz

The digital TV Scan utilities (like `dvbv5-scan`) have use a set of compiled-in defaults for various countries and regions. Those are currently provided as a separate package, called `dtv-scan-tables`. It' s git tree is located at [LinuxTV.org](https://git.linuxtv.org/dtv-scan-tables.git/):

<https://git.linuxtv.org/dtv-scan-tables.git/>

If none of the tables there suit, you can specify a data file on the command line which contains the transponder frequencies. Here is a sample file for the above channel transponders, in the old “channel” format:

```
# Data file for DVB scan program
#
# C Frequency SymbolRate FEC QAM
# S Frequency Polarisation SymbolRate FEC
# T Frequency Bandwidth FEC FEC2 QAM Mode Guard Hier

T 177500000 7MHz AUTO AUTO QAM64 8k 1/16 NONE
T 184500000 7MHz AUTO AUTO QAM64 8k 1/8 NONE
T 191625000 7MHz AUTO AUTO QAM64 8k 1/16 NONE
T 219500000 7MHz AUTO AUTO QAM64 8k 1/16 NONE
T 226500000 7MHz AUTO AUTO QAM64 8k 1/16 NONE
T 557625000 7MHz AUTO AUTO QPSK 8k 1/16 NONE
```

Nowadays, we prefer to use a newer format, with is more verbose and easier to understand. With the new format, the “Seven” channel transponder’ s data is represented by:

```
[Seven]
DELIVERY_SYSTEM = DVBT
FREQUENCY = 177500000
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = AUTO
CODE_RATE_LP = AUTO
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
GUARD_INTERVAL = 1/16
HIERARCHY = NONE
INVERSION = AUTO
```

For an updated version of the complete table, please see:

<https://git.linuxtv.org/dtv-scan-tables.git/tree/dvb-t/au-Melbourne>

When the Digital TV scanning utility runs, it will output a file containing the information for all the audio and video programs that exists into each channel’ s transponders which the card’ s frontend can lock onto. (i.e. any whose signal is strong enough at your antenna).

Here' s the output of the dvbv5 tools from a channel scan took from Melbourne:

```
[ABC HDTV]
SERVICE_ID = 560
VIDEO_PID = 2307
AUDIO_PID = 0
DELIVERY_SYSTEM = DVBT
FREQUENCY = 226500000
INVERSION = OFF
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = 3/4
CODE_RATE_LP = 3/4
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
GUARD_INTERVAL = 1/16
HIERARCHY = NONE

[ABC TV Melbourne]
SERVICE_ID = 561
VIDEO_PID = 512
AUDIO_PID = 650
DELIVERY_SYSTEM = DVBT
FREQUENCY = 226500000
INVERSION = OFF
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = 3/4
CODE_RATE_LP = 3/4
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
GUARD_INTERVAL = 1/16
HIERARCHY = NONE

[ABC TV 2]
SERVICE_ID = 562
VIDEO_PID = 512
AUDIO_PID = 650
DELIVERY_SYSTEM = DVBT
FREQUENCY = 226500000
INVERSION = OFF
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = 3/4
CODE_RATE_LP = 3/4
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
GUARD_INTERVAL = 1/16
HIERARCHY = NONE

[ABC TV 3]
SERVICE_ID = 563
VIDEO_PID = 512
AUDIO_PID = 650
DELIVERY_SYSTEM = DVBT
FREQUENCY = 226500000
INVERSION = OFF
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = 3/4
CODE_RATE_LP = 3/4
```

(continues on next page)

(continued from previous page)

```
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
GUARD_INTERVAL = 1/16
HIERARCHY = NONE
```

[ABC TV 4]

```
SERVICE_ID = 564
VIDEO_PID = 512
AUDIO_PID = 650
DELIVERY_SYSTEM = DVBT
FREQUENCY = 226500000
INVERSION = OFF
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = 3/4
CODE_RATE_LP = 3/4
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
GUARD_INTERVAL = 1/16
HIERARCHY = NONE
```

[ABC DiG Radio]

```
SERVICE_ID = 566
VIDEO_PID = 0
AUDIO_PID = 2311
DELIVERY_SYSTEM = DVBT
FREQUENCY = 226500000
INVERSION = OFF
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = 3/4
CODE_RATE_LP = 3/4
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
GUARD_INTERVAL = 1/16
HIERARCHY = NONE
```

[TEN Digital]

```
SERVICE_ID = 1585
VIDEO_PID = 512
AUDIO_PID = 650
DELIVERY_SYSTEM = DVBT
FREQUENCY = 219500000
INVERSION = OFF
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = 3/4
CODE_RATE_LP = 1/2
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
GUARD_INTERVAL = 1/16
HIERARCHY = NONE
```

[TEN Digital 1]

```
SERVICE_ID = 1586
VIDEO_PID = 512
AUDIO_PID = 650
DELIVERY_SYSTEM = DVBT
FREQUENCY = 219500000
```

(continues on next page)

(continued from previous page)

```
INVERSION = OFF
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = 3/4
CODE_RATE_LP = 1/2
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
GUARD_INTERVAL = 1/16
HIERARCHY = NONE
```

[TEN Digital 2]

```
SERVICE_ID = 1587
VIDEO_PID = 512
AUDIO_PID = 650
DELIVERY_SYSTEM = DVBT
FREQUENCY = 219500000
INVERSION = OFF
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = 3/4
CODE_RATE_LP = 1/2
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
GUARD_INTERVAL = 1/16
HIERARCHY = NONE
```

[TEN Digital 3]

```
SERVICE_ID = 1588
VIDEO_PID = 512
AUDIO_PID = 650
DELIVERY_SYSTEM = DVBT
FREQUENCY = 219500000
INVERSION = OFF
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = 3/4
CODE_RATE_LP = 1/2
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
GUARD_INTERVAL = 1/16
HIERARCHY = NONE
```

[TEN Digital]

```
SERVICE_ID = 1589
VIDEO_PID = 512
AUDIO_PID = 650
DELIVERY_SYSTEM = DVBT
FREQUENCY = 219500000
INVERSION = OFF
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = 3/4
CODE_RATE_LP = 1/2
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
GUARD_INTERVAL = 1/16
HIERARCHY = NONE
```

[TEN Digital 4]

```
SERVICE_ID = 1590
```

(continues on next page)

(continued from previous page)

```
VIDEO_PID = 512
AUDIO_PID = 650
DELIVERY_SYSTEM = DVBT
FREQUENCY = 219500000
INVERSION = OFF
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = 3/4
CODE_RATE_LP = 1/2
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
GUARD_INTERVAL = 1/16
HIERARCHY = NONE
```

[TEN Digital]

```
SERVICE_ID = 1591
VIDEO_PID = 512
AUDIO_PID = 650
DELIVERY_SYSTEM = DVBT
FREQUENCY = 219500000
INVERSION = OFF
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = 3/4
CODE_RATE_LP = 1/2
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
GUARD_INTERVAL = 1/16
HIERARCHY = NONE
```

[TEN HD]

```
SERVICE_ID = 1592
VIDEO_PID = 514
AUDIO_PID = 0
DELIVERY_SYSTEM = DVBT
FREQUENCY = 219500000
INVERSION = OFF
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = 3/4
CODE_RATE_LP = 1/2
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
GUARD_INTERVAL = 1/16
HIERARCHY = NONE
```

[TEN Digital]

```
SERVICE_ID = 1593
VIDEO_PID = 512
AUDIO_PID = 650
DELIVERY_SYSTEM = DVBT
FREQUENCY = 219500000
INVERSION = OFF
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = 3/4
CODE_RATE_LP = 1/2
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
GUARD_INTERVAL = 1/16
```

(continues on next page)

(continued from previous page)

HIERARCHY = NONE

[Nine Digital]

SERVICE\_ID = 1072  
VIDEO\_PID = 513  
AUDIO\_PID = 660  
DELIVERY\_SYSTEM = DVBT  
FREQUENCY = 191625000  
INVERSION = OFF  
BANDWIDTH\_HZ = 7000000  
CODE\_RATE\_HP = 3/4  
CODE\_RATE\_LP = 1/2  
MODULATION = QAM/64  
TRANSMISSION\_MODE = 8K  
GUARD\_INTERVAL = 1/16  
HIERARCHY = NONE

[Nine Digital HD]

SERVICE\_ID = 1073  
VIDEO\_PID = 512  
AUDIO\_PID = 0  
DELIVERY\_SYSTEM = DVBT  
FREQUENCY = 191625000  
INVERSION = OFF  
BANDWIDTH\_HZ = 7000000  
CODE\_RATE\_HP = 3/4  
CODE\_RATE\_LP = 1/2  
MODULATION = QAM/64  
TRANSMISSION\_MODE = 8K  
GUARD\_INTERVAL = 1/16  
HIERARCHY = NONE

[Nine Guide]

SERVICE\_ID = 1074  
VIDEO\_PID = 514  
AUDIO\_PID = 670  
DELIVERY\_SYSTEM = DVBT  
FREQUENCY = 191625000  
INVERSION = OFF  
BANDWIDTH\_HZ = 7000000  
CODE\_RATE\_HP = 3/4  
CODE\_RATE\_LP = 1/2  
MODULATION = QAM/64  
TRANSMISSION\_MODE = 8K  
GUARD\_INTERVAL = 1/16  
HIERARCHY = NONE

[7 Digital]

SERVICE\_ID = 1328  
VIDEO\_PID = 769  
AUDIO\_PID = 770  
DELIVERY\_SYSTEM = DVBT  
FREQUENCY = 177500000  
INVERSION = OFF  
BANDWIDTH\_HZ = 7000000  
CODE\_RATE\_HP = 2/3

(continues on next page)

(continued from previous page)

```
CODE_RATE_LP = 2/3
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
GUARD_INTERVAL = 1/8
HIERARCHY = NONE
```

## [7 Digital 1]

```
SERVICE_ID = 1329
VIDEO_PID = 769
AUDIO_PID = 770
DELIVERY_SYSTEM = DVBT
FREQUENCY = 177500000
INVERSION = OFF
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = 2/3
CODE_RATE_LP = 2/3
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
GUARD_INTERVAL = 1/8
HIERARCHY = NONE
```

## [7 Digital 2]

```
SERVICE_ID = 1330
VIDEO_PID = 769
AUDIO_PID = 770
DELIVERY_SYSTEM = DVBT
FREQUENCY = 177500000
INVERSION = OFF
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = 2/3
CODE_RATE_LP = 2/3
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
GUARD_INTERVAL = 1/8
HIERARCHY = NONE
```

## [7 Digital 3]

```
SERVICE_ID = 1331
VIDEO_PID = 769
AUDIO_PID = 770
DELIVERY_SYSTEM = DVBT
FREQUENCY = 177500000
INVERSION = OFF
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = 2/3
CODE_RATE_LP = 2/3
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
GUARD_INTERVAL = 1/8
HIERARCHY = NONE
```

## [7 HD Digital]

```
SERVICE_ID = 1332
VIDEO_PID = 833
AUDIO_PID = 834
DELIVERY_SYSTEM = DVBT
```

(continues on next page)

(continued from previous page)

```
FREQUENCY = 177500000  
INVERSION = OFF  
BANDWIDTH_HZ = 7000000  
CODE_RATE_HP = 2/3  
CODE_RATE_LP = 2/3  
MODULATION = QAM/64  
TRANSMISSION_MODE = 8K  
GUARD_INTERVAL = 1/8  
HIERARCHY = NONE
```

[7 Program Guide]

```
SERVICE_ID = 1334  
VIDEO_PID = 865  
AUDIO_PID = 866  
DELIVERY_SYSTEM = DVBT  
FREQUENCY = 177500000  
INVERSION = OFF  
BANDWIDTH_HZ = 7000000  
CODE_RATE_HP = 2/3  
CODE_RATE_LP = 2/3  
MODULATION = QAM/64  
TRANSMISSION_MODE = 8K  
GUARD_INTERVAL = 1/8  
HIERARCHY = NONE
```

[SBS HD]

```
SERVICE_ID = 784  
VIDEO_PID = 102  
AUDIO_PID = 103  
DELIVERY_SYSTEM = DVBT  
FREQUENCY = 536500000  
INVERSION = OFF  
BANDWIDTH_HZ = 7000000  
CODE_RATE_HP = 2/3  
CODE_RATE_LP = 2/3  
MODULATION = QAM/64  
TRANSMISSION_MODE = 8K  
GUARD_INTERVAL = 1/8  
HIERARCHY = NONE
```

[SBS DIGITAL 1]

```
SERVICE_ID = 785  
VIDEO_PID = 161  
AUDIO_PID = 81  
DELIVERY_SYSTEM = DVBT  
FREQUENCY = 536500000  
INVERSION = OFF  
BANDWIDTH_HZ = 7000000  
CODE_RATE_HP = 2/3  
CODE_RATE_LP = 2/3  
MODULATION = QAM/64  
TRANSMISSION_MODE = 8K  
GUARD_INTERVAL = 1/8  
HIERARCHY = NONE
```

[SBS DIGITAL 2]

(continues on next page)

(continued from previous page)

```
SERVICE_ID = 786
VIDEO_PID = 162
AUDIO_PID = 83
DELIVERY_SYSTEM = DVBT
FREQUENCY = 536500000
INVERSION = OFF
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = 2/3
CODE_RATE_LP = 2/3
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
GUARD_INTERVAL = 1/8
HIERARCHY = NONE
```

[SBS EPG]

```
SERVICE_ID = 787
VIDEO_PID = 163
AUDIO_PID = 85
DELIVERY_SYSTEM = DVBT
FREQUENCY = 536500000
INVERSION = OFF
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = 2/3
CODE_RATE_LP = 2/3
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
GUARD_INTERVAL = 1/8
HIERARCHY = NONE
```

[SBS RADIO 1]

```
SERVICE_ID = 798
VIDEO_PID = 0
AUDIO_PID = 201
DELIVERY_SYSTEM = DVBT
FREQUENCY = 536500000
INVERSION = OFF
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = 2/3
CODE_RATE_LP = 2/3
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
GUARD_INTERVAL = 1/8
HIERARCHY = NONE
```

[SBS RADIO 2]

```
SERVICE_ID = 799
VIDEO_PID = 0
AUDIO_PID = 202
DELIVERY_SYSTEM = DVBT
FREQUENCY = 536500000
INVERSION = OFF
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = 2/3
CODE_RATE_LP = 2/3
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
```

(continues on next page)

(continued from previous page)

```
GUARD_INTERVAL = 1/8
HIERARCHY = NONE
```

### Digital TV Conditional Access Interface

---

**Note:** This documentation is outdated.

---

This document describes the usage of the high level CI API as in accordance to the Linux DVB API. This is not a documentation for the, existing low level CI API.

**Note:** For the Twinhan/Twinhan clones, the `dst_ca` module handles the CI hardware handling. This module is loaded automatically if a CI (Common Interface, that holds the CAM (Conditional Access Module) is detected.

---

#### **ca\_zap**

A userspace application, like `ca_zap` is required to handle encrypted MPEG-TS streams.

The `ca_zap` userland application is in charge of sending the descrambling related information to the Conditional Access Module (CAM).

This application requires the following to function properly as of now.

- a) Tune to a valid channel, with `szap`.  
eg: `$ szap -c channels.conf -r "TMC" -x`
- b) a `channels.conf` containing a valid PMT PID  
eg: `TMC:11996:h:0:27500:278:512:650:321`  
here 278 is a valid PMT PID. the rest of the values are the same ones that `szap` uses.
- c) after running a `szap`, you have to run `ca_zap`, for the descrambler to function,  
eg: `$ ca_zap channels.conf "TMC"`
- d) Hopefully enjoy your favourite subscribed channel as you do with a FTA card.

**Note:** Currently `ca_zap`, and `dst_test`, both are meant for demonstration purposes only, they can become full fledged applications if necessary.

---

## Cards that fall in this category

At present the cards that fall in this category are the Twinhan and its clones, these cards are available as VVMER, Tomato, Hercules, Orange and so on.

## CI modules that are supported

The CI module support is largely dependent upon the firmware on the cards. Some cards do support almost all of the available CI modules. There is nothing much that can be done in order to make additional CI modules working with these cards.

Modules that have been tested by this driver at present are

- (1) Irdeto 1 and 2 from SCM
- (2) Viaccess from SCM
- (3) Dragoncam

## FAQ

---

### Note:

1. With Digital TV, a single physical channel may have different contents inside it. The specs call each one as a service. This is what a TV user would call "channel". So, in order to avoid confusion, we're calling transponders as the physical channel on this FAQ, and services for the logical channel.
2. The LinuxTV community maintains some Wiki pages which contain a lot of information related to the media subsystem. If you don't find an answer for your needs here, it is likely that you'll be able to get something useful there. It is hosted at:

<https://www.linuxtv.org/wiki/>

---

Some very frequently asked questions about Linux Digital TV support

1. The signal seems to die a few seconds after tuning.

It's not a bug, it's a feature. Because the frontends have significant power requirements (and hence get very hot), they are powered down if they are unused (i.e. if the frontend device is closed). The `dvb-core` module parameter `dvb_shutdown_timeout` allows you to change the timeout (default 5 seconds). Setting the timeout to 0 disables the timeout feature.

2. How can I watch TV?

Together with the Linux Kernel, the Digital TV developers support some simple utilities which are mainly intended for testing and to demonstrate how the DVB API works. This is called DVB v5 tools and are grouped together with the `v4l-utils` git repository:

<https://git.linuxtv.org/v4l-utils.git/>

You can find more information at the LinuxTV wiki:

[https://www.linuxtv.org/wiki/index.php/DVBv5\\_Tools](https://www.linuxtv.org/wiki/index.php/DVBv5_Tools)

The first step is to get a list of services that are transmitted.

This is done by using several existing tools. You can use for example the `dvbv5-scan` tool. You can find more information about it at:

<https://www.linuxtv.org/wiki/index.php/Dvbv5-scan>

There are some other applications like `w_scan`<sup>1</sup> that do a blind scan, trying hard to find all possible channels, but those consumes a large amount of time to run.

Also, some applications like `kaffeine` have their own code to scan for services. So, you don't need to use an external application to obtain such list.

Most of such tools need a file containing a list of channel transponders available on your area. So, LinuxTV developers maintain tables of Digital TV channel transponders, receiving patches from the community to keep them updated.

This list is hosted at:

<https://git.linuxtv.org/dtv-scan-tables.git>

And packaged on several distributions.

`Kaffeine` has some blind scan support for some terrestrial standards. It also relies on DTV scan tables, although it contains a copy of it internally (and, if requested by the user, it will download newer versions of it).

If you are lucky you can just use one of the supplied channel transponders. If not, you may need to seek for such info at the Internet and create a new file. There are several sites with contains physical channel lists. For cable and satellite, usually knowing how to tune into a single channel is enough for the scanning tool to identify the other channels. On some places, this could also work for terrestrial transmissions.

Once you have a transponders list, you need to generate a services list with a tool like `dvbv5-scan`.

Almost all modern Digital TV cards don't have built-in hardware MPEG-decoders. So, it is up to the application to get a MPEG-TS stream provided by the board, split it into audio, video and other data and decode.

### 3. Which Digital TV applications exist?

Several media player applications are capable of tuning into digital TV channels, including `Kaffeine`, `Vlc`, `mplayer` and `MythTV`.

`Kaffeine` aims to be very user-friendly, and it is maintained by one of the Kernel driver developers.

---

<sup>1</sup> [https://www.linuxtv.org/wiki/index.php/W\\_scan](https://www.linuxtv.org/wiki/index.php/W_scan)

A comprehensive list of those and other apps can be found at:

[https://www.linuxtv.org/wiki/index.php/TV\\_Related\\_Software](https://www.linuxtv.org/wiki/index.php/TV_Related_Software)

Some of the most popular ones are linked below:

<https://kde.org/applications/multimedia/org.kde.kaffeine>

KDE media player, focused on Digital TV support

[https://www.linuxtv.org/vdrwiki/index.php/Main\\_Page](https://www.linuxtv.org/vdrwiki/index.php/Main_Page) Klaus Schmidinger's Video Disk Recorder

<https://linuxtv.org/downloads> and <https://git.linuxtv.org/>

Digital TV and other media-related applications and Kernel drivers. The v4l-utils package there contains several swiss knife tools for using with Digital TV.

<http://sourceforge.net/projects/dvbtools/> Dave Chapman's dvbtools package, including dvbstream and dvbtune

<http://www.dbox2.info/> LinuxDVB on the dBox2

<http://www.tuxbox.org/> the TuxBox CVS many interesting DVB applications and the dBox2 DVB source

<http://www.nenie.org/misc/mpsys/> MPSYS: a MPEG2 system library and tools

<https://www.videolan.org/vlc/index.pt.html> Vlc

<http://mplayerhq.hu/> MPlayer

<http://xine.sourceforge.net/> and <http://xinehq.de/> Xine

<http://www.mythtv.org/> MythTV - analog TV and digital TV PVR

<http://dvbsnoop.sourceforge.net/> DVB sniffer program to monitor, analyze, debug, dump or view dvb/mpeg/dsm-cc/mhp stream information (TS, PES, SECTION)

#### 4. Can't get a signal tuned correctly

That could be due to a lot of problems. On my personal experience, usually TV cards need stronger signals than TV sets, and are more sensitive to noise. So, perhaps you just need a better antenna or cabling. Yet, it could also be some hardware or driver issue.

For example, if you are using a Technotrend/Hauppauge DVB-C card without analog module, you might have to use module parameter `adac=-1` (`dvb-ttpci.o`).

Please see the FAQ page at [linuxtv.org](http://linuxtv.org), as it could contain some valuable information:

[https://www.linuxtv.org/wiki/index.php/FAQ\\_%26\\_Troubleshooting](https://www.linuxtv.org/wiki/index.php/FAQ_%26_Troubleshooting)

If that doesn't work, check at the linux-media ML archives, to see if someone else had a similar problem with your hardware and/or digital TV service provider:

<https://lore.kernel.org/linux-media/>

If none of this works, you can try sending an e-mail to the linux-media ML and see if someone else could shed some light. The e-mail is linux-media AT vger.kernel.org.

### 5. The dvb\_net device doesn't give me any packets at all

Run `tcpdump` on the `dvb0_0` interface. This sets the interface into promiscuous mode so it accepts any packets from the PID you have configured with the `dvbnet` utility. Check if there are any packets with the IP addr and MAC addr you have configured with `ifconfig` or with `ip addr`.

If `tcpdump` doesn't give you any output, check the statistics which `ifconfig` or `netstat -ni` outputs. (Note: If the MAC address is wrong, `dvb_net` won't get any input; thus you have to run `tcpdump` before checking the statistics.) If there are no packets at all then maybe the PID is wrong. If there are error packets, then either the PID is wrong or the stream does not conform to the MPE standard (EN 301 192, <http://www.etsi.org/>). You can use e.g. `dvbsnoop` for debugging.

### 6. The dvb\_net device doesn't give me any multicast packets

Check your routes if they include the multicast address range. Additionally make sure that "source validation by reversed path lookup" is disabled:

```
$ "echo 0 > /proc/sys/net/ipv4/conf/dvb0/rp_filter"
```

### 7. What are all those modules that need to be loaded?

In order to make it more flexible and support different hardware combinations, the media subsystem is written on a modular way.

So, besides the Digital TV hardware module for the main chipset, it also needs to load a frontend driver, plus the Digital TV core. If the board also has remote controller, it will also need the remote controller core and the remote controller tables. The same happens if the board has support for analog TV: the core support for `video4linux` need to be loaded.

The actual module names are Linux-kernel version specific, as, from time to time, things change, in order to make the media support more flexible.

## References

The main development site and GIT repository for Digital TV drivers is <https://linuxtv.org>.

The DVB mailing list linux-dvb is hosted at vger. Please see <http://vger.kernel.org/vger-lists.html#linux-media> for details.

There are also some other old lists hosted at: <https://linuxtv.org/lists.php>. If you're interested on that for historic reasons, please check the archive at <https://linuxtv.org/pipermail/linux-dvb/>.

The media subsystem Wiki is hosted at <https://linuxtv.org/wiki/>. There, you'll find lots of information, from both development and usage of media boards. Please check it before asking newbie questions on the mailing list or IRC channels.

The API documentation is documented at the Kernel tree. You can find it in both html and pdf formats, together with other useful documentation at:

- <https://linuxtv.org/docs.php>.

You may also find useful material at <https://linuxtv.org/downloads/>.

In order to get the needed firmware for some drivers to work, there's a script at the kernel tree, at `scripts/get_dvb_firmware`.

### 53.1.5 Cards List

The media subsystem provide support for lots of PCI and USB drivers, plus platform-specific drivers. It also contains several ancillary I<sup>2</sup>C drivers.

The platform-specific drivers are usually present on embedded systems, or are supported by the main board. Usually, setting them is done via OpenFirmware or ACPI.

The PCI and USB drivers, however, are independent of the system's board, and may be added/removed by the user.

You may also take a look at [https://linuxtv.org/wiki/index.php/Hardware\\_Device\\_Information](https://linuxtv.org/wiki/index.php/Hardware_Device_Information) for more details about supported cards.

## USB drivers

The USB boards are identified by an identification called USB ID.

The `lsusb` command allows identifying the USB IDs:

```
$ lsusb
...
Bus 001 Device 015: ID 046d:082d Logitech, Inc. HD Pro Webcam C920
Bus 001 Device 074: ID 2040:b131 Hauppauge
Bus 001 Device 075: ID 2013:024f PCTV Systems nanoStick T2 290e
...
```

Newer camera devices use a standard way to expose themselves as such, via USB Video Class. Those cameras are automatically supported by the `uvc-driver`.

Older cameras and TV USB devices uses USB Vendor Classes: each vendor defines its own way to access the device. This section contains card lists for such vendor-class devices.

While this is not as common as on PCI, sometimes the same USB ID is used by different products. So, several media drivers allow passing a `card=` parameter, in order to setup a card number that would match the correct settings for an specific product type.

The current supported USB cards (not including staging drivers) are listed below<sup>1</sup>.

	Driver	Name
airspy	AirSpy	
au0828	Auvitek	AU0828
b2c2-flexcop-usb	Technisat/B2C2	Air/Sky/Cable2PC USB
cpia2	CPiA2	Video For Linux
cx231xx	Conexant	cx231xx USB video capture
dvb-as102	Abilis	AS102 DVB receiver
dvb-ttusb-budget	Technotrend/Hauppauge	Nova - USB devices
dvb-usb-a800	AVerMedia	AverTV DVB-T USB 2.0 (A800)
dvb-usb-af9005	Afatech	AF9005 DVB-T USB1.1
dvb-usb-af9015	Afatech	AF9015 DVB-T USB2.0
dvb-usb-af9035	Afatech	AF9035 DVB-T USB2.0
dvb-usb-anysee	Anysee	DVB-T/C USB2.0
dvb-usb-au6610	Alcor Micro	AU6610 USB2.0
dvb-usb-az6007	AzureWave	6007 and clones DVB-T/C USB2.0
dvb-usb-az6027	Azurewave	DVB-S/S2 USB2.0 AZ6027
dvb-usb-ce6230	Intel	CE6230 DVB-T USB2.0
dvb-usb-cinergyT2	Terratec	CinergyT2/qanu USB 2.0 DVB-T
dvb-usb-cxusb	Conexant	USB2.0 hybrid
dvb-usb-dib0700	DiBcom	DiB0700
dvb-usb-dibusb-common	DiBcom	DiB3000M-B
dvb-usb-dibusb-mc	DiBcom	DiB3000M-C/P
dvb-usb-digitv	Nebula Electronics	uDigiTV DVB-T USB2.0
dvb-usb-dtt200u	WideView	WT-200U and WT-220U (pen) DVB-T
dvb-usb-dtv5100	AME	DTV-5100 USB2.0 DVB-T
dvb-usb-dvbsky	DVBSky	USB
dvb-usb-dw2102	DvbWorld & TeVii	DVB-S/S2 USB2.0
dvb-usb-ec168	E3C	EC168 DVB-T USB2.0
dvb-usb-gl861	Genesys Logic	GL861 USB2.0
dvb-usb-gp8psk	GENPIX	8PSK->USB module
dvb-usb-lmedm04	LME	DM04/QQBOX DVB-S USB2.0
dvb-usb-m920x	Uli	m920x DVB-T USB2.0
dvb-usb-nova-t-usb2	Hauppauge	WinTV-NOVA-T usb2 DVB-T USB2.0
dvb-usb-opera	Opera1	DVB-S USB2.0 receiver
dvb-usb-pctv452e	Pinnacle	PCTV HDTV Pro USB device/TT Connect S2-3600
dvb-usb-rtl28xxu	Realtek	RTL28xxU DVB USB

Continued on next page

<sup>1</sup> some of the drivers have sub-drivers, not shown at this table. In particular, `gspca` driver has lots of sub-drivers, for cameras not supported by the USB Video Class (UVC) driver, as shown at `gspca` card list.

Table 1 - continued from previous page

	Driver	Name
dvb-usb-technisat-usb2	Technisat	DVB-S/S2 USB2.0
dvb-usb-ttusb2	Pinnacle	400e DVB-S USB2.0
dvb-usb-umt-010	HanfTek	UMT-010 DVB-T USB2.0
dvb_usb_v2	Support for various USB DVB devices	v2
dvb-usb-vp702x	TwinhanDTV	StarBox and clones DVB-S USB2.0
dvb-usb-vp7045	TwinhanDTV	Alpha/MagicBoxII, DNTV tinyUSB2, Beetle USB2.0
em28xx	Empia	EM28xx USB devices
go7007	WIS	GO7007 MPEG encoder
gspca	Drivers for several USB Cameras	
hackrf	HackRF	
hdpvr	Hauppauge	HD PVR
msi2500	Mirics	MSi2500
mxl111sf-tuner	MxL111SF	DTV USB2.0
pvrusb2	Hauppauge	WinTV-PVR USB2
pwc	USB Philips	Cameras
s2250	Sensoray	2250/2251
s2255drv	USB Sensoray	2255 video capture device
smsusb	Siano	SMS1xxx based MDTV receiver
stkwebcam	USB Syntek	DC1125 Camera
tm6000-alsa	TV Master	TM5600/6000/6010 audio
tm6000-dvb	DVB Support for tm6000 based	TV cards
tm6000	TV Master	TM5600/6000/6010 driver
ttusb_dec	Technotrend/Hauppauge	USB DEC devices
usbtv	USBTv007	video capture
uvcvideo	USB Video Class (UVC)	
zd1301	ZyDAS	ZD1301
zr364xx	USB ZR364XX	Camera

### AU0828 cards list

Card number	Card name	USB IDs
0	Unknown board	
1	Hauppauge HVR950Q	2040:7200, 2040:7210, 2040:7217, 2040:721e, 2040:721f, 2040:7280, 0fd9:0000, 2040:7260, 2040:7213, 2040:7270
2	Hauppauge HVR850	2040:7240
3	DViCO FusionHDTV USB	0fe9:d620
4	Hauppauge HVR950Q rev xxF8	2040:7201, 2040:7211, 2040:7281
5	Hauppauge Woodbury	05e1:0480, 2040:8200

## cx231xx cards list

Card number	Card name	USB IDs
0	Unknown CX231xx video grabber	0572:5A3C
1	Conexant Hybrid TV - CARRAERA	0572:58A2
2	Conexant Hybrid TV - SHELBY	0572:58A1
3	Conexant Hybrid TV - RDE253S	0572:58A4
4	Conexant Hybrid TV - RDU253S	0572:58A5
5	Conexant VIDEO GRABBER	0572:58A6, 07ca:c039
6	Conexant Hybrid TV - rde 250	0572:589E
7	Conexant Hybrid TV - RDU 250	0572:58A0
8	Hauppauge EXETER	2040:b120, 2040:b140
9	Hauppauge USB Live 2	2040:c200
10	Pixelview PlayTV USB Hybrid	4000:4001
11	Pixelview Xcapture USB	1D19:6109, 4000:4001
12	Kworld UB430 USB Hybrid	1b80:e424
13	Iconbit Analog Stick U100 FM	1f4d:0237
14	Hauppauge WinTV USB2 FM (PAL)	2040:b110
15	Hauppauge WinTV USB2 FM (NTSC)	2040:b111
16	Elgato Video Capture V2	0fd9:0037
17	Geniatech OTG102	1f4d:0102
18	Kworld UB445 USB Hybrid	1b80:e421
19	Hauppauge WinTV 930C-HD (1113xx) / HVR-900H (111xxx) / PCTV QuatroStick 521e	2040:b130, 2040:b131, 2013:0259
20	Hauppauge WinTV 930C-HD (1114xx) / HVR-901H (1114xx) / PCTV QuatroStick 522e	2040:b131, 2040:b132, 2013:025e
21	Hauppauge WinTV-HVR-955Q (111401)	2040:b123, 2040:b124
22	Terratec Grabby	1f4d:0102
23	Evromedia USB Full Hybrid Full HD	1b80:d3b2
24	Astrometa T2hybrid	15f4:0135
25	The Imaging Source DFG/USB2pro	199e:8002
26	Hauppauge WinTV-HVR-935C	2040:b151
27	Hauppauge WinTV-HVR-975	2040:b150

## EM28xx cards list

Card number	Card name	Empia Chip	USB IDs
0	Unknown EM2800 video grabber	em2800	eb1a:2800

Continued on next page

Table 2 - continued from previous page

Card number	Card name	Empia Chip	USB IDs
1	Unknown EM2750/28xx video grabber	em2820 or em2840	eb1a:2710, eb1a:2820, eb1a:2821, eb1a:2860, eb1a:2861, eb1a:2862, eb1a:2863, eb1a:2870, eb1a:2881, eb1a:2883, eb1a:2868, eb1a:2875
2	Terratec Cinergy 250 USB	em2820 or em2840	0ccd:0036
3	Pinnacle PCTV USB 2	em2820 or em2840	2304:0208
4	Hauppauge WinTV USB 2	em2820 or em2840	2040:4200, 2040:4201
5	MSI VOX USB 2.0	em2820 or em2840	
6	Terratec Cinergy 200 USB	em2800	
7	Leadtek Winfast USB II	em2800	0413:6023
8	Kworld USB2800	em2800	
9	Pinnacle Dazzle DVC 90/100/101/107 / Kaiser Baas Video to DVD maker / Kworld DVD Maker 2 / Plextor ConvertX PX-AV100U	em2820 or em2840	1b80:e302, 1b80:e304, 2304:0207, 2304:021a, 093b:a003
10	Hauppauge WinTV HVR 900	em2880	2040:6500
11	Terratec Hybrid XS	em2880	
12	Kworld PVR TV 2800 RF	em2820 or em2840	
13	Terratec Prodigy XS	em2880	
14	SIIG AVTuner-PVR / Pixelview Prolink PlayTV USB 2.0	em2820 or em2840	
15	V-Gear PocketTV	em2800	
16	Hauppauge WinTV HVR 950	em2883	2040:6513, 2040:6517, 2040:651b
17	Pinnacle PCTV HD Pro Stick	em2880	2304:0227

Continued on next page

Table 2 - continued from previous page

Card number	Card name	Empia Chip	USB IDs
18	Hauppauge WinTV HVR 900 (R2)	em2880	2040:6502
19	EM2860/SAA711X Reference Design	em2860	
20	AMD ATI TV Wonder HD 600	em2880	0438:b002
21	eMPIA Technology, Inc. GrabBeeX+ Video Encoder	em2800	eb1a:2801
22	EM2710/EM2750/EM2751 webcam grabber	em2750	eb1a:2750, eb1a:2751
23	Huaqi DLCW-130	em2750	
24	D-Link DUB-T210 TV Tuner	em2820 or em2840	2001:f112
25	Gadmei UTV310	em2820 or em2840	
26	Hercules Smart TV USB 2.0	em2820 or em2840	
27	Pinnacle PCTV USB 2 (Philips FM1216ME)	em2820 or em2840	
28	Leadtek Winfast USB II Deluxe	em2820 or em2840	
29	EM2860/TVP5150 Reference Design	em2860	eb1a:5051
30	Videology 20K14XUSB USB2.0	em2820 or em2840	
31	Usbgear VD204v9	em2821	
32	Supercomp USB 2.0 TV	em2821	
33	Elgato Video Capture	em2860	0fd9:0033
34	Terratec Cinergy A Hybrid XS	em2860	0ccd:004f
35	Typhoon DVD Maker	em2860	
36	NetGMBH Cam	em2860	
37	Gadmei UTV330	em2860	eb1a:50a6
38	Yakumo MovieMixer	em2861	
39	KWorld PVRTV 300U	em2861	eb1a:e300
40	Plextor ConvertX PX-TV100U	em2861	093b:a005
41	Kworld 350 U DVB-T	em2870	eb1a:e350
42	Kworld 355 U DVB-T	em2870	eb1a:e355, eb1a:e357, eb1a:e359
43	Terratec Cinergy T XS	em2870	
44	Terratec Cinergy T XS (MT2060)	em2870	0ccd:0043
45	Pinnacle PCTV DVB-T	em2870	
46	Compro, VideoMate U3	em2870	185b:2870
47	KWorld DVB-T 305U	em2880	eb1a:e305

Continued on next page

Table 2 - continued from previous page

Card number	Card name	Empia Chip	USB IDs
48	KWorld DVB-T 310U	em2880	
49	MSI DigiVox A/D	em2880	eb1a:e310
50	MSI DigiVox A/D II	em2880	eb1a:e320
51	Terratec Hybrid XS Secam	em2880	0ccd:004c
52	DNT DA2 Hybrid	em2881	
53	Pinnacle Hybrid Pro	em2881	
54	Kworld VS-DVB-T 323UR	em2882	eb1a:e323
55	Terratec Cinergy Hybrid T USB XS (em2882)	em2882	0ccd:005e, 0ccd:0042
56	Pinnacle Hybrid Pro (330e)	em2882	2304:0226
57	Kworld PlusTV HD Hybrid 330	em2883	eb1a:a316
58	Compro VideoMate ForYou/Stereo	em2820 or em2840	185b:2041
59	Pinnacle PCTV HD Mini	em2874	2304:023f
60	Hauppauge WinTV HVR 850	em2883	2040:651f
61	Pixelview PlayTV Box 4 USB 2.0	em2820 or em2840	
62	Gadmei TVR200	em2820 or em2840	
63	Kaiomy TVnPC U2	em2860	eb1a:e303
64	Easy Cap Capture DC-60	em2860	1b80:e309
65	IO-DATA GV-MVP/SZ	em2820 or em2840	04bb:0515
66	Empire dual TV	em2880	
67	Terratec Grabby	em2860	0ccd:0096, 0ccd:10AF
68	Terratec AV350	em2860	0ccd:0084
69	KWorld ATSC 315U HDTV TV Box	em2882	eb1a:a313
70	Evgar inDtube	em2882	
71	Silvercrest Webcam 1.3mpix	em2820 or em2840	
72	Gadmei UTV330+	em2861	
73	Reddo DVB-C USB TV Box	em2870	
74	Actionmaster/LinXcel/Digitus VC211A	em2800	
75	Dikom DK300	em2882	
76	KWorld PlusTV 340U or UB435-Q (ATSC)	em2870	1b80:a340
77	EM2874 Leadership ISDBT	em2874	
78	PCTV nanoStick T2 290e	em28174	2013:024f

Continued on next page

Table 2 - continued from previous page

Card number	Card name	Empia Chip	USB IDs
79	Terratec Cinergy H5	em2884	eb1a:2885, 0ccd:10a2, 0ccd:10ad, 0ccd:10b6
80	PCTV DVB-S2 Stick (460e)	em28174	2013:024c
81	Hauppauge WinTV HVR 930C	em2884	2040:1605
82	Terratec Cinergy HTC Stick	em2884	0ccd:00b2
83	Honestech Vidbox NW03	em2860	eb1a:5006
84	MaxMedia UB425-TC	em2874	1b80:e425
85	PCTV QuatroStick (510e)	em2884	2304:0242
86	PCTV QuatroStick nano (520e)	em2884	2013:0251
87	Terratec Cinergy HTC USB XS	em2884	0ccd:008e, 0ccd:00ac
88	C3 Tech Digital Duo HDTV/SDTV USB	em2884	1b80:e755
89	Delock 61959	em2874	1b80:e1cc
90	KWorld USB ATSC TV Stick UB435-Q V2	em2874	1b80:e346
91	SpeedLink Vicious And Devine Laplace webcam	em2765	1ae7:9003, 1ae7:9004
92	PCTV DVB-S2 Stick (461e)	em28178	2013:0258
93	KWorld USB ATSC TV Stick UB435-Q V3	em2874	1b80:e34c
94	PCTV tripleStick (292e)	em28178	2013:025f, 2013:0264, 2040:0264, 2040:8264, 2040:8268
95	Leadtek VC100	em2861	0413:6f07
96	Terratec Cinergy T2 Stick HD	em28178	eb1a:8179
97	Elgato EyeTV Hybrid 2008 INT	em2884	0fd9:0018
98	PLEX PX-BCUD	em28178	3275:0085
99	Hauppauge WinTV-dualHD DVB	em28174	2040:0265, 2040:8265
100	Hauppauge WinTV-dualHD 01595 ATSC/QAM	em28174	2040:026d, 2040:826d
101	Terratec Cinergy H6 rev. 2	em2884	0ccd:10b2
102	:ZOLID HYBRID TV STICK	em2882	
103	Magix USB Videowandler-2	em2861	1b80:e349
104	PCTV DVB-S2 Stick (461e v2)	em28178	2013:0461, 2013:0259

## TM6000 cards list

Card number	Card name	USB IDs
0	Unknown tm6000 video grabber	
1	Generic tm5600 board	6000:0001
2	Generic tm6000 board	
3	Generic tm6010 board	6000:0002
4	10Moons UT 821	
5	10Moons UT 330	
6	ADSTECH Dual TV USB	06e1:f332
7	Freecom Hybrid Stick / Moka DVB-T Receiver Dual	14aa:0620
8	ADSTECH Mini Dual TV USB	06e1:b339
9	Hauppauge WinTV HVR-900H / WinTV USB2-Stick	2040:6600, 2040:6601, 2040:6610, 2040:6611
10	Beholder Wander DVB-T/TV/FM USB2.0	6000:dec0
11	Beholder Voyager TV/FM USB2.0	6000:dec1
12	Terratec Cinergy Hybrid XE / Cinergy Hybrid-Stick	0ccd:0086, 0ccd:00A5
13	Twinhan TU501(704D1)	13d3:3240, 13d3:3241, 13d3:3243, 13d3:3264
14	Beholder Wander Lite DVB-T/TV/FM USB2.0	6000:dec2
15	Beholder Voyager Lite TV/FM USB2.0	6000:dec3
16	Terratec Grabster AV 150/250 MX	0ccd:0079

**Siano cards list**

Card name	USB IDs
Hauppauge Catamount	2040:1700
Hauppauge Okemo-A	2040:1800
Hauppauge Okemo-B	2040:1801
Hauppauge WinTV MiniCard	2040:2000, 2040:200a, 2040:2010, 2040:2011, 2040:2019
Hauppauge WinTV MiniCard	2040:2009
Hauppauge WinTV MiniStick	2040:5500, 2040:5510, 2040:5520, 2040:5530, 2040:5580, 2040:5590, 2040:b900, 2040:b910, 2040:b980, 2040:b990, 2040:c000, 2040:c010, 2040:c080, 2040:c090, 2040:c0a0, 2040:f5a0
Hauppauge microStick 77e	2013:0257
ONDA Data Card Digital Receiver	19D2:0078
Siano Denver (ATSC-M/H) Digital Receiver	187f:0800
Siano Denver (TDMB) Digital Receiver	187f:0700
Siano Ming Digital Receiver	187f:0310
Siano Nice Digital Receiver	187f:0202, 187f:0202
Siano Nova A Digital Receiver	187f:0200
Siano Nova B Digital Receiver	187f:0201
Siano Pele Digital Receiver	187f:0500
Siano Rio Digital Receiver	187f:0600, 3275:0080
Siano Stellar Digital Receiver	187f:0100
Siano Stellar Digital Receiver ROM	187f:0010
Siano Vega Digital Receiver	187f:0300
Siano Venice Digital Receiver	187f:0301, 187f:0301, 187f:0302
ZTE Data Card Digital Receiver	19D2:0086

## USBvision cards list

Card number	Card name	USB IDs
0	Xanboo	0a6f:0400
1	Belkin USB VideoBus II Adapter	050d:0106
2	Belkin Components USB VideoBus	050d:0207
3	Belkin USB VideoBus II	050d:0208
4	echoFX InterView Lite	0571:0002
5	USBGear USBG-V1 resp. HAMA USB	0573:0003
6	D-Link V100	0573:0400
7	X10 USB Camera	0573:2000
8	Hauppauge WinTV USB Live (PAL B/G)	0573:2d00
9	Hauppauge WinTV USB Live Pro (NTSC M/N)	0573:2d01
10	Zoran Co. PMD (Nogatech) AV-grabber Manhattan	0573:2101
11	Nogatech USB-TV (NTSC) FM	0573:4100
12	PNY USB-TV (NTSC) FM	0573:4110
13	PixelView PlayTv-USB PRO (PAL) FM	0573:4450
14	ZTV ZT-721 2.4GHz USB A/V Receiver	0573:4550
15	Hauppauge WinTV USB (NTSC M/N)	0573:4d00
16	Hauppauge WinTV USB (PAL B/G)	0573:4d01
17	Hauppauge WinTV USB (PAL I)	0573:4d02
18	Hauppauge WinTV USB (PAL/SECAM L)	0573:4d03
19	Hauppauge WinTV USB (PAL D/K)	0573:4d04
20	Hauppauge WinTV USB (NTSC FM)	0573:4d10
21	Hauppauge WinTV USB (PAL B/G FM)	0573:4d11
22	Hauppauge WinTV USB (PAL I FM)	0573:4d12
23	Hauppauge WinTV USB (PAL D/K FM)	0573:4d14
24	Hauppauge WinTV USB Pro (NTSC M/N)	0573:4d2a
25	Hauppauge WinTV USB Pro (NTSC M/N) V2	0573:4d2b
26	Hauppauge WinTV USB Pro (PAL/SECAM B/G/I/D/K/L)	0573:4d2c
27	Hauppauge WinTV USB Pro (NTSC M/N) V3	0573:4d20
28	Hauppauge WinTV USB Pro (PAL B/G)	0573:4d21
29	Hauppauge WinTV USB Pro (PAL I)	0573:4d22
30	Hauppauge WinTV USB Pro (PAL/SECAM L)	0573:4d23
31	Hauppauge WinTV USB Pro (PAL D/K)	0573:4d24
32	Hauppauge WinTV USB Pro (PAL/SECAM BGDK/I/L)	0573:4d25
33	Hauppauge WinTV USB Pro (PAL/SECAM BGDK/I/L) V2	0573:4d26
34	Hauppauge WinTV USB Pro (PAL B/G) V2	0573:4d27
35	Hauppauge WinTV USB Pro (PAL B/G,D/K)	0573:4d28
36	Hauppauge WinTV USB Pro (PAL I,D/K)	0573:4d29
37	Hauppauge WinTV USB Pro (NTSC M/N FM)	0573:4d30
38	Hauppauge WinTV USB Pro (PAL B/G FM)	0573:4d31
39	Hauppauge WinTV USB Pro (PAL I FM)	0573:4d32
40	Hauppauge WinTV USB Pro (PAL D/K FM)	0573:4d34
41	Hauppauge WinTV USB Pro (Temic PAL/SECAM B/G/I/D/K/L FM)	0573:4d35

Continued on next page

Table 3 - continued from previous page

Card number	Card name	USB IDs
42	Hauppauge WinTV USB Pro (Temic PAL B/G FM)	0573:4d36
43	Hauppauge WinTV USB Pro (PAL/SECAM B/G/I/D/K/L FM)	0573:4d37
44	Hauppauge WinTV USB Pro (NTSC M/N FM) V2	0573:4d38
45	Camtel Technology USB TV Genie Pro FM Model TVB330	0768:0006
46	Digital Video Creator I	07d0:0001
47	Global Village GV-007 (NTSC)	07d0:0002
48	Dazzle Fusion Model DVC-50 Rev 1 (NTSC)	07d0:0003
49	Dazzle Fusion Model DVC-80 Rev 1 (PAL)	07d0:0004
50	Dazzle Fusion Model DVC-90 Rev 1 (SECAM)	07d0:0005
51	Eskape Labs MyTV2Go	07f8:9104
52	Pinnacle Studio PCTV USB (PAL)	2304:010d
53	Pinnacle Studio PCTV USB (SECAM)	2304:0109
54	Pinnacle Studio PCTV USB (PAL) FM	2304:0110
55	Miro PCTV USB	2304:0111
56	Pinnacle Studio PCTV USB (NTSC) FM	2304:0112
57	Pinnacle Studio PCTV USB (PAL) FM V2	2304:0210
58	Pinnacle Studio PCTV USB (NTSC) FM V2	2304:0212
59	Pinnacle Studio PCTV USB (PAL) FM V3	2304:0214
60	Pinnacle Studio Linx Video input cable (NTSC)	2304:0300
61	Pinnacle Studio Linx Video input cable (PAL)	2304:0301
62	Pinnacle PCTV Bungee USB (PAL) FM	2304:0419
63	Hauppauge WinTv-USB	2400:4200
64	Pinnacle Studio PCTV USB (NTSC) FM V3	2304:0113
65	Nogatech USB MicroCam NTSC (NV3000N)	0573:3000
66	Nogatech USB MicroCam PAL (NV3001P)	0573:3001

### The gspca cards list

The modules for the gspca webcam drivers are:

- gspca\_main: main driver
- gspca\_driver: subdriver module with driver as follows

	driver	vend:prod	Device
spca501	0000:0000	MystFromOri	Unknown Camera
spca508	0130:0130	Clone Digital	Webcam 11043
se401	03e8:0004	Endpoints/Aox	SE401
zc3xx	03f0:1b07	HP Premium	Starter Cam
m5602	0402:5602	ALi Video	Camera Controller
spca501	040a:0002	Kodak DVC-	325
spca500	040a:0300	Kodak EZ	200
zc3xx	041e:041e	Creative Web	Cam Live!
ov519	041e:4003	Video Blaster	WebCam Go Plus

Continued on next page

Table 4 - continued from previous page

	driver	vend:prod	Device
stv0680	041e:4007		Go Mini
spca500	041e:400a		Creative PC-CAM 300
sunplus	041e:400b		Creative PC-CAM 600
sunplus	041e:4012		PC-Cam350
sunplus	041e:4013		Creative Pccam750
zc3xx	041e:4017		Creative Webcam Mobile PD1090
spca508	041e:4018		Creative Webcam Vista (PD1100)
spca561	041e:401a		Creative Webcam Vista (PD1100)
zc3xx	041e:401c		Creative NX
spca505	041e:401d		Creative Webcam NX ULTRA
zc3xx	041e:401e		Creative Nx Pro
zc3xx	041e:401f		Creative Webcam Notebook PD1171
zc3xx	041e:4022		Webcam NX Pro
pac207	041e:4028		Creative Webcam Vista Plus
zc3xx	041e:4029		Creative WebCam Vista Pro
zc3xx	041e:4034		Creative Instant P0620
zc3xx	041e:4035		Creative Instant P0620D
zc3xx	041e:4036		Creative Live !
sq930x	041e:4038		Creative Joy-IT
zc3xx	041e:403a		Creative Nx Pro 2
spca561	041e:403b		Creative Webcam Vista (VF0010)
sq930x	041e:403c		Creative Live! Ultra
sq930x	041e:403d		Creative Live! Ultra for Notebooks
sq930x	041e:4041		Creative Live! Motion
zc3xx	041e:4051		Creative Live!Cam Notebook Pro (VF0250)
ov519	041e:4052		Creative Live! VISTA IM
zc3xx	041e:4053		Creative Live!Cam Video IM
vc032x	041e:405b		Creative Live! Cam Notebook Ultra (VC0130)
ov519	041e:405f		Creative Live! VISTA VF0330
ov519	041e:4060		Creative Live! VISTA VF0350
ov519	041e:4061		Creative Live! VISTA VF0400
ov519	041e:4064		Creative Live! VISTA VF0420
ov519	041e:4067		Creative Live! Cam Video IM (VF0350)
ov519	041e:4068		Creative Live! VISTA VF0470
sn9c2028	0458:7003		GeniusVideocam Live v2
spca561	0458:7004		Genius VideoCAM Express V2
sn9c2028	0458:7005		Genius Smart 300, version 2
sunplus	0458:7006		Genius Dsc 1.3 Smart
zc3xx	0458:7007		Genius VideoCam V2
zc3xx	0458:700c		Genius VideoCam V3
zc3xx	0458:700f		Genius VideoCam Web V2
sonixj	0458:7025		Genius Eye 311Q
sn9c20x	0458:7029		Genius Look 320s
sonixj	0458:702e		Genius Slim 310 NB
sn9c20x	0458:7045		Genius Look 1320 V2
sn9c20x	0458:704a		Genius Slim 1320
sn9c20x	0458:704c		Genius i-Look 1321

Continued on next page

Table 4 – continued from previous page

	driver	vend:prod	Device
sn9c20x	045e:00f4		LifeCam VX-6000 (SN9C20x + OV9650)
sonixj	045e:00f5		MicroSoft VX3000
sonixj	045e:00f7		MicroSoft VX1000
ov519	045e:028c		Micro\$oft xbox cam
kinect	045e:02ae		Xbox NUI Camera
kinect	045e:02bf		Kinect for Windows NUI Camera
spca561	0461:0815		Micro Innovations IC200 Webcam
sunplus	0461:0821		Fujifilm MV-1
zc3xx	0461:0a00		MicroInnovation WebCam320
stv06xx	046D:08F0		QuickCamMessenger
stv06xx	046D:08F5		QuickCamCommunicate
stv06xx	046D:08F6		QuickCamMessenger (new)
stv06xx	046d:0840		QuickCamExpress
stv06xx	046d:0850		LEGOcam / QuickCam Web
stv06xx	046d:0870		DexxaWebCam USB
spca500	046d:0890		Logitech QuickCam traveler
vc032x	046d:0892		Logitech Orbicam
vc032x	046d:0896		Logitech Orbicam
vc032x	046d:0897		Logitech QuickCam for Dell notebooks
zc3xx	046d:089d		Logitech QuickCam E2500
zc3xx	046d:08a0		Logitech QC IM
zc3xx	046d:08a1		Logitech QC IM 0x08A1 +sound
zc3xx	046d:08a2		Labtec Webcam Pro
zc3xx	046d:08a3		Logitech QC Chat
zc3xx	046d:08a6		Logitech QCim
zc3xx	046d:08a7		Logitech QuickCam Image
zc3xx	046d:08a9		Logitech Notebook Deluxe
zc3xx	046d:08aa		Labtec Webcam Notebook
zc3xx	046d:08ac		Logitech QuickCam Cool
zc3xx	046d:08ad		Logitech QCCommunicate STX
zc3xx	046d:08ae		Logitech QuickCam for Notebooks
zc3xx	046d:08af		Logitech QuickCam Cool
zc3xx	046d:08b9		Logitech QuickCam Express
zc3xx	046d:08d7		Logitech QCam STX
zc3xx	046d:08d8		Logitech Notebook Deluxe
zc3xx	046d:08d9		Logitech QuickCam IM/Connect
zc3xx	046d:08da		Logitech QuickCam Messenger
zc3xx	046d:08dd		Logitech QuickCam for Notebooks
spca500	046d:0900		Logitech Inc. ClickSmart 310
spca500	046d:0901		Logitech Inc. ClickSmart 510
sunplus	046d:0905		Logitech ClickSmart 820
tv8532	046d:0920		Logitech QuickCam Express
tv8532	046d:0921		Labtec Webcam
spca561	046d:0928		Logitech QC Express Etch2
spca561	046d:0929		Labtec Webcam Elch2
spca561	046d:092a		Logitech QC for Notebook
spca561	046d:092b		Labtec Webcam Plus

Continued on next page

Table 4 – continued from previous page

	driver	vend:prod	Device
spca561	046d:092c	Logitech QC chat Elch2	
spca561	046d:092d	Logitech QC Elch2	
spca561	046d:092e	Logitech QC Elch2	
spca561	046d:092f	Logitech QuickCam Express Plus	
sunplus	046d:0960	Logitech ClickSmart 420	
nw80x	046d:d001	Logitech QuickCam Pro (dark focus ring)	
se401	0471:030b	PhilipsPCVC665K	
sunplus	0471:0322	Philips DMVC1300K	
zc3xx	0471:0325	Philips SPC 200 NC	
zc3xx	0471:0326	Philips SPC 300 NC	
sonixj	0471:0327	Philips SPC 600 NC	
sonixj	0471:0328	Philips SPC 700 NC	
zc3xx	0471:032d	Philips SPC 210 NC	
zc3xx	0471:032e	Philips SPC 315 NC	
sonixj	0471:0330	Philips SPC 710 NC	
se401	047d:5001	Kensington67014	
se401	047d:5002	Kensington6701(5/7)	
se401	047d:5003	Kensington67016	
spca501	0497:c001	Smile International	
sunplus	04a5:3003	Benq DC 1300	
sunplus	04a5:3008	Benq DC 1500	
sunplus	04a5:300a	Benq DC 3410	
spca500	04a5:300c	Benq DC 1016	
benq	04a5:3035	Benq DC E300	
vicam	04c1:009d	HomeConnect Webcam [vicam]	
konica	04c8:0720	IntelYC 76	
finepix	04cb:0104	Fujifilm FinePix 4800	
finepix	04cb:0109	Fujifilm FinePix A202	
finepix	04cb:010b	Fujifilm FinePix A203	
finepix	04cb:010f	Fujifilm FinePix A204	
finepix	04cb:0111	Fujifilm FinePix A205	
finepix	04cb:0113	Fujifilm FinePix A210	
finepix	04cb:0115	Fujifilm FinePix A303	
finepix	04cb:0117	Fujifilm FinePix A310	
finepix	04cb:0119	Fujifilm FinePix F401	
finepix	04cb:011b	Fujifilm FinePix F402	
finepix	04cb:011d	Fujifilm FinePix F410	
finepix	04cb:0121	Fujifilm FinePix F601	
finepix	04cb:0123	Fujifilm FinePix F700	
finepix	04cb:0125	Fujifilm FinePix M603	
finepix	04cb:0127	Fujifilm FinePix S300	
finepix	04cb:0129	Fujifilm FinePix S304	
finepix	04cb:012b	Fujifilm FinePix S500	
finepix	04cb:012d	Fujifilm FinePix S602	
finepix	04cb:012f	Fujifilm FinePix S700	
finepix	04cb:0131	Fujifilm FinePix unknown model	
finepix	04cb:013b	Fujifilm FinePix unknown model	

Continued on next page

Table 4 - continued from previous page

	driver	vend:prod	Device
finepix	04cb:013d	Fujifilm FinePix	unknown model
finepix	04cb:013f	Fujifilm FinePix	F420
sunplus	04f1:1001	JVC GC	A50
spca561	04fc:0561	Flexcam	100
spca1528	04fc:1528	Sunplus MD80	clone
sunplus	04fc:500c	Sunplus CA500C	
sunplus	04fc:504a	Aiptek Mini PenCam	1.3
sunplus	04fc:504b	Maxell MaxPocket LE	1.3
sunplus	04fc:5330	Digitrex	2110
sunplus	04fc:5360	Sunplus Generic	
spca500	04fc:7333	PalmPixDC85	
sunplus	04fc:ffff	Pure DigitalDakota	
nw80x	0502:d001	DVC	V6
spca501	0506:00df	3Com HomeConnect	Lite
sunplus	052b:1507	Megapixel 5 Pretec	DC-1007
sunplus	052b:1513	Megapix	V4
sunplus	052b:1803	MegaImage	VI
nw80x	052b:d001	EZCam Pro	p35u
tv8532	0545:808b	Veo	Stingray
tv8532	0545:8333	Veo	Stingray
sunplus	0546:3155	Polaroid PDC3070	
sunplus	0546:3191	Polaroid Ion	80
sunplus	0546:3273	Polaroid PDC2030	
touptek	0547:6801	TTUCMOS08000KPB, AS	MU800
dtcs033	0547:7303	Anchor Chips, Inc	
ov519	054c:0154	Sonny toy4	
ov519	054c:0155	Sonny toy5	
cpia1	0553:0002	CPIA CPiA (version1)	based cameras
stv0680	0553:0202	STV0680	Camera
zc3xx	055f:c005	Mustek Wcam300A	
spca500	055f:c200	Mustek Gsmart	300
sunplus	055f:c211	Kowa Bs888e	Microcamera
spca500	055f:c220	Gsmart	Mini
sunplus	055f:c230	Mustek Digicam	330K
sunplus	055f:c232	Mustek MDC3500	
sunplus	055f:c360	Mustek DV4000	Mpeg4
sunplus	055f:c420	Mustek gSmart	Mini 2
sunplus	055f:c430	Mustek Gsmart	LCD 2
sunplus	055f:c440	Mustek DV	3000
sunplus	055f:c520	Mustek gSmart	Mini 3
sunplus	055f:c530	Mustek Gsmart	LCD 3
sunplus	055f:c540	Gsmart	D30
sunplus	055f:c630	Mustek MDC4000	
sunplus	055f:c650	Mustek MDC5500Z	
nw80x	055f:d001	Mustek Wcam	300 mini
zc3xx	055f:d003	Mustek WCam300A	
zc3xx	055f:d004	Mustek WCam300	AN

Continued on next page

Table 4 - continued from previous page

	driver	vend:prod	Device
conex	0572:0041		Creative Notebook cx11646
ov519	05a9:0511		Video Blaster WebCam 3/WebCam Plus, D-Link USB Digital Video C
ov519	05a9:0518		Creative WebCam
ov519	05a9:0519		OV519 Microphone
ov519	05a9:0530		OmniVision
ov534_9	05a9:1550		OmniVision VEHO FilmScanner
ov519	05a9:2800		OmniVision SuperCAM
ov519	05a9:4519		Webcam Classic
ov534_9	05a9:8065		OmniVision test kit ov538+ov9712
ov519	05a9:8519		OmniVision
ov519	05a9:a511		D-Link USB Digital Video Camera
ov519	05a9:a518		D-Link DSB-C310 Webcam
sunplus	05da:1018		Digital Dream Enigma 1.3
stk014	05e1:0893		Syntek DV4000
gl860	05e3:0503		Genesys Logic PC Camera
gl860	05e3:f191		Genesys Logic PC Camera
vicam	0602:1001		ViCam Webcam
spca561	060b:a001		Maxell Compact Pc PM3
zc3xx	0698:2003		CTX M730V built in
topro	06a2:0003		TP6800 PC Camera, CmoX CX0342 webcam
topro	06a2:6810		Creative Qmax
nw80x	06a5:0000		Typhoon Webcam 100 USB
nw80x	06a5:d001		Divio based webcams
nw80x	06a5:d800		Divio Chicony TwinkleCam, Trust SpaceCam
spca500	06bd:0404		Agfa CL20
spca500	06be:0800		Optimedia
nw80x	06be:d001		EZCam Pro p35u
sunplus	06d6:0031		Trust 610 LCD PowerC@m Zoom
sunplus	06d6:0041		Aashima Technology B.V.
spca506	06e1:a190		ADS Instant VCD
ov534	06f8:3002		Hercules Blog Webcam
ov534_9	06f8:3003		Hercules Dualpix HD Weblog
sonixj	06f8:3004		Hercules Classic Silver
sonixj	06f8:3008		Hercules Deluxe Optical Glass
pac7302	06f8:3009		Hercules Classic Link
pac7302	06f8:301b		Hercules Link
nw80x	0728:d001		AVerMedia Camguard
spca508	0733:0110		ViewQuest VQ110
spca501	0733:0401		Intel Create and Share
spca501	0733:0402		ViewQuest M318B
spca505	0733:0430		Intel PC Camera Pro
sunplus	0733:1311		Digital Dream Epsilon 1.3
sunplus	0733:1314		Mercury 2.1MEG Deluxe Classic Cam
sunplus	0733:2211		Jenoptik jdc 21 LCD
sunplus	0733:2221		Mercury Digital Pro 3.1p
sunplus	0733:3261		Concord 3045 spca536a
sunplus	0733:3281		Cyberpix S550V

Continued on next page

Table 4 - continued from previous page

		driver	vend:prod	Device
spca506	0734:043b			3DeMon USB Capture aka
cpia1	0813:0001			QX3 camera
ov519	0813:0002			Dual Mode USB Camera Plus
spca500	084d:0003			D-Link DSC-350
spca500	08ca:0103			Aiptek PocketDV
sunplus	08ca:0104			Aiptek PocketDVII 1.3
sunplus	08ca:0106			Aiptek Pocket DV3100+
mr97310a	08ca:0110			Trust Spyc@m 100
mr97310a	08ca:0111			Aiptek PenCam VGA+
sunplus	08ca:2008			Aiptek Mini PenCam 2 M
sunplus	08ca:2010			Aiptek PocketCam 3M
sunplus	08ca:2016			Aiptek PocketCam 2 Mega
sunplus	08ca:2018			Aiptek Pencam SD 2M
sunplus	08ca:2020			Aiptek Slim 3000F
sunplus	08ca:2022			Aiptek Slim 3200
sunplus	08ca:2024			Aiptek DV3500 Mpeg4
sunplus	08ca:2028			Aiptek PocketCam4M
sunplus	08ca:2040			Aiptek PocketDV4100M
sunplus	08ca:2042			Aiptek PocketDV5100
sunplus	08ca:2050			Medion MD 41437
sunplus	08ca:2060			Aiptek PocketDV5300
tv8532	0923:010f			ICM532 cams
mr97310a	093a:010e			All known CIF cams with this ID
mr97310a	093a:010f			All known VGA cams with this ID
mars	093a:050f			Mars-Semi Pc-Camera
pac207	093a:2460			Qtec Webcam 100
pac207	093a:2461			HP Webcam
pac207	093a:2463			Philips SPC 220 NC
pac207	093a:2464			Labtec Webcam 1200
pac207	093a:2468			Webcam WB-1400T
pac207	093a:2470			Genius GF112
pac207	093a:2471			Genius VideoCam ge111
pac207	093a:2472			Genius VideoCam ge110
pac207	093a:2474			Genius iLook 111
pac207	093a:2476			Genius e-Messenger 112
pac7311	093a:2600			PAC7311 Typhoon
pac7311	093a:2601			Philips SPC 610 NC
pac7311	093a:2603			Philips SPC 500 NC
pac7311	093a:2608			Trust WB-3300p
pac7311	093a:260e			Gigaware VGA PC Camera, Trust WB-3350p, SIGMA cam 2350
pac7311	093a:260f			SnakeCam
pac7302	093a:2620			Apollo AC-905
pac7302	093a:2621			PAC731x
pac7302	093a:2622			Genius Eye 312
pac7302	093a:2623			Pixart Imaging, Inc.
pac7302	093a:2624			PAC7302
pac7302	093a:2625			Genius iSlim 310

Continued on next page

Table 4 - continued from previous page

		driver	vend:prod	Device
pac7302	093a:2626	Labtec	2200	
pac7302	093a:2627	Genius	FaceCam 300	
pac7302	093a:2628	Genius	iLook 300	
pac7302	093a:2629	Genius	iSlim 300	
pac7302	093a:262a	Webcam	300k	
pac7302	093a:262c	Philips	SPC 230 NC	
jl2005bcd	0979:0227	Various brands, 19 known cameras supported		
jeilinj	0979:0270	Sakar	57379	
jeilinj	0979:0280	Sportscam	DV15, Sakar 57379	
zc3xx	0ac8:0301	Web	Camera	
zc3xx	0ac8:0302	Z-star	Vimicro zc0302	
vc032x	0ac8:0321	Vimicro	generic vc0321	
vc032x	0ac8:0323	Vimicro	Vc0323	
vc032x	0ac8:0328	A4Tech	PK-130MG	
zc3xx	0ac8:301b	Z-Star	zc301b	
zc3xx	0ac8:303b	Vimicro	0x303b	
zc3xx	0ac8:305b	Z-star	Vimicro zc0305b	
zc3xx	0ac8:307b	PC	Camera (ZS0211)	
vc032x	0ac8:c001	Sony	embedded vimicro	
vc032x	0ac8:c002	Sony	embedded vimicro	
vc032x	0ac8:c301	Samsung	Q1 Ultra Premium	
spca508	0af9:0010	Hama	USB Sightcam 100	
spca508	0af9:0011	Hama	USB Sightcam 100	
ov519	0b62:0059	iBOT2	Webcam	
sonixb	0c45:6001	Genius	VideoCAM NB	
sonixb	0c45:6005	Microdia	Sweex Mini Webcam	
sonixb	0c45:6007	Sonix	sn9c101 + Tas5110D	
sonixb	0c45:6009	<a href="#">spcaCam@120</a>		
sonixb	0c45:600d	<a href="#">spcaCam@120</a>		
sonixb	0c45:6011	Microdia	PC Camera (SN9C102)	
sonixb	0c45:6019	Generic	Sonix OV7630	
sonixb	0c45:6024	Generic	Sonix Tas5130c	
sonixb	0c45:6025	Xcam	Shanga	
sonixb	0c45:6027	GeniusEye	310	
sonixb	0c45:6028	Sonix	Btc Pc380	
sonixb	0c45:6029	<a href="#">spcaCam@150</a>		
sonixb	0c45:602a	Meade	ETX-105EC Camera	
sonixb	0c45:602c	Generic	Sonix OV7630	
sonixb	0c45:602d	LIC-200	LG	
sonixb	0c45:602e	Genius	VideoCam Messenger	
sonixj	0c45:6040	Speed	NVC 350K	
sonixj	0c45:607c	Sonix	sn9c102p Hv7131R	
sonixb	0c45:6083	VideoCAM	Look	
sonixb	0c45:608c	VideoCAM	Look	
sonixb	0c45:608f	PC	Camera (SN9C103 + OV7630)	
sonixb	0c45:60a8	VideoCAM	Look	
sonixb	0c45:60aa	VideoCAM	Look	

Continued on next page

Table 4 - continued from previous page

		driver	vend:prod	Device
sonixb	0c45:60af	VideoCAM	Look	
sonixb	0c45:60b0	Genius VideoCam	Look	
sonixj	0c45:60c0	Sangha	Sn535	
sonixj	0c45:60ce	USB-PC-Camera-168	(TALK-5067)	
sonixj	0c45:60ec	SN9C105+MO4000		
sonixj	0c45:60fb	Surfer	NoName	
sonixj	0c45:60fc	LG-LIC300		
sonixj	0c45:60fe	Microdia	Audio	
sonixj	0c45:6100	PC Camera	(SN9C128)	
sonixj	0c45:6102	PC Camera	(SN9C128)	
sonixj	0c45:610a	PC Camera	(SN9C128)	
sonixj	0c45:610b	PC Camera	(SN9C128)	
sonixj	0c45:610c	PC Camera	(SN9C128)	
sonixj	0c45:610e	PC Camera	(SN9C128)	
sonixj	0c45:6128	Microdia/Sonix	SNP325	
sonixj	0c45:612a	Avant	Camera	
sonixj	0c45:612b	Speed-Link	REFLECT2	
sonixj	0c45:612c	Typhoon Rasy	Cam 1.3MPix	
sonixj	0c45:612e	PC Camera	(SN9C110)	
sonixj	0c45:6130	Sonix	Pccam	
sonixj	0c45:6138	Sn9c120	Mo4000	
sonixj	0c45:613a	Microdia	Sonix PC Camera	
sonixj	0c45:613b	Surfer	SN-206	
sonixj	0c45:613c	Sonix	Pccam168	
sonixj	0c45:613e	PC Camera	(SN9C120)	
sonixj	0c45:6142	Hama	PC-Webcam AC-150	
sonixj	0c45:6143	Sonix	Pccam168	
sonixj	0c45:6148	Digitus DA-70811/ZSMC	USB PC Camera ZS211/Microdia	
sonixj	0c45:614a	Frontech	E-Ccam (JIL-2225)	
sn9c20x	0c45:6240	PC Camera	(SN9C201 + MT9M001)	
sn9c20x	0c45:6242	PC Camera	(SN9C201 + MT9M111)	
sn9c20x	0c45:6248	PC Camera	(SN9C201 + OV9655)	
sn9c20x	0c45:624c	PC Camera	(SN9C201 + MT9M112)	
sn9c20x	0c45:624e	PC Camera	(SN9C201 + SOI968)	
sn9c20x	0c45:624f	PC Camera	(SN9C201 + OV9650)	
sn9c20x	0c45:6251	PC Camera	(SN9C201 + OV9650)	
sn9c20x	0c45:6253	PC Camera	(SN9C201 + OV9650)	
sn9c20x	0c45:6260	PC Camera	(SN9C201 + OV7670)	
sn9c20x	0c45:6270	PC Camera	(SN9C201 + MT9V011/MT9V111/MT9V112)	
sn9c20x	0c45:627b	PC Camera	(SN9C201 + OV7660)	
sn9c20x	0c45:627c	PC Camera	(SN9C201 + HV7131R)	
sn9c20x	0c45:627f	PC Camera	(SN9C201 + OV9650)	
sn9c20x	0c45:6280	PC Camera	(SN9C202 + MT9M001)	
sn9c20x	0c45:6282	PC Camera	(SN9C202 + MT9M111)	
sn9c20x	0c45:6288	PC Camera	(SN9C202 + OV9655)	
sn9c20x	0c45:628c	PC Camera	(SN9C201 + MT9M112)	
sn9c20x	0c45:628e	PC Camera	(SN9C202 + SOI968)	

Continued on next page

Table 4 - continued from previous page

		driver	vend:prod	Device
sn9c20x	0c45:628f			PC Camera (SN9C202 + OV9650)
sn9c20x	0c45:62a0			PC Camera (SN9C202 + OV7670)
sn9c20x	0c45:62b0			PC Camera (SN9C202 + MT9V011/MT9V111/MT9V112)
sn9c20x	0c45:62b3			PC Camera (SN9C202 + OV9655)
sn9c20x	0c45:62bb			PC Camera (SN9C202 + OV7660)
sn9c20x	0c45:62bc			PC Camera (SN9C202 + HV7131R)
sn9c2028	0c45:8001			Wild Planet Digital Spy Camera
sn9c2028	0c45:8003			Sakar #11199, #6637x, #67480 keychain cams
sn9c2028	0c45:8008			Mini-Shotz ms-350
sn9c2028	0c45:800a			Vivitar Vivicam 3350B
sunplus	0d64:0303			Sunplus FashionCam DXG
ov519	0e96:c001			TRUST 380 USB2 SPACEC@M
etoms	102c:6151			Qcam Sangha CIF
etoms	102c:6251			Qcam xxxxxx VGA
ov519	1046:9967			W9967CF/W9968CF WebCam IC, Video Blaster WebCam Go
zc3xx	10fd:0128			Typhoon Webshot II USB 300k 0x0128
spca561	10fd:7e50			FlyCam Usb 100
zc3xx	10fd:804d			Typhoon Webshot II Webcam [zc0301]
zc3xx	10fd:8050			Typhoon Webshot II USB 300k
ov534	1415:2000			Sony HD Eye for PS3 (SLEH 00201)
pac207	145f:013a			Trust WB-1300N
pac7302	145f:013c			Trust
sn9c20x	145f:013d			Trust WB-3600R
vc032x	15b8:6001			HP 2.0 Megapixel
vc032x	15b8:6002			HP 2.0 Megapixel rz406aa
stk1135	174f:6a31			ASUSlaptop, MT9M112 sensor
spca501	1776:501c			Arowana 300K CMOS Camera
t613	17a1:0128			TASCORP JPEG Webcam, NGS Cyclops
vc032x	17ef:4802			Lenovo Vc0323+MI1310_SOC
pac7302	1ae7:2001			SpeedLinkSnappy Mic SL-6825-SBK
pac207	2001:f115			D-Link DSB-C120
sq905c	2770:9050			Disney pix micro (CIF)
sq905c	2770:9051			Lego Bionicle
sq905c	2770:9052			Disney pix micro 2 (VGA)
sq905c	2770:905c			All 11 known cameras with this ID
sq905	2770:9120			All 24 known cameras with this ID
sq905c	2770:913d			All 4 known cameras with this ID
sq930x	2770:930b			Sweex Motion Tracking / I-Tec iCam Tracer
sq930x	2770:930c			Trust WB-3500T / NSG Robbie 2.0
spca500	2899:012c			Toppro Industrial
ov519	8020:ef04			ov519
spca508	8086:0110			Intel Easy PC Camera
spca500	8086:0630			Intel Pocket PC Camera
spca506	99fa:8988			Grandtec V.cap
sn9c20x	a168:0610			Dino-Lite Digital Microscope (SN9C201 + HV7131R)
sn9c20x	a168:0611			Dino-Lite Digital Microscope (SN9C201 + HV7131R)
sn9c20x	a168:0613			Dino-Lite Digital Microscope (SN9C201 + HV7131R)

Continued on next page

Table 4 - continued from previous page

	driver	vend:prod	Device
sn9c20x	a168:0614		Dino-Lite Digital Microscope (SN9C201 + MT9M111)
sn9c20x	a168:0615		Dino-Lite Digital Microscope (SN9C201 + MT9M111)
sn9c20x	a168:0617		Dino-Lite Digital Microscope (SN9C201 + MT9M111)
sn9c20x	a168:0618		Dino-Lite Digital Microscope (SN9C201 + HV7131R)
spca561	abcd:cdee		Petcam

**dvb-usb-dib0700 cards list**

Card name	USB IDs
ASUS My Cinema U3000 Mini DVBT Tuner	0b05:171f
ASUS My Cinema U3100 Mini DVBT Tuner	0b05:173f
AVerMedia AVerTV DVB-T Express	07ca:b568
AVerMedia AVerTV DVB-T Volar	07ca:a807, 07ca:b808
Artec T14BR DVB-T	05d8:810f
Asus My Cinema-U3000Hybrid	0b05:1736
Compro Videomate U500	185b:1e78, 185b:1e80
DiBcom NIM7090 reference design	10b8:1bb2
DiBcom NIM8096MD reference design	10b8:1fa8
DiBcom NIM9090MD reference design	10b8:2384
DiBcom STK7070P reference design	10b8:1ebc
DiBcom STK7070PD reference design	10b8:1ebe
DiBcom STK7700D reference design	10b8:1ef0
DiBcom STK7700P reference design	10b8:1e14, 10b8:1e78
DiBcom STK7770P reference design	10b8:1e80
DiBcom STK807xP reference design	10b8:1f90
DiBcom STK807xPVR reference design	10b8:1f98
DiBcom STK8096-PVR reference design	2013:1faa, 10b8:1faa
DiBcom STK8096GP reference design	10b8:1fa0
DiBcom STK9090M reference design	10b8:2383
DiBcom TFE7090PVR reference design	10b8:1bb4
DiBcom TFE7790P reference design	10b8:1e6e
DiBcom TFE8096P reference design	10b8:1f9C
Elgato EyeTV DTT	0fd9:0021
Elgato EyeTV DTT rev. 2	0fd9:003f

Continued on next page

Table 5 - continued from previous page

Card name	USB IDs
Elgato EyeTV Diversity	0fd9:0011
Elgato EyeTV Dtt Dlx PD378S	0fd9:0020
EvolutePC TVWay+	1e59:0002
Gigabyte U7000	1044:7001
Gigabyte U8000-RH	1044:7002
Hama DVB=T Hybrid USB Stick	147f:2758
Hauppauge ATSC MiniCard (B200)	2040:b200
Hauppauge ATSC MiniCard (B210)	2040:b210
Hauppauge Nova-T 500 Dual DVB-T	2040:9941, 2040:9950
Hauppauge Nova-T MyTV.t	2040:7080
Hauppauge Nova-T Stick	2040:7050, 2040:7060, 2040:7070
Hauppauge Nova-TD Stick (52009)	2040:5200
Hauppauge Nova-TD Stick/Elgato Eye-TV Diversity	2040:9580
Hauppauge Nova-TD-500 (84xxx)	2040:8400
Leadtek WinFast DTV Dongle H	0413:60f6
Leadtek Winfast DTV Dongle (STK7700P based)	0413:6f00, 0413:6f01
Medion CTX1921 DVB-T USB	1660:1921
Microsoft Xbox One Digital TV Tuner	045e:02d5
PCTV 2002e	2013:025c
PCTV 2002e SE	2013:025d
Pinnacle Expresscard 320cx	2304:022e
Pinnacle PCTV 2000e	2304:022c
Pinnacle PCTV 282e	2013:0248, 2304:0248
Pinnacle PCTV 340e HD Pro USB Stick	2304:023d
Pinnacle PCTV 72e	2304:0236
Pinnacle PCTV 73A	2304:0243
Pinnacle PCTV 73e	2304:0237
Pinnacle PCTV 73e SE	2013:0245, 2304:0245
Pinnacle PCTV DVB-T Flash Stick	2304:0228
Pinnacle PCTV Dual DVB-T Diversity Stick	2304:0229
Pinnacle PCTV HD Pro USB Stick	2304:023a
Pinnacle PCTV HD USB Stick	2304:023b
Pinnacle PCTV Hybrid Stick Solo	2304:023e
Prolink Pixelview SBTVD	1554:5010
Sony PlayTV	1415:0003
TechniSat AirStar TeleStick 2	14f7:0004
Terratec Cinergy DT USB XS Diversity/ T5	0ccd:0081, 0ccd:10a1
Terratec Cinergy DT XS Diversity	0ccd:005a
Terratec Cinergy HT Express	0ccd:0060
Terratec Cinergy HT USB XE	0ccd:0058
Terratec Cinergy T Express	0ccd:0062

Continued on next page

Table 5 - continued from previous page

Card name	USB IDs
Terratec Cinergy T USB XXS (HD)/ T3	0ccd:0078, 0ccd:10a0, 0ccd:00ab
Uniwill STK7700P based (Hama and others)	1584:6003
YUAN High-Tech DiBcom STK7700D	1164:1e8c
YUAN High-Tech MC770	1164:0871
YUAN High-Tech STK7700D	1164:1efc
YUAN High-Tech STK7700PH	1164:1f08
Yuan EC372S	1164:1edc
Yuan PD378S	1164:2edc

**dvb-usb-dibusb-mb cards list**

Card name	USB IDs
AVerMedia AverTV DVBT USB1.1	14aa:0001, 14aa:0002
Artec T1 USB1.1 TVBOX with AN2135	05d8:8105, 05d8:8106
Artec T1 USB1.1 TVBOX with AN2235	05d8:8107, 05d8:8108
Artec T1 USB1.1 TVBOX with AN2235 (faulty USB IDs)	0547:2235
Artec T1 USB2.0	05d8:8109, 05d8:810a
Compro Videomate DVB-U2000 - DVB-T USB1.1 (please confirm to linux-dvb)	185b:d000, 145f:010c, 185b:d001
DiBcom USB1.1 DVB-T reference design (MOD3000)	10b8:0bb8, 10b8:0bb9
Grandtec USB1.1 DVB-T	5032:0fa0, 5032:0bb8, 5032:0fa1, 5032:0bb9
KWorld V-Stream XPERT DTV - DVB-T USB1.1	eb1a:17de, eb1a:17df
KWorld Xpert DVB-T USB2.0	eb2a:17de
KWorld/ADSTech Instant DVB-T USB2.0	06e1:a333, 06e1:a334
TwinhanDTV USB-Ter USB1.1 / Magic Box I / HAMA USB1.1 DVB-T device	13d3:3201, 1822:3201, 13d3:3202, 1822:3202
Unknown USB1.1 DVB-T device ???? please report the name to the author	1025:005e, 1025:005f
VideoWalker DVB-T USB	0458:701e, 0458:701f

**dvb-usb-dibusb-mc cards list**

Card name	USB IDs
Artec T1 USB2.0 TVBOX (please check the warm ID)	05d8:8109, 05d8:810a
Artec T14 - USB2.0 DVB-T	05d8:810b, 05d8:810c
DiBcom USB2.0 DVB-T reference design (MOD3000P)	10b8:0bc6, 10b8:0bc7
GRAND - USB2.0 DVB-T adapter	5032:0bc6, 5032:0bc7
Humax/Coex DVB-T USB Stick 2.0 High Speed	10b9:5000, 10b9:5001
LITE-ON USB2.0 DVB-T Tuner	04ca:f000, 04ca:f001
Leadtek - USB2.0 Winfast DTV dongle	0413:6025, 0413:6026
MSI Digivox Mini SL	eb1a:e360, eb1a:e361

**dvb-usb-a800 cards list**

Card name	USB IDs
AVerMedia AverTV DVB-T USB 2.0 (A800)	07ca:a800, 07ca:a801

**dvb-usb-af9005 cards list**

Card name	USB IDs
Afatech DVB-T USB1.1 stick	15a4:9020
Ansonic DVB-T USB1.1 stick	10b9:6000
TerraTec Cinergy T USB XE	0ccd:0055

**dvb-usb-az6027 cards list**

Card name	USB IDs
AZUREWAVE DVB-S/S2 USB2.0 (AZ6027)	13d3:3275
Elgato EyeTV Sat	0fd9:002a, 0fd9:0025, 0fd9:0036
TERRATEC S7	0ccd:10a4
TERRATEC S7 MKII	0ccd:10ac
Technisat SkyStar USB 2 HD CI	14f7:0001, 14f7:0002

**dvb-usb-cinergyT2 cards list**

Card name	USB IDs
TerraTec/qanu USB2.0 Highspeed DVB-T Receiver	0ccd:0x0038

**dvb-usb-cxusb cards list**

Card name	USB IDs
AVerMedia AVerTVHD Volar (A868R)	
Conexant DMB-TH Stick	
DViCO FusionHDTV DVB-T Dual Digital 2	
DViCO FusionHDTV DVB-T Dual Digital 4	
DViCO FusionHDTV DVB-T Dual Digital 4 (rev 2)	
DViCO FusionHDTV DVB-T Dual USB	
DViCO FusionHDTV DVB-T NANO2	
DViCO FusionHDTV DVB-T USB (LGZ201)	
DViCO FusionHDTV DVB-T USB (TH7579)	
DViCO FusionHDTV5 USB Gold	
DigitalNow DVB-T Dual USB	
Medion MD95700 (MDUSBTV-HYBRID)	
Mygica D689 DMB-TH	

**dvb-usb-digitv cards list**

Card name	USB IDs
Nebula Electronics uDigiTV DVB-T USB2.0)	0547:0201

**dvb-usb-dtt200u cards list**

Card name	USB IDs
WideView WT-220U PenType Receiver (Miglia)	18f3:0220
WideView WT-220U PenType Receiver (Typhoon/Freecom)	14aa:0222, 14aa:0220, 14aa:0221, 14aa:0225, 14aa:0226
WideView WT-220U PenType Receiver (based on ZL353)	14aa:022a, 14aa:022b
WideView/Yuan/Yakumo/Hama/Typhoon DVB-T USB2.0 (WT-200U)	14aa:0201, 14aa:0301

**dvb-usb-dtv5100 cards list**

Card name	USB IDs
AME DTV-5100 USB2.0 DVB-T	0x06be:0xa232

**dvb-usb-dw2102 cards list**

Card name	USB IDs
DVBWorld DVB-C 3101 USB2.0	04b4:3101
DVBWorld DVB-S 2101 USB2.0	04b4:0x2101
DVBWorld DVB-S 2102 USB2.0	04b4:2102
DVBWorld DW2104 USB2.0	04b4:2104
GOTVIEW Satellite HD	0x1fE1:5456
Geniatech T220 DVB-T/T2 USB2.0	0x1f4d:0xD220
SU3000HD DVB-S USB2.0	0x1f4d:0x3000
TeVii S482 (tuner 1)	0x9022:0xd483
TeVii S482 (tuner 2)	0x9022:0xd484
TeVii S630 USB	0x9022:d630
TeVii S650 USB2.0	0x9022:d650
TeVii S662	0x9022:d662
TechnoTrend TT-connect S2-4600	0b48:3011
TerraTec Cinergy S USB	0ccd:0064
Terratec Cinergy S2 USB BOX	0ccd:0x0105
Terratec Cinergy S2 USB HD	0ccd:00a8
Terratec Cinergy S2 USB HD Rev.2	0ccd:00b0
Terratec Cinergy S2 USB HD Rev.3	0ccd:0102
X3M TV SPC1400HD PCI	0x1f4d:0x3100

**dvb-usb-gp8psk cards list**

Card name	USB IDs
Genpix 8PSK-to-USB2 Rev.1 DVB-S receiver	09c0:0200, 09c0:0201
Genpix 8PSK-to-USB2 Rev.2 DVB-S receiver	09c0:0202
Genpix SkyWalker-1 DVB-S receiver	09c0:0203
Genpix SkyWalker-2 DVB-S receiver	09c0:0206

**dvb-usb-m920x cards list**

Card name	USB IDs
DTV-DVB UDTT7049	13d3:3219
Dposh DVB-T USB2.0	1498:9206, 1498:a090
LifeView TV Walker Twin DVB-T USB2.0	10fd:0514, 10fd:0513
MSI DIGI VOX mini II DVB-T USB2.0	10fd:1513
MSI Mega Sky 580 DVB-T USB2.0	0db0:5580
Pinnacle PCTV 310e	13d3:3211

**dvb-usb-nova-t-usb2 cards list**

Card name	USB IDs
Hauppauge WinTV-NOVA-T usb2	2040:9300, 2040:9301

**dvb-usb-opera1 cards list**

Card name	USB IDs
Opera1 DVB-S USB2.0	04b4:2830, 695c:3829

**dvb-usb-pctv452e cards list**

Card name	USB IDs
PCTV HDTV USB	2304:021f
Technotrend TT Connect S2-3600	0b48:3007
Technotrend TT Connect S2-3650-CI	0b48:300a

**dvb-usb-technisat-usb2 cards list**

Card name	USB IDs
Technisat SkyStar USB HD (DVB-S/S2)	14f7:0500

**dvb-usb-ttusb2 cards list**

Card name	USB IDs
Pinnacle 400e DVB-S USB2.0	2304:020f
Pinnacle 450e DVB-S USB2.0	2304:0222
Technotrend TT-connect CT-3650	0b48:300d
Technotrend TT-connect S-2400	0b48:3006
Technotrend TT-connect S-2400 (8kB EEPROM)	0b48:3009

**dvb-usb-umt-010 cards list**

Card name	USB IDs
Hanftek UMT-010 DVB-T USB2.0	15f4:0001, 15f4:0015

**dvb-usb-vp702x cards list**

Card name	USB IDs
TwinhanDTV StarBox DVB-S USB2.0 (VP7021)	13d3:3207

**dvb-usb-vp7045 cards list**

Card name	USB IDs
DigitalNow TinyUSB 2 DVB-t Receiver	13d3:3223, 13d3:3224
Twinhan USB2.0 DVB-T receiver (TwinhanDTV Alpha/MagicBox II)	13d3:3205, 13d3:3206

## dvb-usb-af9015 cards list

Card name	USB IDs
AVerMedia A309	07ca:a309
AVerMedia AVerTV DVB-T Volar X	07ca:a815
Afatech AF9015 reference design	15a4:9015, 15a4:9016
AverMedia AVerTV Red HD+ (A850T)	07ca:850b
AverMedia AVerTV Volar Black HD (A850)	07ca:850a
AverMedia AVerTV Volar GPS 805 (A805)	07ca:a805
AverMedia AVerTV Volar M (A815Mac)	07ca:815a
Conceptronic USB2.0 DVB-T CTVDIGRCU V3.0	1b80:e397
DigitalNow TinyTwin	13d3:3226
DigitalNow TinyTwin v2	1b80:e402
DigitalNow TinyTwin v3	1f4d:9016
Fujitsu-Siemens Slim Mobile USB DVB-T	07ca:8150
Genius TVGo DVB-T03	0458:4012
KWorld Digital MC-810	1b80:c810
KWorld PlusTV DVB-T PCI Pro Card (DVB-T PC160-T)	1b80:c161
KWorld PlusTV Dual DVB-T PCI (DVB-T PC160-2T)	1b80:c160
KWorld PlusTV Dual DVB-T Stick (DVB-T 399U)	1b80:e399, 1b80:e400
KWorld USB DVB-T Stick Mobile (UB383-T)	1b80:e383
KWorld USB DVB-T TV Stick II (VS-DVB-T 395U)	1b80:e396, 1b80:e39b, 1b80:e395, 1b80:e39a
Leadtek WinFast DTV Dongle Gold	0413:6029
Leadtek WinFast DTV2000DS	0413:6a04
MSI DIGIVOX Duo	1462:8801
MSI Digi VOX mini III	1462:8807
Pinnacle PCTV 71e	2304:022b
Sveon STV20 Tuner USB DVB-T HDTV	1b80:e39d
Sveon STV22 Dual USB DVB-T Tuner HDTV	1b80:e401
Telestar Starstick 2	10b9:8000
TerraTec Cinergy T Stick Dual RC	0ccd:0099
TerraTec Cinergy T Stick RC	0ccd:0097
TerraTec Cinergy T USB XE	0ccd:0069
TrekStor DVB-T USB Stick	15a4:901b

Continued on next page

Table 6 - continued from previous page

Card name	USB IDs
TwinHan AzureWave AD-TU700(704J)	13d3:3237
Xtensions XD-380	1ae7:0381

**dvb-usb-af9035 cards list**

Card name	USB IDs
AVerMedia AVerTV Volar HD/PRO (A835)	07ca:a835, 07ca:b835
AVerMedia HD Volar (A867)	07ca:1867, 07ca:a867, 07ca:0337
AVerMedia TD310 DVB-T2	07ca:1871
AVerMedia Twinstar (A825)	07ca:0825
Afatech AF9035 reference design	15a4:9035, 15a4:1000, 15a4:1001, 15a4:1002, 15a4:1003
Asus U3100Mini Plus	0b05:1779
Avermedia A835B(1835)	07ca:1835
Avermedia A835B(2835)	07ca:2835
Avermedia A835B(3835)	07ca:3835
Avermedia A835B(4835)	07ca:4835
Avermedia AVerTV Volar HD 2 (TD110)	07ca:a110
Avermedia H335	07ca:0335
Digital Dual TV Receiver CTVDIG-DUAL_V2	1b80:e410
EVOLVEO XtraTV stick	1f4d:a115
Hauppauge WinTV-MiniStick 2	2040:f900
ITE 9135 Generic	048d:9135
ITE 9135(9005) Generic	048d:9005
ITE 9135(9006) Generic	048d:9006
ITE 9303 Generic	048d:9306
Kworld UB499-2T T09	1b80:e409
Leadtek WinFast DTV Dongle Dual	0413:6a05
Logilink VG0022A	1d19:0100
PCTV AndroiDTV (78e)	2013:025a
PCTV microStick (79e)	2013:0262
Sveon STV22 Dual DVB-T HDTV	1b80:e411
TerraTec Cinergy T Stick	0ccd:0093
TerraTec Cinergy T Stick (rev. 2)	0ccd:00aa
TerraTec Cinergy T Stick Dual RC (rev. 2)	0ccd:0099
TerraTec Cinergy TC2 Stick	0ccd:10b2
TerraTec T1	0ccd:10ae

### dvb-usb-anysee cards list

Card name	USB IDs
Anysee	04b4:861f, 1c73:861f

### dvb-usb-au6610 cards list

Card name	USB IDs
Sigmathek DVB-110	058f:6610

### dvb-usb-az6007 cards list

Card name	USB IDs
Azurewave 6007	13d3:0ccd
Technisat CableStar Combo HD CI	14f7:0003
Terratec H7	0ccd:10b4, 0ccd:10a3

### dvb-usb-ce6230 cards list

Card name	USB IDs
AVerMedia A310 USB 2.0 DVB-T tuner	07ca:a310
Intel CE9500 reference design	8086:9500

**dvb-usb-dvbsky cards list**

Card name	USB IDs
DVBSky S960/S860	0572:6831
DVBSky S960CI	0572:960c
DVBSky T330	0572:0320
DVBSky T680CI	0572:680c
MyGica Mini DVB-T2 USB Stick T230	0572:c688
MyGica Mini DVB-T2 USB Stick T230C	0572:c689
MyGica Mini DVB-T2 USB Stick T230C Lite	0572:c699
MyGica Mini DVB-T2 USB Stick T230C v2	0572:c68a
TechnoTrend TT-connect CT2-4650 CI	0b48:3012
TechnoTrend TT-connect CT2-4650 CI v1.1	0b48:3015
TechnoTrend TT-connect S2-4650 CI	0b48:3017
TechnoTrend TVStick CT2-4400	0b48:3014
Terratec Cinergy S2 Rev.4	0ccd:0105
Terratec H7 Rev.4	0ccd:10a5

**dvb-usb-ec168 cards list**

Card name	USB IDs
E3C EC168 reference design	18b4:1689, 18b4:fffa, 18b4:fffb, 18b4:1001, 18b4:1002

**dvb-usb-gl861 cards list**

Card name	USB IDs
774 Friio White ISDB-T USB2.0	7a69:0001
A-LINK DTU DVB-T USB2.0	05e3:f170
MSI Mega Sky 55801 DVB-T USB2.0	0db0:5581

**dvb-usb-lmedm04 cards list**

Card name	USB IDs
DM04_LME2510C_DVB-S	3344:1120
DM04_LME2510C_DVB-S RS2000	3344:22f0
DM04_LME2510_DVB-S	3344:1122

**dvb-usb-mxl111sf cards list**

Card name	USB IDs
HCW 117xxx	2040:b702
HCW 126xxx	2040:c602, 2040:c60a
Hauppauge 117xxx ATSC+	2040:b700, 2040:b703, 2040:b753, 2040:b763, 2040:b757, 2040:b767
Hauppauge 117xxx DVBT	2040:b704, 2040:b764
Hauppauge 126xxx	2040:c612, 2040:c61a
Hauppauge 126xxx ATSC	2040:c601, 2040:c609, 2040:b701
Hauppauge 126xxx ATSC+	2040:c600, 2040:c603, 2040:c60b, 2040:c653, 2040:c65b
Hauppauge 126xxx DVBT	2040:c604, 2040:c60c
Hauppauge 138xxx DVBT	2040:d854, 2040:d864, 2040:d8d4, 2040:d8e4
Hauppauge Mercury	2040:d853, 2040:d863, 2040:d8d3, 2040:d8e3, 2040:d8ff
Hauppauge WinTV-Aero-M	2040:c613, 2040:c61b

**dvb-usb-rtl28xxu cards list**

Card name	USB IDs
ASUS My Cinema-U3100Mini Plus V2	1b80:d3a8
Astrometa DVB-T2	15f4:0131
Compro VideoMate U620F	185b:0620
Compro VideoMate U650F	185b:0650
Crypto ReDi PC 50 A	1f4d:a803
Dexatek DK DVB-T Dongle	1d19:1101
Dexatek DK mini DVB-T Dongle	1d19:1102
DigitalNow Quad DVB-T Receiver	0413:6680
Freecom USB2.0 DVB-T	14aa:0160, 14aa:0161
G-Tek Electronics Group Lifeview LV5TDLX DVB-T	1f4d:b803
GIGABYTE U7300	1b80:d393
Genius TVGo DVB-T03	0458:707f
GoTView MasterHD 3	5654:ca42
Leadtek WinFast DTV Dongle mini	0413:6a03

Continued on next page

Table 8 - continued from previous page

Card name	USB IDs
Leadtek WinFast DTV2000DS Plus	0413:6f12
Leadtek Winfast DTV Dongle Mini D	0413:6f0f
MSI DIGIVOX Micro HD	1d19:1104
MaxMedia HU394-T	1b80:d394
PROlectrix DV107669	1f4d:d803
Peak DVB-T USB	1b80:d395
Realtek RTL2831U reference design	0bda:2831
Realtek RTL2832U reference design	0bda:2832, 0bda:2838
Sveon STV20	1b80:d39d
Sveon STV21	1b80:d3b0
Sveon STV27	1b80:d3af
TURBO-X Pure TV Tuner DTT-2000	1b80:d3a4
TerraTec Cinergy T Stick Black	0ccd:00a9
TerraTec Cinergy T Stick RC (Rev. 3)	0ccd:00d3
TerraTec Cinergy T Stick+	0ccd:00d7
TerraTec NOXON DAB Stick	0ccd:00b3
TerraTec NOXON DAB Stick (rev 2)	0ccd:00e0
TerraTec NOXON DAB Stick (rev 3)	0ccd:00b4
Trekstor DVB-T Stick Terres 2.0	1f4d:C803

### dvb-usb-zd1301 cards list

Card name	USB IDs
ZyDAS ZD1301 reference design	0ace:13a1

### Other USB cards list

	Driver	Card name	USB IDs
airspy	Airspy		1d50:60a1
dvb-as102	Abilis Systems DVB-Titan		1BA6:0001
dvb-as102	PCTV Systems picoStick (74e)		2013:0246
dvb-as102	Elgato EyeTV DTT Deluxe		0fd9:002c
dvb-as102	nBox DVB-T Dongle		0b89:0007
dvb-as102	Sky IT Digital Key (green led)		2137:0001
b2c2-flexcop-usb	Technisat/B2C2 FlexCop II/IIB/III Digital TV		0af7:0101
cpia2	Vision' s CPiA2 cameras such as the Digital Blue QX5		0553:0100, 0553:
go7007	WIS GO7007 MPEG encoder		1943:a250, 093b:
hackrf	HackRF Software Decoder Radio		1d50:6089
hdpvr	Hauppauge HD PVR		2040:4900, 2040:
msi2500	Mirics MSi3101 SDR Dongle		1df7:2500, 2040:c

Continued on next page

Table 9 - continued from previous page

	Driver	Card name	USB IDs
pvrusb2		Hauppauge WinTV-PVR USB2	2040:2900, 2040:
pwc		Creative Webcam 5	041E:400C
pwc		Creative Webcam Pro Ex	041E:4011
pwc		Logitech QuickCam 3000 Pro	046D:08B0
pwc		Logitech QuickCam Notebook Pro	046D:08B1
pwc		Logitech QuickCam 4000 Pro	046D:08B2
pwc		Logitech QuickCam Zoom (old model)	046D:08B3
pwc		Logitech QuickCam Zoom (new model)	046D:08B4
pwc		Logitech QuickCam Orbit/Sphere	046D:08B5
pwc		Logitech/Cisco VT Camera	046D:08B6
pwc		Logitech ViewPort AV 100	046D:08B7
pwc		Logitech QuickCam	046D:08B8
pwc		Philips PCA645VC	0471:0302
pwc		Philips PCA646VC	0471:0303
pwc		Askey VC010 type 2	0471:0304
pwc		Philips PCVC675K (Vesta)	0471:0307
pwc		Philips PCVC680K (Vesta Pro)	0471:0308
pwc		Philips PCVC690K (Vesta Pro Scan)	0471:030C
pwc		Philips PCVC730K (ToUCam Fun), PCVC830 (ToUCam II)	0471:0310
pwc		Philips PCVC740K (ToUCam Pro), PCVC840 (ToUCam II)	0471:0311
pwc		Philips PCVC750K (ToUCam Pro Scan)	0471:0312
pwc		Philips PCVC720K/40 (ToUCam XS)	0471:0313
pwc		Philips SPC 900NC	0471:0329
pwc		Philips SPC 880NC	0471:032C
pwc		Sotec Afina Eye	04CC:8116
pwc		Samsung MPC-C10	055D:9000
pwc		Samsung MPC-C30	055D:9001
pwc		Samsung SNC-35E (Ver3.0)	055D:9002
pwc		Askey VC010 type 1	069A:0001
pwc		AME Co. Afina Eye	06BE:8116
pwc		Visionite VCS-UC300	0d81:1900
pwc		Visionite VCS-UM100	0d81:1910
s2255drv		Sensoray 2255	1943:2255, 19
stk1160		STK1160 USB video capture dongle	05e1:0408
stkwebcam		Syntek DC1125	174f:a311, 05e
dvb-ttusb-budget		Technotrend/Hauppauge Nova-USB devices	0b48:1003, 0b
dvb-ttusb_dec		Technotrend/Hauppauge MPEG decoder DEC3000-s	0b48:1006
dvb-ttusb_dec		Technotrend/Hauppauge MPEG decoder	0b48:1007
dvb-ttusb_dec		Technotrend/Hauppauge MPEG decoder DEC2000-t	0b48:1008
dvb-ttusb_dec		Technotrend/Hauppauge MPEG decoder DEC2540-t	0b48:1009
usbtv		Fushicai USBTV007 Audio-Video Grabber	1b71:3002, 1f
zr364xx		USB ZR364XX Camera	08ca:0109, 04

## PCI drivers

The PCI boards are identified by an identification called PCI ID. The PCI ID is actually composed by two parts:

- Vendor ID and device ID;
- Subsystem ID and Subsystem device ID;

The `lspci -nn` command allows identifying the vendor/device PCI IDs:

```
$ lspci -nn
...
00:0a.0 Multimedia controller [0480]: Philips Semiconductors SAA7131/
↳SAA7133/SAA7135 Video Broadcast Decoder [1131:7133] (rev d1)
00:0b.0 Multimedia controller [0480]: Brooktree Corporation Bt878 Audio
↳Capture [109e:0878] (rev 11)
01:00.0 Multimedia video controller [0400]: Conexant Systems, Inc.
↳CX23887/8 PCIe Broadcast Audio and Video Decoder with 3D Comb
↳[14f1:8880] (rev 0f)
02:01.0 Multimedia video controller [0400]: Internext Compression Inc.
↳iTVC15 (CX23415) Video Decoder [4444:0803] (rev 01)
02:02.0 Multimedia video controller [0400]: Conexant Systems, Inc.
↳CX23418 Single-Chip MPEG-2 Encoder with Integrated Analog Video/
↳Broadcast Audio Decoder [14f1:5b7a]
02:03.0 Multimedia video controller [0400]: Brooktree Corporation Bt878
↳Video Capture [109e:036e] (rev 11)
...
```

The subsystem IDs can be obtained using `lspci -vn`

```
$ lspci -vn
...
00:0a.0 0480: 1131:7133 (rev d1)
Subsystem: 1461:f01d
Flags: bus master, medium devsel, latency 32, IRQ 209
Memory at e2002000 (32-bit, non-prefetchable) [size=2K]
Capabilities: [40] Power Management version 2
...
```

At the above example, the first card uses the `saa7134` driver, and has a vendor/device PCI ID equal to `1131:7133` and a PCI subsystem ID equal to `1461:f01d` (see `Saa7134` card list).

Unfortunately, sometimes the same PCI subsystem ID is used by different products. So, several media drivers allow passing a `card=` parameter, in order to setup a card number that would match the correct settings for an specific board.

The current supported PCI/PCIe cards (not including staging drivers) are listed below<sup>1</sup>.

	Driver	Name
	<code>altera-ci</code>	Altera FPGA based CI module
	<code>b2c2-flexcop-pci</code>	Technisat/B2C2 Air/Sky/Cable2PC PCI

Continued on next page

<sup>1</sup> some of the drivers have sub-drivers, not shown at this table

Table 10 - continued from previous page

	Driver	Name
bt878	DVB/ATSC Support for bt878 based TV cards	
bttv	BT8x8 Video For Linux	
cobalt	Cisco Cobalt	
cx18	Conexant cx23418 MPEG encoder	
cx23885	Conexant cx23885 (2388x successor)	
cx25821	Conexant cx25821	
cx88xx	Conexant 2388x (bt878 successor)	
ddbbridge	Digital Devices bridge	
dm1105	SDMC DM1105 based PCI cards	
dt3155	DT3155 frame grabber	
dvb-ttpci	AV7110 cards	
earth-pt1	PT1 cards	
earth-pt3	Earthsoft PT3 cards	
hexium_gemini	Hexium Gemini frame grabber	
hexium_orion	Hexium HV-PCI6 and Orion frame grabber	
hopper	HOPPER based cards	
ipu3-cio2	Intel ipu3-cio2 driver	
ivtv	Conexant cx23416/cx23415 MPEG encoder/decoder	
ivtvfb	Conexant cx23415 framebuffer	
mantis	MANTIS based cards	
meye	Sony Vaio Picturebook Motion Eye	
mxb	Siemens-Nixdorf 'Multimedia eXtension Board'	
netup-unidvb	NetUP Universal DVB card	
ngene	Micronas nGene	
pluto2	Pluto2 cards	
saa7134	Philips SAA7134	
saa7164	NXP SAA7164	
smipcie	SMI PCIe DVBSky cards	
solo6x10	Bluecherry / Softlogic 6x10 capture cards (MPEG-4/H.264)	
sta2x11_vip	STA2X11 VIP Video For Linux	
tw5864	Techwell TW5864 video/audio grabber and encoder	
tw686x	Intersil/Techwell TW686x	
tw68	Techwell tw68x Video For Linux	

Some of those drivers support multiple devices, as shown at the card lists below:

### BTTV cards list

Card number	Card name	PCI subsystem IDs
0	* UNKNOWN/GENERIC *	
1	MIRO PCTV	
2	Hauppauge (bt848)	
3	STB, Gateway P/N 6000699 (bt848)	

Continued on next page

Table 11 - continued from previous page

Card number	Card name	PCI subsystem IDs
4	Intel Create and Share PCI/ Smart Video Recorder III	
5	Diamond DTV2000	
6	AVerMedia TVPhone	
7	MATRIX-Vision MV-Delta	
8	Lifeview FlyVideo II (Bt848) LR26 / MAXI TV Video PCI2 LR26	
9	IMS/IXmicro TurboTV	
10	Hauppauge (bt878)	0070:13eb, 0070:3900, 2636:10b4
11	MIRO PCTV pro	
12	ADS Technologies Channel Surfer TV (bt848)	
13	AVerMedia TVCapture 98	1461:0002, 1461:0004, 1461:0300
14	Aimslab Video Highway Xtreme (VHX)	
15	Zoltrix TV-Max	a1a0:a0fc
16	Prolink Pixelview PlayTV (bt878)	
17	Leadtek WinView 601	
18	AVEC Intercapture	
19	Lifeview FlyVideo II EZ /FlyKit LR38 Bt848 (capture only)	
20	CEI Raffles Card	
21	Lifeview FlyVideo 98/ Lucky Star Image World ConferenceTV LR50	
22	Askey CPH050/ Phoebe Tv Master + FM	14ff:3002
23	Modular Technology MM201/MM202/MM205/MM210/MM215 PCTV, bt878	1457:0101
24	Askey CPH05X/06X (bt878) [many vendors]	144f:3002, 144f:3005, 144f:5000, 14ff:3000
25	Terratec TerraTV+ Version 1.0 (Bt848)/ Terra TValue Version 1.0/ Vobis TV-Boostar	
26	Hauppauge WinCam newer (bt878)	
27	Lifeview FlyVideo 98/ MAXI TV Video PCI2 LR50	
28	Terratec TerraTV+ Version 1.1 (bt878)	153b:1127, 1852:1852
29	Imagenation PXC200	1295:200a
30	Lifeview FlyVideo 98 LR50	1f7f:1850
31	Formac iProTV, Formac ProTV I (bt848)	
32	Intel Create and Share PCI/ Smart Video Recorder III	

Continued on next page

Table 11 - continued from previous page

Card number	Card name	PCI subsystem IDs
33	Terratec TerraTValue Version Bt878	153b:1117, 153b:1118, 153b:1119, 153b:111a, 153b:1134, 153b:5018
34	Leadtek WinFast 2000/ WinFast 2000 XP	107d:6606, 107d:6609, 6606:217d, f6ff:fff6
35	Liferview FlyVideo 98 LR50 / Chronos Video Shuttle II	1851:1850, 1851:a050
36	Liferview FlyVideo 98FM LR50 / Typhoon TView TV/FM Tuner	1852:1852
37	Prolink PixelView PlayTV pro	
38	Askey CPH06X TView99	144f:3000, 144f:a005, a04f:a0fc
39	Pinnacle PCTV Studio/Rave	11bd:0012, bd11:1200, bd11:ff00, 11bd:ff12
40	STB TV PCI FM, Gateway P/N 6000704 (bt878), 3Dfx VoodooTV 100	10b4:2636, 10b4:2645, 121a:3060
41	AVerMedia TVPhone 98	1461:0001, 1461:0003
42	ProVideo PV951	aa0c:146c
43	Little OnAir TV	
44	Sigma TVII-FM	
45	MATRIX-Vision MV-Delta 2	
46	Zoltrix Genie TV/FM	15b0:4000, 15b0:400a, 15b0:400d, 15b0:4010, 15b0:4016
47	Terratec TV/Radio+	153b:1123
48	Askey CPH03x/ Dynalink Magic TView	
49	IODATA GV-BCTV3/PCI	10fc:4020
50	Prolink PV-BT878P+4E / PixelView PlayTV PAK / Lenco MXTV-9578 CP	
51	Eagle Wireless Capricorn2 (bt878A)	
52	Pinnacle PCTV Studio Pro	
53	Typhoon TView RDS + FM Stereo / KNC1 TV Station RDS	
54	Liferview FlyVideo 2000 /FlyVideo A2/ Lifetec LT 9415 TV [LR90]	
55	Askey CPH031/ BESTBUY Easy TV	
56	Liferview FlyVideo 98FM LR50	a051:41a0

Continued on next page

Table 11 - continued from previous page

Card number	Card name	PCI subsystem IDs
57	GrandTec 'Grand Video Capture' (Bt848)	4344:4142
58	Askey CPH060/ Phoebe TV Master Only (No FM)	
59	Askey CPH03x TV Capturer	
60	Modular Technology MM100PCTV	
61	AG Electronics GMV1	15cb:0101
62	Askey CPH061/ BESTBUY Easy TV (bt878)	
63	ATI TV-Wonder	1002:0001
64	ATI TV-Wonder VE	1002:0003
65	Lifview FlyVideo 2000S LR90	
66	Terratec TValueRadio	153b:1135, 153b:ff3b
67	IODATA GV-BCTV4/PCI	10fc:4050
68	3Dfx VoodooTV FM (Euro)	10b4:2637
69	Active Imaging AIMMS	
70	Prolink Pixelview PV-BT878P+ (Rev.4C,8E)	
71	Lifview FlyVideo 98EZ (capture only) LR51	1851:1851
72	Prolink Pixelview PV-BT878P+9B (PlayTV Pro rev.9B FM+NICAM)	1554:4011
73	Sensoray 311/611	6000:0311, 6000:0611
74	RemoteVision MX (RV605)	
75	Powercolor MTV878/ MTV878R/ MTV878F	
76	Canopus WinDVR PCI (COMPAQ Presario 3524JP, 5112JP)	0e11:0079
77	GrandTec Multi Capture Card (Bt878)	
78	Jetway TV/Capture JW-TV878-FBK, Kworld KW-TV878RF	0a01:17de
79	DSP Design TCVIDEO	
80	Hauppauge WinTV PVR	0070:4500
81	IODATA GV-BCTV5/PCI	10fc:4070, 10fc:d018
82	Osprey 100/150 (878)	0070:ff00
83	Osprey 100/150 (848)	
84	Osprey 101 (848)	
85	Osprey 101/151	
86	Osprey 101/151 w/ svid	
87	Osprey 200/201/250/251	
88	Osprey 200/250	0070:ff01
89	Osprey 210/220/230	
90	Osprey 500	0070:ff02
91	Osprey 540	0070:ff04
92	Osprey 2000	0070:ff03
93	IDS Eagle	
94	Pinnacle PCTV Sat	11bd:001c
95	Formac ProTV II (bt878)	
96	MachTV	
97	Euresys Picolo	

Continued on next page

Table 11 - continued from previous page

Card number	Card name	PCI subsystem IDs
98	ProVideo PV150	aa00:1460, aa01:1461, aa02:1462, aa03:1463, aa04:1464, aa05:1465, aa06:1466, aa07:1467
99	AD-TVK503	
100	Hercules Smart TV Stereo	
101	Pace TV & Radio Card	
102	IVC-200	0000:a155, 0001:a155, 0002:a155, 0003:a155, 0100:a155, 0101:a155, 0102:a155, 0103:a155, 0800:a155, 0801:a155, 0802:a155, 0803:a155
103	Grand X-Guard / Trust 814PCI	0304:0102
104	Nebula Electronics DigiTV	0071:0101
105	ProVideo PV143	aa00:1430, aa00:1431, aa00:1432, aa00:1433, aa03:1433
106	PHYTEC VD-009-X1 VD-011 MiniDIN (bt878)	
107	PHYTEC VD-009-X1 VD-011 Combi (bt878)	
108	PHYTEC VD-009 MiniDIN (bt878)	
109	PHYTEC VD-009 Combi (bt878)	
110	IVC-100	ff00:a132
111	IVC-120G	ff00:a182, ff01:a182, ff02:a182, ff03:a182, ff04:a182, ff05:a182, ff06:a182, ff07:a182, ff08:a182, ff09:a182, ff0a:a182, ff0b:a182, ff0c:a182, ff0d:a182, ff0e:a182, ff0f:a182
112	pcHDTV HD-2000 TV	7063:2000

Continued on next page

Table 11 - continued from previous page

Card number	Card name	PCI subsystem IDs
113	Twinhan DST + clones	11bd:0026, 1822:0001, 270f:fc00, 1822:0026
114	Winfast VC100	107d:6607
115	Teppro TEV-560/InterVision IV-560	
116	SIMUS GVC1100	aa6a:82b2
117	NGS NGSTV+	
118	LMLBT4	
119	Tekram M205 PRO	
120	Conceptronic CONTVFMi	
121	Euresys Picolo Tetra	1805:0105, 1805:0106, 1805:0107, 1805:0108
122	Spirit TV Tuner	
123	AVerMedia AVerTV DVB-T 771	1461:0771
124	AverMedia AverTV DVB-T 761	1461:0761
125	MATRIX Vision Sigma-SQ	
126	MATRIX Vision Sigma-SLC	
127	APAC Viewcomp 878(AMAX)	
128	DViCO FusionHDTV DVB-T Lite	18ac:db10, 18ac:db11
129	V-Gear MyVCD	
130	Super TV Tuner	
131	Tibet Systems 'Progress DVR' CS16	
132	Kodicom 4400R (master)	
133	Kodicom 4400R (slave)	
134	Adlink RTV24	
135	DViCO FusionHDTV 5 Lite	18ac:d500
136	Acorp Y878F	9511:1540
137	Conceptronic CTVFMi v2	036e:109e
138	Prolink Pixelview PV-BT878P+ (Rev.2E)	
139	Prolink PixelView PlayTV MPEG2 PV-M4900	
140	Osprey 440	0070:ff07
141	Asound Skyeye PCTV	
142	Sabrent TV-FM (bttv version)	
143	Hauppauge ImpactVCB (bt878)	0070:13eb
144	MagicTV	
145	SSAI Security Video Interface	4149:5353
146	SSAI Ultrasound Video Interface	414a:5353
147	VoodooTV 200 (USA)	121a:3000
148	DViCO FusionHDTV 2	dbc0:d200
149	Typhoon TV-Tuner PCI (50684)	
150	Geovision GV-600	008a:763c
151	Kozumi KTV-01C	

Continued on next page

Table 11 - continued from previous page

Card number	Card name	PCI subsystem IDs
152	Encore ENL TV-FM-2	1000:1801
153	PHYTEC VD-012 (bt878)	
154	PHYTEC VD-012-X1 (bt878)	
155	PHYTEC VD-012-X2 (bt878)	
156	IVCE-8784	0000:f050, 0001:f050, 0002:f050, 0003:f050
157	Geovision GV-800(S) (master)	800a:763d
158	Geovision GV-800(S) (slave)	800b:763d, 800c:763d, 800d:763d
159	ProVideo PV183	1830:1540, 1831:1540, 1832:1540, 1833:1540, 1834:1540, 1835:1540, 1836:1540, 1837:1540
160	Tongwei Video Technology TD-3116	f200:3116
161	Aposonic W-DVR	0279:0228
162	Adlink MPG24	
163	Bt848 Capture 14MHz	
164	CyberVision CV06 (SV)	
165	Kworld V-Stream Xpert TV PVR878	
166	PCI-8604PW	

### **CX18 cards list**

Those cards are supported by cx18 driver:

- Hauppauge HVR-1600 (ESMT memory)
- Hauppauge HVR-1600 (Samsung memory)
- Compro VideoMate H900
- Yuan MPC718 MiniPCI DVB-T/Analog
- Conexant Raptor PAL/SECAM
- Toshiba Qosmio DVB-T/Analog
- Leadtek WinFast PVR2100
- Leadtek WinFast DVR3100
- GoTView PCI DVD3 Hybrid
- Hauppauge HVR-1600 (s5h1411/tda18271)

## cx23885 cards list

Card number	Card name	PCI subsystem IDs
0	UNKNOWN/GENERIC	0070:3400
1	Hauppauge WinTV-HVR1800lp	0070:7600
2	Hauppauge WinTV-HVR1800	0070:7800, 0070:7801, 0070:7809
3	Hauppauge WinTV-HVR1250	0070:7911
4	DViCO FusionHDTV5 Express	18ac:d500
5	Hauppauge WinTV-HVR1500Q	0070:7790, 0070:7797
6	Hauppauge WinTV-HVR1500	0070:7710, 0070:7717
7	Hauppauge WinTV-HVR1200	0070:71d1, 0070:71d3
8	Hauppauge WinTV-HVR1700	0070:8101
9	Hauppauge WinTV-HVR1400	0070:8010
10	DViCO FusionHDTV7 Dual Express	18ac:d618
11	DViCO FusionHDTV DVB-T Dual Express	18ac:db78
12	Leadtek Winfast PxDVR3200 H	107d:6681
13	Compro VideoMate E650F	185b:e800
14	TurboSight TBS 6920	6920:8888
15	TeVii S470	d470:9022
16	DVBWorld DVB-S2 2005	0001:2005
17	NetUP Dual DVB-S2 CI	1b55:2a2c
18	Hauppauge WinTV-HVR1270	0070:2211
19	Hauppauge WinTV-HVR1275	0070:2215, 0070:221d, 0070:22f2
20	Hauppauge WinTV-HVR1255	0070:2251, 0070:22f1
21	Hauppauge WinTV-HVR1210	0070:2291, 0070:2295, 0070:2299, 0070:229d, 0070:22f0, 0070:22f3, 0070:22f4, 0070:22f5
22	Mygica X8506 DMB-TH	14f1:8651
23	Magic-Pro ProHDTV Extreme 2	14f1:8657
24	Hauppauge WinTV-HVR1850	0070:8541
25	Compro VideoMate E800	1858:e800
26	Hauppauge WinTV-HVR1290	0070:8551
27	Mygica X8558 PRO DMB-TH	14f1:8578
28	LEADTEK WinFast PxTV1200	107d:6f22

Continued on next page

Table 12 - continued from previous page

Card number	Card name	PCI subsystem IDs
29	GoTView X5 3D Hybrid	5654:2390
30	NetUP Dual DVB-T/C-CI RF	1b55:e2e4
31	Leadtek Winfast PxDVR3200 H XC4000	107d:6f39
32	MPX-885	
33	Mygica X8502/X8507 ISDB-T	14f1:8502
34	TerraTec Cinergy T PCIe Dual	153b:117e
35	TeVii S471	d471:9022
36	Hauppauge WinTV-HVR1255	0070:2259
37	Prof Revolution DVB-S2 8000	8000:3034
38	Hauppauge WinTV-HVR4400/HVR5500	0070:c108, 0070:c138, 0070:c1f8
39	AVerTV Hybrid Express Slim HC81R	1461:d939
40	TurboSight TBS 6981	6981:8888
41	TurboSight TBS 6980	6980:8888
42	Leadtek Winfast PxDVR2200	107d:6f21
43	Hauppauge ImpactVCB-e	0070:7133, 0070:7137
44	DViCO FusionHDTV DVB-T Dual Express2	18ac:db98
45	DVBSky T9580	4254:9580
46	DVBSky T980C	4254:980c
47	DVBSky S950C	4254:950c
48	Technotrend TT-budget CT2-4500 CI	13c2:3013
49	DVBSky S950	4254:0950
50	DVBSky S952	4254:0952
51	DVBSky T982	4254:0982
52	Hauppauge WinTV-HVR5525	0070:f038
53	Hauppauge WinTV Starburst	0070:c12a
54	ViewCast 260e	1576:0260
55	ViewCast 460e	1576:0460
56	Hauppauge WinTV-QuadHD-DVB	0070:6a28, 0070:6b28
57	Hauppauge WinTV-QuadHD-ATSC	0070:6a18, 0070:6b18
58	Hauppauge WinTV-HVR-1265(161111)	0070:2a18
59	Hauppauge WinTV-Starburst2	0070:f02a
60	Hauppauge WinTV-QuadHD-DVB(885)	
61	Hauppauge WinTV-QuadHD-ATSC(885)	
62	AVerMedia CE310B	1461:3100

## CX88 cards list

Card number	Card name	PCI subsystem IDs
0	UNKNOWN/GENERIC	
1	Hauppauge WinTV 34xxx models	0070:3400, 0070:3401
2	GDI Black Gold	14c7:0106, 14c7:0107
3	PixelView	1554:4811
4	ATI TV Wonder Pro	1002:00f8, 1002:00f9
5	Leadtek Winfast 2000XP Expert	107d:6611, 107d:6613
6	AverTV Studio 303 (M126)	1461:000b
7	MSI <a href="#">TV-@nywhere</a> Master	1462:8606
8	Leadtek Winfast DV2000	107d:6620, 107d:6621
9	Leadtek PVR 2000	107d:663b, 107d:663c, 107d:6632, 107d:6630, 107d:6638, 107d:6631, 107d:6637, 107d:663d
10	IODATA GV-VCP3/PCI	10fc:d003
11	Prolink PlayTV PVR	
12	ASUS PVR-416	1043:4823, 1461:c111
13	MSI <a href="#">TV-@nywhere</a>	
14	KWorld/VStream XPert DVB-T	17de:08a6
15	DViCO FusionHDTV DVB-T1	18ac:db00
16	KWorld LTV883RF	
17	DViCO FusionHDTV 3 Gold-Q	18ac:d810, 18ac:d800
18	Hauppauge Nova-T DVB-T	0070:9002, 0070:9001, 0070:9000
19	Conexant DVB-T reference design	14f1:0187
20	Provideo PV259	1540:2580
21	DViCO FusionHDTV DVB-T Plus	18ac:db10, 18ac:db11
22	pcHDTV HD3000 HDTV	7063:3000
23	digitalnow DNTV Live! DVB-T	17de:a8a6
24	Hauppauge WinTV 28xxx (Roslyn) models	0070:2801
25	Digital-Logic MICROSPACE Entertainment Center (MEC)	14f1:0342
26	IODATA GV/BCTV7E	10fc:d035

Continued on next page

Table 13 - continued from previous page

Card number	Card name	PCI subsystem IDs
27	PixelView PlayTV Ultra Pro (Stereo)	
28	DViCO FusionHDTV 3 Gold-T	18ac:d820
29	ADS Tech Instant TV DVB-T PCI	1421:0334
30	TerraTec Cinergy 1400 DVB-T	153b:1166
31	DViCO FusionHDTV 5 Gold	18ac:d500
32	AverMedia UltraTV Media Center PCI 550	1461:8011
33	Kworld V-Stream Xpert DVD	
34	ATI HDTV Wonder	1002:a101
35	WinFast DTV1000-T	107d:665f
36	AVerTV 303 (M126)	1461:000a
37	Hauppauge Nova-S-Plus DVB-S	0070:9201, 0070:9202
38	Hauppauge Nova-SE2 DVB-S	0070:9200
39	KWorld DVB-S 100	17de:08b2, 1421:0341
40	Hauppauge WinTV-HVR1100 DVB-T/Hybrid	0070:9400, 0070:9402
41	Hauppauge WinTV-HVR1100 DVB-T/Hybrid (Low Profile)	0070:9800, 0070:9802
42	digitalnow DNTV Live! DVB-T Pro	1822:0025, 1822:0019
43	KWorld/VStream XPert DVB-T with cx22702	17de:08a1, 12ab:2300
44	DViCO FusionHDTV DVB-T Dual Digital	18ac:db50, 18ac:db54
45	KWorld HardwareMpegTV XPert	17de:0840, 1421:0305
46	DViCO FusionHDTV DVB-T Hybrid	18ac:db40, 18ac:db44
47	pcHDTV HD5500 HDTV	7063:5500
48	Kworld MCE 200 Deluxe	17de:0841
49	PixelView PlayTV P7000	1554:4813
50	NPG Tech Real TV FM Top 10	14f1:0842
51	WinFast DTV2000 H	107d:665e
52	Geniatech DVB-S	14f1:0084
53	Hauppauge WinTV-HVR3000 TriMode Analog/DVB-S/DVB-T	0070:1404, 0070:1400, 0070:1401, 0070:1402
54	Norwood Micro TV Tuner	
55	Shenzhen Tungsten Ages Tech TE-DTV-250 / Swann OEM	c180:c980
56	Hauppauge WinTV-HVR1300 DVB-T/Hybrid MPEG Encoder	0070:9600, 0070:9601, 0070:9602
57	ADS Tech Instant Video PCI	1421:0390

Continued on next page

Table 13 - continued from previous page

Card number	Card name	PCI subsystem IDs
58	Pinnacle PCTV HD 800i	11bd:0051
59	DViCO FusionHDTV 5 PCI nano	18ac:d530
60	Pinnacle Hybrid PCTV	12ab:1788
61	Leadtek TV2000 XP Global	107d:6f18, 107d:6618, 107d:6619
62	PowerColor RA330	14f1:ea3d
63	Geniatech X8000-MT DVBT	14f1:8852
64	DViCO FusionHDTV DVB-T PRO	18ac:db30
65	DViCO FusionHDTV 7 Gold	18ac:d610
66	Prolink Pixelview MPEG 8000GT	1554:4935
67	Kworld PlusTV HD PCI 120 (ATSC 120)	17de:08c1
68	Hauppauge WinTV-HVR4000 DVB-S/S2/T/Hybrid	0070:6900, 0070:6904, 0070:6902
69	Hauppauge WinTV-HVR4000(Lite) DVB-S/S2	0070:6905, 0070:6906
70	TeVii S460 DVB-S/S2	d460:9022
71	Omicom SS4 DVB-S/S2 PCI	A044:2011
72	TBS 8920 DVB-S/S2	8920:8888
73	TeVii S420 DVB-S	d420:9022
74	Prolink Pixelview Global Extreme	1554:4976
75	PROF 7300 DVB-S/S2	B033:3033
76	SATTRADE ST4200 DVB-S/S2	b200:4200
77	TBS 8910 DVB-S	8910:8888
78	Prof 6200 DVB-S	b022:3022
79	Terratec Cinergy HT PCI MKII	153b:1177
80	Hauppauge WinTV-IR Only	0070:9290
81	Leadtek WinFast DTV1800 Hybrid	107d:6654
82	WinFast DTV2000 H rev. J	107d:6f2b
83	Prof 7301 DVB-S/S2	b034:3034
84	Samsung SMT 7020 DVB-S	18ac:dc00, 18ac:dccd
85	Twinhan VP-1027 DVB-S	1822:0023
86	TeVii S464 DVB-S/S2	d464:9022
87	Leadtek WinFast DTV2000 H PLUS	107d:6f42
88	Leadtek WinFast DTV1800 H (XC4000)	107d:6f38
89	Leadtek TV2000 XP Global (SC4100)	107d:6f36
90	Leadtek TV2000 XP Global (XC4100)	107d:6f43
91	NotOnlyTV LV3H	

## IVTV cards list

Card number	Card name	PCI subsystem IDs
0	Hauppauge WinTV PVR-250	IVTV16 104d:813d
1	Hauppauge WinTV PVR-350	IVTV16 104d:813d
2	Hauppauge WinTV PVR-150	IVTV16 104d:813d
3	AVerMedia M179	IVTV15 1461:a3cf, IVTV15 1461:a3ce
4	Yuan MPG600, Kuroutoshikou ITVC16-STVLP	IVTV16 12ab:fff3, IVTV16 12ab:ffff
5	YUAN MPG160, Kuroutoshikou ITVC15-STVLP, I/O Data GV-M2TV/PCI	IVTV15 10fc:40a0
6	Yuan PG600, Diamond PVR-550	IVTV16 ff92:0070, IVTV16 ffab:0600
7	Adaptec VideOh! AVC-2410	IVTV16 9005:0093
8	Adaptec VideOh! AVC-2010	IVTV16 9005:0092
9	Nagase Transgear 5000TV	IVTV16 1461:bfff
10	AOpen VA2000MAX-SNT6	IVTV16 0000:ff5f
11	Yuan MPG600GR, Kuroutoshikou CX23416GYC-STVLP	IVTV16 12ab:0600, IVTV16 fbab:0600, IVTV16 1154:0523
12	I/O Data GV-MVP/RX, GV-MVP/RX2W (dual tuner)	IVTV16 10fc:d01e, IVTV16 10fc:d038, IVTV16 10fc:d039
13	I/O Data GV-MVP/RX2E	IVTV16 10fc:d025
14	GotView PCI DVD	IVTV16 12ab:0600
15	GotView PCI DVD2 Deluxe	IVTV16 ffac:0600
16	Yuan MPC622	IVTV16 ff01:d998
17	Digital Cowboy DCT-MTVP1	IVTV16 1461:bfff

Continued on next page

Table 14 - continued from previous page

Card number	Card name	PCI subsystem IDs
18	Yuan PG600-2, GotView PCI DVD Lite	IVTV16 ffab:0600, IVTV16 ffad:0600
19	Club3D ZAP-TV1x01	IVTV16 ffab:0600
20	AVerTV MCE 116 Plus	IVTV16 1461:c439
21	ASUS Falcon2	IVTV16 1043:4b66, IVTV16 1043:462e, IVTV16 1043:4b2e
22	AVerMedia PVR-150 Plus / AVerTV M113 Partsnic (Daewoo) Tuner	IVTV16 1461:c034, IVTV16 1461:c035
23	AVerMedia EZMaker PCI Deluxe	IVTV16 1461:c03f
24	AVerMedia M104	IVTV16 1461:c136
25	Buffalo PC-MV5L/PCI	IVTV16 1154:052b
26	AVerMedia UltraTV 1500 MCE / AVerTV M113 Philips Tuner	IVTV16 1461:c019, IVTV16 1461:c01b
27	Sony VAIO Giga Pocket (ENX Kikyou)	IVTV16 104d:813d
28	Hauppauge WinTV PVR-350 (V1)	IVTV16 104d:813d
29	Yuan MPG600GR, Kuroutoshikou CX23416GYC-STVLP (no GR)	IVTV16 104d:813d
30	Yuan MPG600GR, Kuroutoshikou CX23416GYC-STVLP (no GR/YCS)	IVTV16 104d:813d

**SAA7134 cards list**

Card number	Card name	PCI subsystem IDs
0	UNKNOWN/GENERIC	
1	Proteus Pro [philips reference design]	1131:2001, 1131:2001

Continued on next page

Table 15 - continued from previous page

Card number	Card name	PCI subsystem IDs
2	LifeView FlyVIDEO3000	5168:0138, 4e42:0138
3	LifeView/Typhoon FlyVIDEO2000	5168:0138, 4e42:0138
4	EMPRESS	1131:6752
5	SKNet Monster TV	1131:4e85
6	Tevion MD 9717	
7	KNC One TV-Station RDS / Typhoon TV Tuner RDS	1131:fe01, 1894:fe01
8	Terratec Cinergy 400 TV	153b:1142
9	Medion 5044	
10	Kworld/KuroutoShikou SAA7130-TVPCI	
11	Terratec Cinergy 600 TV	153b:1143
12	Medion 7134	16be:0003, 16be:5000
13	Typhoon TV+Radio 90031	
14	ELSA EX-VISION 300TV	1048:226b
15	ELSA EX-VISION 500TV	1048:226a
16	ASUS TV-FM 7134	1043:4842, 1043:4830, 1043:4840
17	AOPEN VA1000 POWER	1131:7133
18	BMK MPEX No Tuner	
19	Compro VideoMate TV	185b:c100
20	Matrox CronosPlus	102B:48d0
21	10MOONS PCI TV CAPTURE CARD	1131:2001
22	AverMedia M156 / Medion 2819	1461:a70b
23	BMK MPEX Tuner	
24	KNC One TV-Station DVR	1894:a006
25	ASUS TV-FM 7133	1043:4843
26	Pinnacle PCTV Stereo (saa7134)	11bd:002b
27	Manli MuchTV M-TV002	
28	Manli MuchTV M-TV001	
29	Nagase Sangyo TransGear 3000TV	1461:050c
30	Elitegroup ECS TVP3XP FM1216 Tuner Card(PAL-BG,FM)	1019:4cb4
31	Elitegroup ECS TVP3XP FM1236 Tuner Card (NTSC,FM)	1019:4cb5
32	AVACS SmartTV	
33	AVerMedia DVD EZMaker	1461:10ff
34	Noval Prime TV 7133	
35	AverMedia AverTV Studio 305	1461:2115
36	UPMOST PURPLE TV	12ab:0800
37	Items MuchTV Plus / IT-005	
38	Terratec Cinergy 200 TV	153b:1152

Continued on next page

Table 15 - continued from previous page

Card number	Card name	PCI subsystem IDs
39	LifeView FlyTV Platinum Mini	5168:0212, 4e42:0212, 5169:1502
40	Compro VideoMate TV PVR/FM	185b:c100
41	Compro VideoMate TV Gold+	185b:c100
42	Sabrent SBT-TVFM (saa7130)	
43	:Zolid Xpert TV7134	
44	Empire PCI TV-Radio LE	
45	Avermedia AVerTV Studio 307	1461:9715
46	AVerMedia Cardbus TV/Radio (E500)	1461:d6ee
47	Terratec Cinergy 400 mobile	153b:1162
48	Terratec Cinergy 600 TV MK3	153b:1158
49	Compro VideoMate Gold+ Pal	185b:c200
50	Pinnacle PCTV 300i DVB-T + PAL	11bd:002d
51	ProVideo PV952	1540:9524
52	AverMedia AverTV/305	1461:2108
53	ASUS TV-FM 7135	1043:4845
54	LifeView FlyTV Platinum FM / Gold	5168:0214, 5168:5214, 1489:0214, 5168:0304
55	LifeView FlyDVB-T DUO / MSI TV@nywhere Duo	5168:0306, 4E42:0306
56	Avermedia AVerTV 307	1461:a70a
57	Avermedia AVerTV GO 007 FM	1461:f31f
58	ADS Tech Instant TV (saa7135)	1421:0350, 1421:0351, 1421:0370, 1421:1370
59	Kworld/Tevion V-Stream Xpert TV PVR7134	
60	LifeView/Typhoon/Genius FlyDVB-T Duo Cardbus	5168:0502, 4e42:0502, 1489:0502
61	Philips TOUGH DVB-T reference design	1131:2004
62	Compro VideoMate TV Gold+II	
63	Kworld Xpert TV PVR7134	
64	FlyTV mini Asus Digimatrix	1043:0210
65	V-Stream Studio TV Terminator	
66	Yuan TUN-900 (saa7135)	
67	Beholder BeholdTV 409 FM	0000:4091
68	GoTView 7135 PCI	5456:7135
69	Philips EUROPA V3 reference design	1131:2004
70	Compro Videomate DVB-T300	185b:c900
71	Compro Videomate DVB-T200	185b:c901
72	RTD Embedded Technologies VFG7350	1435:7350

Continued on next page

Table 15 - continued from previous page

Card number	Card name	PCI subsystem IDs
73	RTD Embedded Technologies VFG7330	1435:7330
74	LifeView FlyTV Platinum Mini2	14c0:1212
75	AVerMedia AVerTVHD MCE A180	1461:1044
76	SKNet MonsterTV Mobile	1131:4ee9
77	Pinnacle PCTV 40i/50i/110i (saa7133)	11bd:002e
78	ASUSTeK P7131 Dual	1043:4862
79	Sedna/MuchTV PC TV Cardbus TV/Radio (ITO25 Rev:2B)	
80	ASUS Digimatrix TV	1043:0210
81	Philips Tiger reference design	1131:2018
82	MSI TV@Anywhere plus	1462:6231, 1462:8624
83	Terratec Cinergy 250 PCI TV	153b:1160
84	LifeView FlyDVB Trio	5168:0319
85	AverTV DVB-T 777	1461:2c05, 1461:2c05
86	LifeView FlyDVB-T / Genius VideoWonder DVB-T	5168:0301, 1489:0301
87	ADS Instant TV Duo Cardbus PTV331	0331:1421
88	Tevion/KWorld DVB-T 220RF	17de:7201
89	ELSA EX-VISION 700TV	1048:226c
90	Kworld ATSC110/115	17de:7350, 17de:7352
91	AVerMedia A169 B	1461:7360
92	AVerMedia A169 B1	1461:6360
93	Medion 7134 Bridge #2	16be:0005
94	LifeView FlyDVB-T Hybrid Cardbus/MSI TV @nywhere A/D NB	5168:3306, 5168:3502, 5168:3307, 4e42:3502
95	LifeView FlyVIDEO3000 (NTSC)	5169:0138
96	Medion Md8800 Quadro	16be:0007, 16be:0008, 16be:000d
97	LifeView FlyDVB-S /Acorp TV134DS	5168:0300, 4e42:0300
98	Proteus Pro 2309	0919:2003
99	AVerMedia TV Hybrid A16AR	1461:2c00
100	Asus Europa2 OEM	1043:4860
101	Pinnacle PCTV 310i	11bd:002f
102	Avermedia AVerTV Studio 507	1461:9715
103	Compro Videomate DVB-T200A	

Continued on next page

Table 15 - continued from previous page

Card number	Card name	PCI subsystem IDs
104	Hauppauge WinTV-HVR1110 DVB-T/Hybrid	0070:6700, 0070:6701, 0070:6702, 0070:6703, 0070:6704, 0070:6705
105	Terratec Cinergy HT PCMCIA	153b:1172
106	Encore ENLTV	1131:2342, 1131:2341, 3016:2344
107	Encore ENLTV-FM	1131:230f
108	Terratec Cinergy HT PCI	153b:1175
109	Philips Tiger - S Reference design	
110	Avermedia M102	1461:f31e
111	ASUS P7131 4871	1043:4871
112	ASUSTeK P7131 Hybrid	1043:4876
113	Elitegroup ECS TVP3XP FM1246 Tuner Card (PAL,FM)	1019:4cb6
114	KWorld DVB-T 210	17de:7250
115	Sabrent PCMCIA TV-PCB05	0919:2003
116	10MOONS TM300 TV Card	1131:2304
117	Avermedia Super 007	1461:f01d
118	Beholder BeholdTV 401	0000:4016
119	Beholder BeholdTV 403	0000:4036
120	Beholder BeholdTV 403 FM	0000:4037
121	Beholder BeholdTV 405	0000:4050
122	Beholder BeholdTV 405 FM	0000:4051
123	Beholder BeholdTV 407	0000:4070
124	Beholder BeholdTV 407 FM	0000:4071
125	Beholder BeholdTV 409	0000:4090
126	Beholder BeholdTV 505 FM	5ace:5050
127	Beholder BeholdTV 507 FM / BeholdTV 509 FM	5ace:5070, 5ace:5090
128	Beholder BeholdTV Columbus TV/FM	0000:5201
129	Beholder BeholdTV 607 FM	5ace:6070
130	Beholder BeholdTV M6	5ace:6190
131	Twinhan Hybrid DTV-DVB 3056 PCI	1822:0022
132	Genius TVGO AM11MCE	
133	NXP Snake DVB-S reference design	
134	Medion/Creatix CTX953 Hybrid	16be:0010
135	MSI TV@nywhere A/D v1.1	1462:8625
136	AVerMedia Cardbus TV/Radio (E506R)	1461:f436
137	AVerMedia Hybrid TV/Radio (A16D)	1461:f936
138	Avermedia M115	1461:a836
139	Compro VideoMate T750	185b:c900
140	Avermedia DVB-S Pro A700	1461:a7a1

Continued on next page

Table 15 - continued from previous page

Card number	Card name	PCI subsystem IDs
141	Avermedia DVB-S Hybrid+FM A700	1461:a7a2
142	Beholder BeholdTV H6	5ace:6290
143	Beholder BeholdTV M63	5ace:6191
144	Beholder BeholdTV M6 Extra	5ace:6193
145	AVerMedia MiniPCI DVB-T Hybrid M103	1461:f636, 1461:f736
146	ASUSTeK P7131 Analog	
147	Asus Tiger 3in1	1043:4878
148	Encore ENLTV-FM v5.3	1a7f:2008
149	Avermedia PCI pure analog (M135A)	1461:f11d
150	Zogis Real Angel 220	
151	ADS Tech Instant HDTV	1421:0380
152	Asus Tiger Rev:1.00	1043:4857
153	Kworld Plus TV Analog Lite PCI	17de:7128
154	Avermedia AVerTV GO 007 FM Plus	1461:f31d
155	Hauppauge WinTV-HVR1150 ATSC/QAM-Hybrid	0070:6706, 0070:6708
156	Hauppauge WinTV-HVR1120 DVB-T/Hybrid	0070:6707, 0070:6709, 0070:670a
157	Avermedia AVerTV Studio 507UA	1461:a11b
158	AVerMedia Cardbus TV/Radio (E501R)	1461:b7e9
159	Beholder BeholdTV 505 RDS	0000:505B
160	Beholder BeholdTV 507 RDS	0000:5071
161	Beholder BeholdTV 507 RDS	0000:507B
162	Beholder BeholdTV 607 FM	5ace:6071
163	Beholder BeholdTV 609 FM	5ace:6090
164	Beholder BeholdTV 609 FM	5ace:6091
165	Beholder BeholdTV 607 RDS	5ace:6072
166	Beholder BeholdTV 607 RDS	5ace:6073
167	Beholder BeholdTV 609 RDS	5ace:6092
168	Beholder BeholdTV 609 RDS	5ace:6093
169	Compro VideoMate S350/S300	185b:c900
170	AverMedia AVerTV Studio 505	1461:a115
171	Beholder BeholdTV X7	5ace:7595
172	RoverMedia TV Link Pro FM	19d1:0138
173	Zolid Hybrid TV Tuner PCI	1131:2004
174	Asus Europa Hybrid OEM	1043:4847
175	Leadtek Winfast DTV1000S	107d:6655
176	Beholder BeholdTV 505 RDS	0000:5051
177	Hawell HW-404M7	
178	Beholder BeholdTV H7	5ace:7190
179	Beholder BeholdTV A7	5ace:7090
180	Avermedia PCI M733A	1461:4155, 1461:4255
181	TechoTrend TT-budget T-3000	13c2:2804

Continued on next page

Table 15 - continued from previous page

Card number	Card name	PCI subsystem IDs
182	Kworld PCI SBTVD/ISDB-T Full-Seg Hybrid	17de:b136
183	Compro VideoMate Vista M1F	185b:c900
184	Encore ENLTV-FM 3	1a7f:2108
185	MagicPro ProHDTV Pro2 DMB-TH/Hybrid	17de:d136
186	Beholder BeholdTV 501	5ace:5010
187	Beholder BeholdTV 503 FM	5ace:5030
188	Sensoray 811/911	6000:0811, 6000:0911
189	Kworld PC150-U	17de:a134
190	Asus My Cinema PS3-100	1043:48cd
191	Hawell HW-9004V1	
192	AverMedia AverTV Satellite Hybrid+FM A706	1461:2055
193	WIS Voyager or compatible	1905:7007
194	AverMedia AverTV/505	1461:a10a
195	Leadtek Winfast TV2100 FM	107d:6f3a
196	SnaZio* TVPVR PRO	1779:13cf

**SAA7164 cards list**

Card number	Card name	PCI subsystem IDs
0	Unknown	
1	Generic Rev2	
2	Generic Rev3	
3	Hauppauge WinTV-HVR2250	0070:8880, 0070:8810
4	Hauppauge WinTV-HVR2200	0070:8980
5	Hauppauge WinTV-HVR2200	0070:8900
6	Hauppauge WinTV-HVR2200	0070:8901
7	Hauppauge WinTV-HVR2250	0070:8891, 0070:8851
8	Hauppauge WinTV-HVR2250	0070:88A1
9	Hauppauge WinTV-HVR2200	0070:8940
10	Hauppauge WinTV-HVR2200	0070:8953
11	Hauppauge WinTV-HVR2255(proto)	0070:f111
12	Hauppauge WinTV-HVR2255	0070:f111
13	Hauppauge WinTV-HVR2205	0070:f123, 0070:f120

## Platform drivers

There are several drivers that are focused on providing support for functionality that are already included at the main board, and don't use neither USB nor PCI bus. Those drivers are called platform drivers, and are very popular on embedded devices.

The current supported of platform drivers (not including staging drivers) are listed below

	Driver	Name
am437x-vpfe	TI AM437x	VPFE
aspeed-video	Aspeed	AST2400 and AST2500
atmel-isc	ATMEL	Image Sensor Controller (ISC)
atmel-isi	ATMEL	Image Sensor Interface (ISI)
c8sectpfe	SDR	platform devices
c8sectpfe	SDR	platform devices
cafe_ccic	Marvell 88ALP01 (Cafe)	CMOS Camera Controller
cdns-csi2rx	Cadence	MIPI-CSI2 RX Controller
cdns-csi2tx	Cadence	MIPI-CSI2 TX Controller
coda-vpu	Chips&Media	Coda multi-standard codec IP
dm355_ccdc	TI DM355	CCDC video capture
dm644x_ccdc	TI DM6446	CCDC video capture
exynos-fimc-is	EXYNOS4x12	FIMC-IS (Imaging Subsystem)
exynos-fimc-lite	EXYNOS	FIMC-LITE camera interface
exynos-gsc	Samsung Exynos	G-Scaler
exy	Samsung S5P/EXYNOS4	SoC series Camera Subsystem
fsl-viu	Freescale	VIU
imx-pxp	i.MX	Pixel Pipeline (PXP)
isdf	TI DM365	ISIF video capture
mmp_camera	Marvell Armada 610	integrated camera controller
mtk_jpeg	Mediatek	JPEG Codec
mtk-mdp	Mediatek	MDP
mtk-vcodec-dec	Mediatek	Video Codec
mtk-vpu	Mediatek	Video Processor Unit
mx2_emmaprp	MX2 eMMA	PrP
omap3-isp	OMAP 3	Camera
omap-vout	OMAP2/OMAP3	V4L2-Display
pxa_camera	PXA27x	Quick Capture Interface
qcom-camss	Qualcomm	V4L2 Camera Subsystem
rcar-csi2	R-Car	MIPI CSI-2 Receiver
rcar_drif	Renesas	Digital Radio Interface (DRIF)
rcar-fcp	Renesas	Frame Compression Processor
rcar_fdp1	Renesas	Fine Display Processor
rcar_jpu	Renesas	JPEG Processing Unit
rcar-vin	R-Car	Video Input (VIN)
renesas-ceu	Renesas	Capture Engine Unit (CEU)
rockchip-rga	Rockchip	Raster 2d Graphic Acceleration Unit
s3c-camif	Samsung S3C24XX/S3C64XX	SoC Camera Interface

Continued on next page

Table 16 - continued from previous page

Driver	Name
s5p-csis	S5P/EXYNOS MIPI-CSI2 receiver (MIPI-CSIS)
s5p-fimc	S5P/EXYNOS4 FIMC/CAMIF camera interface
s5p-g2d	Samsung S5P and EXYNOS4 G2D 2d graphics accelerator
s5p-jpeg	Samsung S5P/Exynos3250/Exynos4 JPEG codec
s5p-mfc	Samsung S5P MFC Video Codec
sh_veu	SuperH VEU mem2mem video processing
sh_vou	SuperH VOU video output
stm32-dcml	STM32 Digital Camera Memory Interface (DCMI)
sun4i-csi	Allwinner A10 CMOS Sensor Interface Support
sun6i-csi	Allwinner V3s Camera Sensor Interface
sun8i-di	Allwinner Deinterlace
sun8i-rotate	Allwinner DE2 rotation
ti-cal	TI Memory-to-memory multimedia devices
ti-csc	TI DVB platform devices
ti-vpe	TI VPE (Video Processing Engine)
venus-enc	Qualcomm Venus V4L2 encoder/decoder
via-camera	VIAFB camera controller
video-mux	Video Multiplexer
vpif_display	TI DaVinci VPIF V4L2-Display
vpif_capture	TI DaVinci VPIF video capture
vpss	TI DaVinci VPBE V4L2-Display
vsp1	Renesas VSP1 Video Processing Engine
xilinx-tpg	Xilinx Video Test Pattern Generator
xilinx-video	Xilinx Video IP (EXPERIMENTAL)
xilinx-vtc	Xilinx Video Timing Controller

### MMC/SDIO DVB adapters

Driver	Name
smssdio	Siano SMS1xxx based MDTV via SDIO interface

### Radio drivers

There is also support for pure AM/FM radio, and even for some FM radio transmitters:

Driver	Name
si4713	Silicon Labs Si4713 FM Radio Transmitter
radio-aztech	Aztech/Packard Bell Radio
radio-cadet	ADS Cadet AM/FM Tuner
radio-gemtek	GemTek Radio card (or compatible)
radio-maxiradio	Guillemot MAXI Radio FM 2000 radio
radio-miropcm20	miroSOUND PCM20 radio
radio-aimslab	AIMSlab RadioTrack (aka RadioReveal)

Continued on next page

Table 17 - continued from previous page

	Driver	Name
radio-rtrack2	AIMSlab	RadioTrack II
saa7706h	SAA7706H	Car Radio DSP
radio-sf16fmi	SF16-FMI/SF16-FMP/SF16-FMD	Radio
radio-sf16fmr2	SF16-FMR2/SF16-FMD2	Radio
radio-shark	Griffin radio	SHARK USB radio receiver
shark2	Griffin radio	SHARK2 USB radio receiver
radio-si470x-common	Silicon Labs Si470x	FM Radio Receiver
radio-si476x	Silicon Laboratories Si476x	I2C FM Radio
radio-tea5764	TEA5764	I2C FM radio
tef6862	TEF6862	Car Radio Enhanced Selectivity Tuner
radio-terratec	TerraTec	ActiveRadio ISA Standalone
radio-timb		Enable the Timberdale radio driver
radio-trust		Trust FM radio card
radio-typhoon		Typhoon Radio (a.k.a. EcoRadio)
radio-wl1273	Texas Instruments WL1273	I2C FM Radio
fm_drv		ISA radio devices
fm_drv		ISA radio devices
radio-zoltrix	Zoltrix	Radio
dsbr100	D-Link/GemTek	USB FM radio
radio-keene	Keene	FM Transmitter USB
radio-ma901	Masterkit MA901	USB FM radio
radio-mr800	AverMedia MR 800	USB FM radio
radio-raremono	Thanko' s	Raremono AM/FM/SW radio
radio-si470x-usb	Silicon Labs Si470x	FM Radio Receiver support with USB
radio-usb-si4713	Silicon Labs Si4713	FM Radio Transmitter support with USB

## I<sup>2</sup>C drivers

The I<sup>2</sup>C (Inter-Integrated Circuit) bus is a three-wires bus used internally at the media cards for communication between different chips. While the bus is not visible to the Linux Kernel, drivers need to send and receive commands via the bus. The Linux Kernel driver abstraction has support to implement different drivers for each component inside an I<sup>2</sup>C bus, as if the bus were visible to the main system board.

One of the problems with I<sup>2</sup>C devices is that sometimes the same device may work with different I<sup>2</sup>C hardware. This is common, for example, on devices that comes with a tuner for North America market, and another one for Europe. Some drivers have a `tuner= modprobe` parameter to allow using a different tuner number in order to address such issue.

The current supported of I<sup>2</sup>C drivers (not including staging drivers) are listed below.

**Audio decoders, processors and mixers**

Driver	Name
cs3308	Cirrus Logic CS3308 audio ADC
cs5345	Cirrus Logic CS5345 audio ADC
cs53l32a	Cirrus Logic CS53L32A audio ADC
mSP3400	Micronas MSP34xx audio decoders
sony-btf-mpx	Sony BTF' s internal MPX
tda1997x	NXP TDA1997x HDMI receiver
tda7432	Philips TDA7432 audio processor
tda9840	Philips TDA9840 audio processor
tea6415c	Philips TEA6415C audio processor
tea6420	Philips TEA6420 audio processor
tlv320aic23b	Texas Instruments TLV320AIC23B audio codec
tvaudio	Simple audio decoder chips
uda1342	Philips UDA1342 audio codec
vp27smpx	Panasonic VP27' s internal MPX
wm8739	Wolfson Microelectronics WM8739 stereo audio ADC
wm8775	Wolfson Microelectronics WM8775 audio ADC with input mixer

**Audio/Video compression chips**

Driver	Name
saa6752hs	Philips SAA6752HS MPEG-2 Audio/Video Encoder

**Camera sensor devices**

Driver	Name
et8ek8	ET8EK8 camera sensor
hi556	Hynix Hi-556 sensor
imx214	Sony IMX214 sensor
imx219	Sony IMX219 sensor
imx258	Sony IMX258 sensor
imx274	Sony IMX274 sensor
imx290	Sony IMX290 sensor
imx319	Sony IMX319 sensor
imx355	Sony IMX355 sensor
m5mols	Fujitsu M-5MOLS 8MP sensor
mt9m001	mt9m001
mt9m032	MT9M032 camera sensor
mt9m111	mt9m111, mt9m112 and mt9m131
mt9p031	Aptina MT9P031
mt9t001	Aptina MT9T001
mt9t112	Aptina MT9T111/MT9T112

Continued on next page

Table 18 – continued from previous page

Driver	Name
mt9v011	Micron mt9v011 sensor
mt9v032	Micron MT9V032 sensor
mt9v111	Aptina MT9V111 sensor
noon010pc30	Siliconfile NOON010PC30 sensor
ov13858	OmniVision OV13858 sensor
ov2640	OmniVision OV2640 sensor
ov2659	OmniVision OV2659 sensor
ov2680	OmniVision OV2680 sensor
ov2685	OmniVision OV2685 sensor
ov5640	OmniVision OV5640 sensor
ov5645	OmniVision OV5645 sensor
ov5647	OmniVision OV5647 sensor
ov5670	OmniVision OV5670 sensor
ov5675	OmniVision OV5675 sensor
ov5695	OmniVision OV5695 sensor
ov6650	OmniVision OV6650 sensor
ov7251	OmniVision OV7251 sensor
ov7640	OmniVision OV7640 sensor
ov7670	OmniVision OV7670 sensor
ov772x	OmniVision OV772x sensor
ov7740	OmniVision OV7740 sensor
ov8856	OmniVision OV8856 sensor
ov9640	OmniVision OV9640 sensor
ov9650	OmniVision OV9650/OV9652 sensor
rj54n1cb0c	Sharp RJ54N1CB0C sensor
s5c73m3	Samsung S5C73M3 sensor
s5k4ecgx	Samsung S5K4ECGX sensor
s5k5baf	Samsung S5K5BAF sensor
s5k6a3	Samsung S5K6A3 sensor
s5k6aa	Samsung S5K6AAFX sensor
smiapp	SMIA+ +/SMIA sensor
sr030pc30	Siliconfile SR030PC30 sensor
vs6624	ST VS6624 sensor

## Flash devices

Driver	Name
adp1653	ADP1653 flash
lm3560	LM3560 dual flash driver
lm3646	LM3646 dual flash driver

## IR I2C driver

Driver	Name
ir-kbd-i2c	I2C module for IR

## Lens drivers

Driver	Name
ad5820	AD5820 lens voice coil
ak7375	AK7375 lens voice coil
dw9714	DW9714 lens voice coil
dw9807-vc	DW9807 lens voice coil

## Miscellaneous helper chips

Driver	Name
video-i2c	I2C transport video
m52790	Mitsubishi M52790 A/V switch
st-mipid02	STMicroelectronics MIPID02 CSI-2 to PARALLEL bridge
ths7303	THS7303/53 Video Amplifier

## RDS decoders

Driver	Name
saa6588	SAA6588 Radio Chip RDS decoder

## SDR tuner chips

Driver	Name
max2175	Maxim 2175 RF to Bits tuner

## Video and audio decoders

Driver	Name
cx25840	Conexant CX2584x audio/video decoders
saa717x	Philips SAA7171/3/4 audio/video decoders

**Video decoders**

Driver	Name
adv7180	Analog Devices ADV7180 decoder
adv7183	Analog Devices ADV7183 decoder
adv748x	Analog Devices ADV748x decoder
adv7604	Analog Devices ADV7604 decoder
adv7842	Analog Devices ADV7842 decoder
bt819	BT819A VideoStream decoder
bt856	BT856 VideoStream decoder
bt866	BT866 VideoStream decoder
ks0127	KS0127 video decoder
ml86v7667	OKI ML86V7667 video decoder
saa7110	Philips SAA7110 video decoder
saa7115	Philips SAA7111/3/4/5 video decoders
tc358743	Toshiba TC358743 decoder
tvp514x	Texas Instruments TVP514x video decoder
tvp5150	Texas Instruments TVP5150 video decoder
tvp7002	Texas Instruments TVP7002 video decoder
tw2804	Techwell TW2804 multiple video decoder
tw9903	Techwell TW9903 video decoder
tw9906	Techwell TW9906 video decoder
tw9910	Techwell TW9910 video decoder
vpx3220	vpx3220a, vpx3216b & vpx3214c video decoders

**Video encoders**

Driver	Name
ad9389b	Analog Devices AD9389B encoder
adv7170	Analog Devices ADV7170 video encoder
adv7175	Analog Devices ADV7175 video encoder
adv7343	ADV7343 video encoder
adv7393	ADV7393 video encoder
adv7511-v4l2	Analog Devices ADV7511 encoder
ak881x	AK8813/AK8814 video encoders
saa7127	Philips SAA7127/9 digital video encoders
saa7185	Philips SAA7185 video encoder
ths8200	Texas Instruments THS8200 video encoder

## Video improvement chips

Driver	Name
upd64031a	NEC Electronics uPD64031A Ghost Reduction
upd64083	NEC Electronics uPD64083 3-Dimensional Y/C separation

## Tuner drivers

Driver	Name
e4000	Elonics E4000 silicon tuner
fc0011	Fitipower FC0011 silicon tuner
fc0012	Fitipower FC0012 silicon tuner
fc0013	Fitipower FC0013 silicon tuner
fc2580	FCI FC2580 silicon tuner
it913x	ITE Tech IT913x silicon tuner
m88rs6000t	Montage M88RS6000 internal tuner
max2165	Maxim MAX2165 silicon tuner
mc44s803	Freescale MC44S803 Low Power CMOS Broadband tuners
msi001	Mirics MSi001
mt2060	Microtune MT2060 silicon IF tuner
mt2063	Microtune MT2063 silicon IF tuner
mt20xx	Microtune 2032 / 2050 tuners
mt2131	Microtune MT2131 silicon tuner
mt2266	Microtune MT2266 silicon tuner
mxl301rf	MaxLinear MxL301RF tuner
mxl5005s	MaxLinear MSL5005S silicon tuner
mxl5007t	MaxLinear MxL5007T silicon tuner
qm1d1b0004	Sharp QM1D1B0004 tuner
qm1d1c0042	Sharp QM1D1C0042 tuner
qt1010	Quantek QT1010 silicon tuner
r820t	Rafael Micro R820T silicon tuner
si2157	Silicon Labs Si2157 silicon tuner
tuner-types	Simple tuner support
tda18212	NXP TDA18212 silicon tuner
tda18218	NXP TDA18218 silicon tuner
tda18250	NXP TDA18250 silicon tuner
tda18271	NXP TDA18271 silicon tuner
tda827x	Philips TDA827X silicon tuner
tda8290	TDA 8290/8295 + 8275(a)/18271 tuner combo
tda9887	TDA 9885/6/7 analog IF demodulator
tea5761	TEA 5761 radio tuner
tea5767	TEA 5767 radio tuner
tua9001	Infineon TUA9001 silicon tuner
tuner-xc2028	XCeive xc2028/xc3028 tuners
xc4000	Xceive XC4000 silicon tuner
xc5000	Xceive XC5000 silicon tuner

## Tuner cards list

	Tuner number	Card name
0		Temec PAL (4002 FH5)
1		Philips PAL_I (FI1246 and compatibles)
2		Philips NTSC (FI1236,FM1236 and compatibles)
3		Philips (SECAM+PAL_BG) (FI1216MF, FM1216MF, FR1216MF)
4		NoTuner
5		Philips PAL_BG (FI1216 and compatibles)
6		Temec NTSC (4032 FY5)
7		Temec PAL_I (4062 FY5)
8		Temec NTSC (4036 FY5)
9		Alps HSBH1
10		Alps TSBE1
11		Alps TSBB5
12		Alps TSBE5
13		Alps TSBC5
14		Temec PAL_BG (4006FH5)
15		Alps TSCH6
16		Temec PAL_DK (4016 FY5)
17		Philips NTSC_M (MK2)
18		Temec PAL_I (4066 FY5)
19		Temec PAL* auto (4006 FN5)
20		Temec PAL_BG (4009 FR5) or PAL_I (4069 FR5)
21		Temec NTSC (4039 FR5)
22		Temec PAL/SECAM multi (4046 FM5)
23		Philips PAL_DK (FI1256 and compatibles)
24		Philips PAL/SECAM multi (FQ1216ME)
25		LG PAL_I+FM (TAPC-I001D)
26		LG PAL_I (TAPC-I701D)
27		LG NTSC+FM (TPI8NSR01F)
28		LG PAL_BG+FM (TPI8PSB01D)
29		LG PAL_BG (TPI8PSB11D)
30		Temec PAL* auto + FM (4009 FN5)
31		SHARP NTSC_JP (2U5JF5540)
32		Samsung PAL TCPM9091PD27
33		MT20xx universal
34		Temec PAL_BG (4106 FH5)
35		Temec PAL_DK/SECAM_L (4012 FY5)
36		Temec NTSC (4136 FY5)
37		LG PAL (newer TAPC series)
38		Philips PAL/SECAM multi (FM1216ME MK3)
39		LG NTSC (newer TAPC series)
40		HITACHI V7-J180AT
41		Philips PAL_MK (FI1216 MK)
42		Philips FCV1236D ATSC/NTSC dual in
43		Philips NTSC MK3 (FM1236MK3 or FM1236/F)
44		Philips 4 in 1 (ATI TV Wonder Pro/Conexant)

Continued on next page

Table 20 – continued from previous page

	Tuner number	Card name
45		Microtune 4049 FM5
46		Panasonic VP27s/ENGE4324D
47		LG NTSC (TAPE series)
48		Tenna TNF 8831 BGFF)
49		Microtune 4042 FI5 ATSC/NTSC dual in
50		TCL 2002N
51		Philips PAL/SECAM_D (FM 1256 I-H3)
52		Thomson DTT 7610 (ATSC/NTSC)
53		Philips FQ1286
54		Philips/NXP TDA 8290/8295 + 8275/8275A/18271
55		TCL 2002MB
56		Philips PAL/SECAM multi (FQ1216AME MK4)
57		Philips FQ1236A MK4
58		Ymec TVision TVF-8531MF/8831MF/8731MF
59		Ymec TVision TVF-5533MF
60		Thomson DTT 761X (ATSC/NTSC)
61		Tena TNF9533-D/IF/TNF9533-B/DF
62		Philips TEA5767HN FM Radio
63		Philips FMD1216ME MK3 Hybrid Tuner
64		LG TDVS-H06xF
65		Ymec TVF66T5-B/DF
66		LG TALN series
67		Philips TD1316 Hybrid Tuner
68		Philips TUV1236D ATSC/NTSC dual in
69		Tena TNF 5335 and similar models
70		Samsung TCPN 2121P30A
71		Xceive xc2028/xc3028 tuner
72		Thomson FE6600
73		Samsung TCPG 6121P30A
75		Philips TEA5761 FM Radio
76		Xceive 5000 tuner
77		TCL tuner MF02GIP-5N-E
78		Philips FMD1216MEX MK3 Hybrid Tuner
79		Philips PAL/SECAM multi (FM1216 MK5)
80		Philips FQ1216LME MK3 PAL/SECAM w/active loopthrough
81		Partsnic (Daewoo) PTI-5NF05
82		Philips CU1216L
83		NXP TDA18271
84		Sony BTF-Pxn01Z
85		Philips FQ1236 MK5
86		Tena TNF5337 MFD
87		Xceive 4000 tuner
88		Xceive 5000C tuner
89		Sony BTF-PG472Z PAL/SECAM
90		Sony BTF-PK467Z NTSC-M-JP
91		Sony BTF-PB463Z NTSC-M

### Frontend drivers

---

**Note:**

- 1) There is no guarantee that every frontend driver works out of the box with every card, because of different wiring.
  - 2) The demodulator chips can be used with a variety of tuner/PLL chips, and not all combinations are supported. Often the demodulator and tuner/PLL chip are inside a metal box for shielding, and the whole metal box has its own part number.
- 

### Common Interface (EN50221) controller drivers

Driver	Name
cxd2099	Sony CXD2099AR Common Interface driver
sp2	CIMaX SP2

### ATSC (North American/Korean Terrestrial/Cable DTV) frontends

Driver	Name
au8522_dig	Auvitek AU8522 based DTV demod
au8522_decoder	Auvitek AU8522 based ATV demod
bcm3510	Broadcom BCM3510
lg2160	LG Electronics LG216x based
lgdt3305	LG Electronics LGDT3304 and LGDT3305 based
lgdt3306a	LG Electronics LGDT3306A based
lgdt330x	LG Electronics LGDT3302/LGDT3303 based
nxt200x	NxtWave Communications NXT2002/NXT2004 based
or51132	Oren OR51132 based
or51211	Oren OR51211 based
s5h1409	Samsung S5H1409 based
s5h1411	Samsung S5H1411 based

### DVB-C (cable) frontends

Driver	Name
stv0297	ST STV0297 based
tda10021	Philips TDA10021 based
tda10023	Philips TDA10023 based
ves1820	VLSI VES1820 based

**DVB-S (satellite) frontends**

Driver	Name
cx24110	Conexant CX24110 based
cx24116	Conexant CX24116 based
cx24117	Conexant CX24117 based
cx24120	Conexant CX24120 based
cx24123	Conexant CX24123 based
ds3000	Montage Tehnology DS3000 based
mb86a16	Fujitsu MB86A16 based
mt312	Zarlink VP310/MT312/ZL10313 based
s5h1420	Samsung S5H1420 based
si21xx	Silicon Labs SI21XX based
stb6000	ST STB6000 silicon tuner
stv0288	ST STV0288 based
stv0299	ST STV0299 based
stv0900	ST STV0900 based
stv6110	ST STV6110 silicon tuner
tda10071	NXP TDA10071
tda10086	Philips TDA10086 based
tda8083	Philips TDA8083 based
tda8261	Philips TDA8261 based
tda826x	Philips TDA826X silicon tuner
ts2020	Montage Tehnology TS2020 based tuners
tua6100	Infineon TUA6100 PLL
cx24113	Conexant CX24113/CX24128 tuner for DVB-S/DSS
itd1000	Integrant ITD1000 Zero IF tuner for DVB-S/DSS
ves1x93	VLSI VES1893 or VES1993 based
zl10036	Zarlink ZL10036 silicon tuner
zl10039	Zarlink ZL10039 silicon tuner

**DVB-T (terrestrial) frontends**

Driver	Name
af9013	Afatech AF9013 demodulator
cx22700	Conexant CX22700 based
cx22702	Conexant cx22702 demodulator (OFDM)
cxd2820r	Sony CXD2820R
cxd2841er	Sony CXD2841ER
cxd2880	Sony CXD2880 DVB-T2/T tuner + demodulator
dib3000mb	DiBcom 3000M-B
dib3000mc	DiBcom 3000P/M-C
dib7000m	DiBcom 7000MA/MB/PA/PB/MC
dib7000p	DiBcom 7000PC
dib9000	DiBcom 9000
drxd	Micronas DRXD driver
ec100	E3C EC100
l64781	LSI L64781
mt352	Zarlink MT352 based
nxt6000	NxtWave Communications NXT6000 based
rtl2830	Realtek RTL2830 DVB-T
rtl2832	Realtek RTL2832 DVB-T
rtl2832_sdr	Realtek RTL2832 SDR
s5h1432	Samsung s5h1432 demodulator (OFDM)
si2168	Silicon Labs Si2168
sp8870	Spase sp8870 based
sp887x	Spase sp887x based
stv0367	ST STV0367 based
tda10048	Philips TDA10048HN based
tda1004x	Philips TDA10045H/TDA10046H based
zd1301_demod	ZyDAS ZD1301
zl10353	Zarlink ZL10353 based

**Digital terrestrial only tuners/PLL**

Driver	Name
dvb-pll	Generic I2C PLL based tuners
dib0070	DiBcom DiB0070 silicon base-band tuner
dib0090	DiBcom DiB0090 silicon base-band tuner

**ISDB-S (satellite) & ISDB-T (terrestrial) frontends**

Driver	Name
mn88443x	Socionext MN88443x
tc90522	Toshiba TC90522

**ISDB-T (terrestrial) frontends**

Driver	Name
dib8000	DiBcom 8000MB/MC
mb86a20s	Fujitsu mb86a20s
s921	Sharp S921 frontend

**Multistandard (cable + terrestrial) frontends**

Driver	Name
drxk	Micronas DRXK based
mn88472	Panasonic MN88472
mn88473	Panasonic MN88473
si2165	Silicon Labs si2165 based
tda18271c2dd	NXP TDA18271C2 silicon tuner

**Multistandard (satellite) frontends**

Driver	Name
m88ds3103	Montage Technology M88DS3103
mxl5xx	MaxLinear MxL5xx based tuner-demodulators
stb0899	STB0899 based
stb6100	STB6100 based tuners
stv090x	STV0900/STV0903(A/B) based
stv0910	STV0910 based
stv6110x	STV6110/(A) based tuners
stv6111	STV6111 based tuners

### SEC control devices for DVB-S

Driver	Name
a8293	Allegro A8293
af9033	Afatech AF9033 DVB-T demodulator
ascot2e	Sony Ascot2E tuner
atbm8830	AltoBeam ATBM8830/8831 DMB-TH demodulator
drx39xyj	Micronas DRX-J demodulator
helene	Sony HELENE Sat/Ter tuner (CXD2858ER)
horus3a	Sony Horus3A tuner
isl6405	ISL6405 SEC controller
isl6421	ISL6421 SEC controller
isl6423	ISL6423 SEC controller
ix2505v	Sharp IX2505V silicon tuner
lgs8gl5	Silicon Legend LGS-8GL5 demodulator (OFDM)
lgs8gxx	Legend Silicon LGS8913/LGS8GL5/LGS8GXX DMB-TH demodulator
lnbh25	LNBH25 SEC controller
lnbh29	LNBH29 SEC controller
lnbp21	LNBP21/LNBH24 SEC controllers
lnbp22	LNBP22 SEC controllers
m88rs2000	M88RS2000 DVB-S demodulator and tuner
tda665x	TDA665x tuner

### Tools to develop new frontends

Driver	Name
dvb_dummy_fe	Dummy frontend driver

### Firewire driver

The media subsystem also provides a firewire driver for digital TV:

Driver	Name
fireDTV	FireDTV and FloppyDTV

### Test drivers

In order to test userspace applications, there's a number of virtual drivers, with provide test functionality, simulating real hardware devices:

Driver	Name
vicodec	Virtual Codec Driver
vim2m	Virtual Memory-to-Memory Driver
vimc	Virtual Media Controller Driver (VIMC)
vivid	Virtual Video Test Driver

### 53.1.6 Video4Linux (V4L) driver-specific documentation

#### The bttv driver

##### Release notes for bttv

You' ll need at least these config options for bttv:

```
./scripts/config -e PCI
./scripts/config -m I2C
./scripts/config -m INPUT
./scripts/config -m MEDIA_SUPPORT
./scripts/config -e MEDIA_PCI_SUPPORT
./scripts/config -e MEDIA_ANALOG_TV_SUPPORT
./scripts/config -e MEDIA_DIGITAL_TV_SUPPORT
./scripts/config -e MEDIA_RADIO_SUPPORT
./scripts/config -e RC_CORE
./scripts/config -m VIDEO_BT848
```

If your board has digital TV, you' ll also need:

```
./scripts/config -m DVB_BT8XX
```

In this case, please see [How to get the bt8xx cards working](#) for additional notes.

#### Make bttv work with your card

If you have bttv compiled and installed, just booting the Kernel should be enough for it to try probing it. However, depending on the model, the Kernel may require additional information about the hardware, as the device may not be able to provide such info directly to the Kernel.

If it doesn' t bttv likely could not autodetect your card and needs some insmod options. The most important insmod option for bttv is "card=n" to select the correct card type. If you get video but no sound you' ve very likely specified the wrong (or no) card type. A list of supported cards is in [BTTV cards list](#).

If bttv takes very long to load (happens sometimes with the cheap cards which have no tuner), try adding this to your modules configuration file (usually, it is either `/etc/modules.conf` or some file at `/etc/modules-load.d/`, but the actual place depends on your distribution):

```
options i2c-algo-bit bit_test=1
```

Some cards may require an extra firmware file to work. For example, for the WinTV/PVR you need one firmware file from its driver CD, called: `hcamc.rbf`. It is inside a self-extracting zip file called `pvr45xxx.exe`. Just placing it at the `/etc/firmware` directory should be enough for it to be autoload during the driver's probing mode (e. g. when the Kernel boots or when the driver is manually loaded via `modprobe` command).

If your card isn't listed in BTTV cards list or if you have trouble making audio work, please read [Still doesn't work?](#).

### Autodetecting cards

`bttv` uses the PCI Subsystem ID to autodetect the card type. `lspci` lists the Subsystem ID in the second line, looks like this:

```
00:0a.0 Multimedia video controller: Brooktree Corporation Bt878 (rev 02)
      Subsystem: Hauppauge computer works Inc. WinTV/GO
      Flags: bus master, medium devsel, latency 32, IRQ 5
      Memory at e2000000 (32-bit, prefetchable) [size=4K]
```

only bt878-based cards can have a subsystem ID (which does not mean that every card really has one). bt848 cards can't have a Subsystem ID and therefore can't be autodetected. There is a list with the ID's at BTTV cards list (in case you are interested or want to mail patches with updates).

### Still doesn't work?

I do NOT have a lab with 30+ different grabber boards and a PAL/NTSC/SECAM test signal generator at home, so I often can't reproduce your problems. This makes debugging very difficult for me.

If you have some knowledge and spare time, please try to fix this yourself (patches very welcome of course...) You know: The linux slogan is "Do it yourself" .

There is a mailing list at <http://vger.kernel.org/vger-lists.html#linux-media>

If you have trouble with some specific TV card, try to ask there instead of mailing me directly. The chance that someone with the same card listens there is much higher...

For problems with sound: There are a lot of different systems used for TV sound all over the world. And there are also different chips which decode the audio signal. Reports about sound problems ("stereo doesn't work") are pretty useless unless you include some details about your hardware and the TV sound scheme used in your country (or at least the country you are living in).

## Modprobe options

**Note:** The following argument list can be outdated, as we might add more options if ever needed. In case of doubt, please check with `modinfo <module>`.

This command prints various information about a kernel module, among them a complete and up-to-date list of insmod options.

### bttv

The bt848/878 (grabber chip) driver

insmod args:

```

card=n          card type, see CARDLIST for a list.
tuner=n        tuner type, see CARDLIST for a list.
radio=0/1      card supports radio
pll=0/1/2      pll settings

                  0: don't use PLL
                  1: 28 MHz crystal installed
                  2: 35 MHz crystal installed

triton1=0/1    for Triton1 (+others) compatibility
vsfx=0/1      yet another chipset bug compatibility bit
                see README.quirks for details on these two.

bigendian=n    Set the endianness of the gfx framebuffer.
                Default is native endian.
fieldnr=0/1    Count fields. Some TV descrambling software
                needs this, for others it only generates
                50 useless IRQs/sec. default is 0 (off).
autoload=0/1   autoload helper modules (tuner, audio).
                default is 1 (on).
bttv_verbose=0/1/2 verbose level (at insmod time, while
                looking at the hardware). default is 1.
bttv_debug=0/1 debug messages (for capture).
                default is 0 (off).
irq_debug=0/1  irq handler debug messages.
                default is 0 (off).
gbuffers=2-32  number of capture buffers for mmap'ed capture.
                default is 4.
gbufsize=      size of capture buffers. default and
                maximum value is 0x208000 (~2MB)
no_overlay=0    Enable overlay on broken hardware. There
                are some chipsets (SIS for example) which
                are known to have problems with the PCI DMA
                push used by bttv. bttv will disable overlay
                by default on this hardware to avoid crashes.
                With this insmod option you can override this.
no_overlay=1    Disable overlay. It should be used by broken
                hardware that doesn't support PCI2PCI direct
                transfers.
automute=0/1    Automatically mutes the sound if there is
                no TV signal, on by default. You might try

```

(continues on next page)

(continued from previous page)

	to disable this if you have bad input signal quality which leading to unwanted sound dropouts.
chroma_agc=0/1	AGC of chroma signal, off by default.
adc_crush=0/1	Luminance ADC crush, on by default.
i2c_udelay=	Allow reduce I2C speed. Default is 5 usecs (meaning 66,67 Kbps). The default is the maximum supported speed by kernel bitbang algorithm. You may use lower numbers, if I2C messages are lost (16 is known to work on all supported cards).
bttv_gpio=0/1	
gpiomask=	
audioall=	
audiomux=	
	See Sound-FAQ for a detailed description.
remap, card, radio and pll	accept up to four comma-separated arguments (for multiple boards).

**tuner** The tuner driver. You need this unless you want to use only with a camera or the board doesn't provide analog TV tuning.

insmod args:

debug=1	print some debug info to the syslog
type=n	type of the tuner chip. n as follows: see CARDLIST for a complete list.
pal=[bdgil]	select PAL variant (used for some tuners only, important for the audio carrier).

**tvaudio** Provide a single driver for all simple i2c audio control chips (tda/tea\*).

insmod args:

tda8425 = 1	enable/disable the support for the
tda9840 = 1	various chips.
tda9850 = 1	The tea6300 can't be autodetected and is
tda9855 = 1	therefore off by default, if you have
tda9873 = 1	this one on your card (STB uses these)
tda9874a = 1	you have to enable it explicitly.
tea6300 = 0	The two tda985x chips use the same i2c
tea6420 = 1	address and can't be distingished from
pic16c54 = 1	each other, you might have to disable
	the wrong one.
debug = 1	print debug messages

**msp3400** The driver for the msp34xx sound processor chips. If you have a stereo card, you probably want to insmod this one.

insmod args:

debug=1/2	print some debug info to the syslog, 2 is more verbose.
simple=1	Use the "short programming" method. Newer

(continues on next page)

(continued from previous page)

<code>once=1</code>	<code>msp34xx</code> versions support this. You need this for dbx stereo. Default is on if supported by the chip.
<code>amsound=1</code>	Don't check the TV-stations Audio mode every few seconds, but only once after channel switches.
	Audio carrier is AM/NICAM at 6.5 Mhz. This should improve things for french people, the carrier autoscans seems to work with FM only...

### If the box freezes hard with bttv

It might be a bttv driver bug. It also might be bad hardware. It also might be something else ...

Just mailing me “bttv freezes” isn’ t going to help much. This README has a few hints how you can help to pin down the problem.

### bttv bugs

If some version works and another doesn’ t it is likely to be a driver bug. It is very helpful if you can tell where exactly it broke (i.e. the last working and the first broken version).

With a hard freeze you probably doesn’ t find anything in the logfiles. The only way to capture any kernel messages is to hook up a serial console and let some terminal application log the messages. /me uses screen. See Linux Serial Console for details on setting up a serial console.

Read Bug hunting to learn how to get any useful information out of a register+stack dump printed by the kernel on protection faults (so-called “kernel oops” ).

If you run into some kind of deadlock, you can try to dump a call trace for each process using `sysrq-t` (see Linux Magic System Request Key Hacks). This way it is possible to figure where exactly some process in “D” state is stuck.

I’ ve seen reports that bttv 0.7.x crashes whereas 0.8.x works rock solid for some people. Thus probably a small buglet left somewhere in bttv 0.7.x. I have no idea where exactly, it works stable for me and a lot of other people. But in case you have problems with the 0.7.x versions you can give 0.8.x a try ...

### hardware bugs

Some hardware can't deal with PCI-PCI transfers (i.e. grabber => vga). Sometimes problems show up with bttv just because of the high load on the PCI bus. The bt848/878 chips have a few workarounds for known incompatibilities, see README.quirks.

Some folks report that increasing the pci latency helps too, although I'm not sure whenever this really fixes the problems or only makes it less likely to happen. Both bttv and btaudio have a insmod option to set the PCI latency of the device.

Some mainboard have problems to deal correctly with multiple devices doing DMA at the same time. bttv + ide seems to cause this sometimes, if this is the case you likely see freezes only with video and hard disk access at the same time. Updating the IDE driver to get the latest and greatest workarounds for hardware bugs might fix these problems.

### other

If you use some binary-only yunk (like nvidia module) try to reproduce the problem without.

IRQ sharing is known to cause problems in some cases. It works just fine in theory and many configurations. Nevertheless it might be worth a try to shuffle around the PCI cards to give bttv another IRQ or make it share the IRQ with some other piece of hardware. IRQ sharing with VGA cards seems to cause trouble sometimes. I've also seen funny effects with bttv sharing the IRQ with the ACPI bridge (and apci-enabled kernel).

### Bttv quirks

Below is what the bt878 data book says about the PCI bug compatibility modes of the bt878 chip.

The triton1 insmod option sets the EN\_TBFX bit in the control register. The vsfx insmod option does the same for EN\_VSFX bit. If you have stability problems you can try if one of these options makes your box work solid.

drivers/pci/quirks.c knows about these issues, this way these bits are enabled automatically for known-buggy chipsets (look at the kernel messages, bttv tells you).

### Normal PCI Mode

The PCI REQ signal is the logical-or of the incoming function requests. The internal GNT[0:1] signals are gated asynchronously with GNT and demultiplexed by the audio request signal. Thus the arbiter defaults to the video function at power-up and parks there during no requests for bus access. This is desirable since the video will request the bus more often. However, the audio will have highest bus access priority. Thus the audio will have first access to the bus even when issuing a request after the video request but before the PCI external arbiter has granted access to the Bt879. Neither function can preempt the other once on the bus.

The duration to empty the entire video PCI FIFO onto the PCI bus is very short compared to the bus access latency the audio PCI FIFO can tolerate.

### **430FX Compatibility Mode**

When using the 430FX PCI, the following rules will ensure compatibility:

- (1) Deassert REQ at the same time as asserting FRAME.
- (2) Do not reassert REQ to request another bus transaction until after finishing the previous transaction.

Since the individual bus masters do not have direct control of REQ, a simple logical-or of video and audio requests would violate the rules. Thus, both the arbiter and the initiator contain 430FX compatibility mode logic. To enable 430FX mode, set the EN\_TBFX bit as indicated in Device Control Register on page 104.

When EN\_TBFX is enabled, the arbiter ensures that the two compatibility rules are satisfied. Before GNT is asserted by the PCI arbiter, this internal arbiter may still logical-or the two requests. However, once the GNT is issued, this arbiter must lock in its decision and now route only the granted request to the REQ pin. The arbiter decision lock happens regardless of the state of FRAME because it does not know when FRAME will be asserted (typically - each initiator will assert FRAME on the cycle following GNT). When FRAME is asserted, it is the initiator's responsibility to remove its request at the same time. It is the arbiter's responsibility to allow this request to flow through to REQ and not allow the other request to hold REQ asserted. The decision lock may be removed at the end of the transaction: for example, when the bus is idle (FRAME and IRDY). The arbiter decision may then continue asynchronously until GNT is again asserted.

### **Interfacing with Non-PCI 2.1 Compliant Core Logic**

A small percentage of core logic devices may start a bus transaction during the same cycle that GNT is de-asserted. This is non PCI 2.1 compliant. To ensure compatibility when using PCs with these PCI controllers, the EN\_VSFX bit must be enabled (refer to Device Control Register on page 104). When in this mode, the arbiter does not pass GNT to the internal functions unless REQ is asserted. This prevents a bus transaction from starting the same cycle as GNT is de-asserted. This also has the side effect of not being able to take advantage of bus parking, thus lowering arbitration performance. The Bt879 drivers must query for these non-compliant devices, and set the EN\_VSFX bit only if required.

### Other elements of the tvcards array

If you are trying to make a new card work you might find it useful to know what the other elements in the tvcards array are good for:

video_inputs	- # of video inputs the card has
audio_inputs	- historical cruft, not used any more.
tuner	- which input is the tuner
svhs	- which input is svhs (all others are labeled composite)
muxsel	- video mux, input->registervalue mapping
pll	- same as pll= insmod option
tuner_type	- same as tuner= insmod option
*_modulename	- hint whenever some card needs this or that audio module loaded to work properly.
has_radio	- whenever this TV card has a radio tuner.
no_msp34xx	- "1" disables loading of msp3400.o module
no_tda9875	- "1" disables loading of tda9875.o module
needs_tvaudio	- set to "1" to load tvaudio.o module

If some config item is specified both from the tvcards array and as insmod option, the insmod option takes precedence.

### Cards

---

**Note:** For a more updated list, please check [https://linuxtv.org/wiki/index.php/Hardware\\_Device\\_Information](https://linuxtv.org/wiki/index.php/Hardware_Device_Information)

---

### Supported cards: Bt848/Bt848a/Bt849/Bt878/Bt879 cards

All cards with Bt848/Bt848a/Bt849/Bt878/Bt879 and normal Composite/S-VHS inputs are supported. Teletext and Intercast support (PAL only) for ALL cards via VBI sample decoding in software.

Some cards with additional multiplexing of inputs or other additional fancy chips are only partially supported (unless specifications by the card manufacturer are given). When a card is listed here it isn't necessarily fully supported.

All other cards only differ by additional components as tuners, sound decoders, EEPROMs, teletext decoders ...

### MATRIX Vision

MV-Delta - Bt848A - 4 Composite inputs, 1 S-VHS input (shared with 4th composite) - EEPROM

<http://www.matrix-vision.de/>

This card has no tuner but supports all 4 composite (1 shared with an S-VHS input) of the Bt848A. Very nice card if you only have satellite TV but several tuners connected to the card via composite.

Many thanks to Matrix-Vision for giving us 2 cards for free which made Bt848a/Bt849 single crystal operation support possible!!!

### **Miro/Pinnacle PCTV**

- Bt848 some (all??) come with 2 crystals for PAL/SECAM and NTSC
- PAL, SECAM or NTSC TV tuner (Philips or TEMIC)
- MSP34xx sound decoder on add on board decoder is supported but AFAIK does not yet work (other sound MUX setting in GPIO port needed??? somebody who fixed this???)
- 1 tuner, 1 composite and 1 S-VHS input
- tuner type is autodetected

<http://www.miro.de/> <http://www.miro.com/>

Many thanks for the free card which made first NTSC support possible back in 1997!

### **Hauppauge Win/TV pci**

There are many different versions of the Hauppauge cards with different tuners (TV+Radio ...), teletext decoders. Note that even cards with same model numbers have (depending on the revision) different chips on it.

- Bt848 (and others but always in 2 crystal operation???) newer cards have a Bt878
- PAL, SECAM, NTSC or tuner with or without Radio support

e.g.:

- PAL:
  - TDA5737: VHF, hyperband and UHF mixer/oscillator for TV and VCR 3-band tuners
  - TSA5522: 1.4 GHz I2C-bus controlled synthesizer, I2C 0xc2-0xc3
- NTSC:
  - TDA5731: VHF, hyperband and UHF mixer/oscillator for TV and VCR 3-band tuners
  - TSA5518: no datasheet available on Philips site
- Philips SAA5246 or SAA5284 ( or no) Teletext decoder chip with buffer RAM (e.g. Winbond W24257AS-35: 32Kx8 CMOS static RAM) SAA5246 (I2C 0x22) is supported
- 256 bytes EEPROM: Microchip 24LC02B or Philips 8582E2Y with configuration information I2C address 0xa0 (24LC02B also responds to 0xa2-0xaf)
- 1 tuner, 1 composite and (depending on model) 1 S-VHS input
- 14052B: mux for selection of sound source

- sound decoder: TDA9800, MSP34xx (stereo cards)

### Askey CPH-Series

Developed by TelSignal(?), OEMed by many vendors (Typhoon, Anubis, Dynalink)

- Card series: - CPH01x: BT848 capture only - CPH03x: BT848 - CPH05x: BT878 with FM - CPH06x: BT878 (w/o FM) - CPH07x: BT878 capture only
- TV standards: - CPH0x0: NTSC-M/M - CPH0x1: PAL-B/G - CPH0x2: PAL-I/I - CPH0x3: PAL-D/K - CPH0x4: SECAM-L/L - CPH0x5: SECAM-B/G - CPH0x6: SECAM-D/K - CPH0x7: PAL-N/N - CPH0x8: PAL-B/H - CPH0x9: PAL-M/M
- CPH03x was often sold as “TV capturer” .

Identifying:

- 1) 878 cards can be identified by PCI Subsystem-ID: - 144f:3000 = CPH06x - 144F:3002 = CPH05x w/ FM - 144F:3005 = CPH06x\_LC (w/o remote control)
- 2) The cards have a sticker with “CPH” -model on the back.
- 3) These cards have a number printed on the PCB just above the tuner metal box: - “80-CP2000300-x” = CPH03X - “80-CP2000500-x” = CPH05X - “80-CP2000600-x” = CPH06X / CPH06x\_LC

Askey sells these cards as “Magic TView series” , Brand “MagicXpress” . Other OEM often call these “Tview” , “TView99” or else.

### Lifeview Flyvideo Series:

The naming of these series differs in time and space.

**Identifying:**

- 1) Some models can be identified by PCI subsystem ID:
  - 1852:1852 = Flyvideo 98 FM
  - 1851:1850 = Flyvideo 98
  - 1851:1851 = Flyvideo 98 EZ (capture only)
- 2) There is a print on the PCB:
  - LR25 = Flyvideo (Zoran ZR36120, SAA7110A)
  - LR26 Rev.N = Flyvideo II (Bt848)
  - LR26 Rev.O = Flyvideo II (Bt878)
  - LR37 Rev.C = Flyvideo EZ (Capture only, ZR36120 + SAA7110)
  - LR38 Rev.A1= Flyvideo II EZ (Bt848 capture only)
  - LR50 Rev.Q = Flyvideo 98 (w/EEPROM and PCI subsystem ID)
  - LR50 Rev.W = Flyvideo 98 (no EEPROM)

- LR51 Rev.E = Flyvideo 98 EZ (capture only)
- LR90 = Flyvideo 2000 (Bt878)
- LR90 Flyvideo 2000S (Bt878) w/Stereo TV (Package incl. LR91 daughterboard)
- LR91 = Stereo daughter card for LR90
- LR97 = Flyvideo DVBS
- LR99 Rev.E = Low profile card for OEM integration (only internal audio!) bt878
- LR136 = Flyvideo 2100/3100 (Low profile, SAA7130/SAA7134)
- LR137 = Flyvideo DV2000/DV3000 (SAA7130/SAA7134 + IEEE1394)
- LR138 Rev.C= Flyvideo 2000 (SAA7130)
- LR138 Flyvideo 3000 (SAA7134) w/Stereo TV
  - These exist in variations w/FM and w/Remote sometimes denoted by suffixes "FM" and "R" .

3) You have a laptop (miniPCI card):

- Product = FlyTV Platinum Mini
- Model/Chip = LR212/saa7135
- Lifeview.com.tw states (Feb. 2002): "The FlyVideo2000 and FlyVideo2000s product name have renamed to FlyVideo98." Their Bt8x8 cards are listed as discontinued.
- Flyvideo 2000S was probably sold as Flyvideo 3000 in some countries(Europe?). The new Flyvideo 2000/3000 are SAA7130/SAA7134 based.

"Flyvideo II" had been the name for the 848 cards, nowadays (in Germany) this name is re-used for LR50 Rev.W.

The Lifeview website mentioned Flyvideo III at some time, but such a card has not yet been seen (perhaps it was the german name for LR90 [stereo]). These cards are sold by many OEMs too.

FlyVideo A2 (Elta 8680)= LR90 Rev.F (w/Remote, w/o FM, stereo TV by tda9821) {Germany}

Lifeview 3000 (Elta 8681) as sold by Plus(April 2002), Germany = LR138 w/saa7134

### lifeview config coding on gpio pins 0-9

- LR50 rev. Q (“PARTS: 7031505116), Tuner wurde als Nr. 5 erkannt, Eingänge SVideo, TV, Composite, Audio, Remote:
- CP9..1=100001001 (1: 0-Ohm-Widerstand gegen GND unbestückt; 0: bestückt)

### Typhoon TV card series:

These can be CPH, Flyvideo, Pixelview or KNC1 series.

Typhoon is the brand of Anubis.

Model 50680 got re-used, some model no. had different contents over time.

Models:

- 50680 “TV Tuner PCI Pal BG” (old,red package)=can be CPH03x(bt848) or CPH06x(bt878)
- 50680 “TV Tuner Pal BG” (blue package)= Pixelview PV-BT878P+ (Rev 9B)
- 50681 “TV Tuner PCI Pal I” (variant of 50680)
- 50682 “TView TV/FM Tuner Pal BG” = Flyvideo 98FM (LR50 Rev.Q)

---

**Note:** The package has a picture of CPH05x (which would be a real TView)

---

- 50683 “TV Tuner PCI SECAM” (variant of 50680)
- 50684 “TV Tuner Pal BG” = Pixelview 878TV(Rev.3D)
- 50686 “TV Tuner” = KNC1 TV Station
- 50687 “TV Tuner stereo” = KNC1 TV Station pro
- 50688 “TV Tuner RDS” (black package) = KNC1 TV Station RDS
- 50689 TV SAT DVB-S CARD CI PCI (SAA7146AH, SU1278?) = “KNC1 TV Station DVB-S”
- 50692 “TV/FM Tuner” (small PCB)
- 50694 TV TUNER CARD RDS (PHILIPS CHIPSET SAA7134HL)
- 50696 TV TUNER STEREO (PHILIPS CHIPSET SAA7134HL, MK3ME Tuner)
- 50804 PC-SAT TV/Audio Karte = Techni-PC-Sat (ZORAN 36120PQC, Tuner:Alps)
- 50866 TVIEW SAT RECEIVER+ADR
- 50868 “TV/FM Tuner Pal I” (variant of 50682)
- 50999 “TV/FM Tuner Secam” (variant of 50682)

## Guillemot

Models:

- Maxi-TV PCI (ZR36120)
- Maxi TV Video 2 = LR50 Rev.Q (FI1216MF, PAL BG+SECAM)
- Maxi TV Video 3 = CPH064 (PAL BG + SECAM)

## Mentor

Mentor TV card ( "55-878TV-U1" ) = Pixelview 878TV(Rev.3F) (w/FM w/Remote)

## Prolink

- TV cards:
  - PixelView Play TV pro - (Model: PV-BT878P+ REV 8E)
  - PixelView Play TV pro - (Model: PV-BT878P+ REV 9D)
  - PixelView Play TV pro - (Model: PV-BT878P+ REV 4C / 8D / 10A )
  - PixelView Play TV - (Model: PV-BT848P+)
  - 878TV - (Model: PV-BT878TV)
- Multimedia TV packages (card + software pack):
  - PixelView Play TV Theater - (Model: PV-M4200) = PixelView Play TV pro + Software
  - PixelView Play TV PAK - (Model: PV-BT878P+ REV 4E)
  - PixelView Play TV/VCR - (Model: PV-M3200 REV 4C / 8D / 10A )
  - PixelView Studio PAK - (Model: M2200 REV 4C / 8D / 10A )
  - PixelView PowerStudio PAK - (Model: PV-M3600 REV 4E)
  - PixelView DigitalVCR PAK - (Model: PV-M2400 REV 4C / 8D / 10A )
  - PixelView PlayTV PAK II (TV/FM card + usb camera) PV-M3800
  - PixelView PlayTV XP PV-M4700,PV-M4700(w/FM)
  - PixelView PlayTV DVR PV-M4600 package contents:PixelView PlayTV pro, windvr & videoMail s/w
- Further Cards:
  - PV-BT878P+rev.9B (Play TV Pro, opt. w/FM w/NICAM)
  - PV-BT878P+rev.2F
  - PV-BT878P Rev.1D (bt878, capture only)
  - XCapture PV-CX881P (cx23881)
  - PlayTV HD PV-CX881PL+, PV-CX881PL+(w/FM) (cx23881)

- DTV3000 PV-DTV3000P+ DVB-S CI = Twinhan VP-1030
- DTV2000 DVB-S = Twinhan VP-1020
- Video Conferencing:
  - PixelView Meeting PAK - (Model: PV-BT878P)
  - PixelView Meeting PAK Lite - (Model: PV-BT878P)
  - PixelView Meeting PAK plus - (Model: PV-BT878P+rev 4C/8D/10A)
  - PixelView Capture - (Model: PV-BT848P)
  - PixelView PlayTV USB pro
  - Model No. PV-NT1004+, PV-NT1004+ (w/FM) = NT1004 USB decoder chip + SAA7113 video decoder chip

### Dynalink

These are CPH series.

### Phoebemicro

- TV Master = CPH030 or CPH060
- TV Master FM = CPH050

### Genius/Kye

- Video Wonder/Genius Internet Video Kit = LR37 Rev.C
- Video Wonder Pro II (848 or 878) = LR26

### Tekram

- VideoCap C205 (Bt848)
- VideoCap C210 (zr36120 +Philips)
- CaptureTV M200 (ISA)
- CaptureTV M205 (Bt848)

## Lucky Star

- Image World Conference TV = LR50 Rev. Q

## Leadtek

- WinView 601 (Bt848)
- WinView 610 (Zoran)
- WinFast2000
- WinFast2000 XP

## Support for the Leadtek WinView 601 TV/FM

Author of this section: Jon Tombs <[jon@gte.esi.us.es](mailto:jon@gte.esi.us.es)>

This card is basically the same as all the rest (Bt484A, Philips tuner), the main difference is that they have attached a programmable attenuator to 3 GPIO lines in order to give some volume control. They have also stuck an infra-red remote control decoded on the board, I will add support for this when I get time (it simple generates an interrupt for each key press, with the key code is placed in the GPIO port).

I don' t yet have any application to test the radio support. The tuner frequency setting should work but it is possible that the audio multiplexer is wrong. If it doesn' t work, send me email.

- No Thanks to Leadtek they refused to answer any questions about their hardware. The driver was written by visual inspection of the card. If you use this driver, send an email insult to them, and tell them you won' t continue buying their hardware unless they support Linux.
- Little thanks to Princeton Technology Corp (<http://www.princeton.com.tw>) who make the audio attenuator. Their publicly available data-sheet available on their web site doesn' t include the chip programming information! Hidden on their server are the full data-sheets, but don' t ask how I found it.

To use the driver I use the following options, the tuner and pll settings might be different in your country. You can force it via modprobe parameters. For example:

```
modprobe bttv tuner=1 pll=28 radio=1 card=17
```

Sets tuner type 1 (Philips PAL\_I), PLL with a 28 MHz crystal, enables FM radio and selects bttv card ID 17 (Leadtek WinView 601).

### KNC One

- TV-Station
- TV-Station SE (+Software Bundle)
- TV-Station pro (+TV stereo)
- TV-Station FM (+Radio)
- TV-Station RDS (+RDS)
- TV Station SAT (analog satellite)
- TV-Station DVB-S

---

**Note:** newer Cards have saa7134, but model name stayed the same?

---

### Provideo

- PV951 or PV-951, now named PV-951T (also are sold as: Boeder TV-FM Video Capture Card, Titanmedia Supervision TV-2400, Provideo PV951 TF, 3DeMon PV951, MediaForte TV-Vision PV951, Yoko PV951, Vivanco Tuner Card PCI Art.-Nr.: 68404 )
- Surveillance Series:
  - PV-141
  - PV-143
  - PV-147
  - PV-148 (capture only)
  - PV-150
  - PV-151
- TV-FM Tuner Series:
  - PV-951TDV (tv tuner + 1394)
  - PV-951T/TF
  - PV-951PT/TF
  - PV-956T/TF Low Profile
  - PV-911

## Highscreen

Models:

- TV Karte = LR50 Rev.S
- TV-Boostar = Terratec Terra TV+ Version 1.0 (Bt848, tda9821) “ceb105.pcb”

## Zoltrix

Models:

- Face to Face Capture (Bt848 capture only) (PCB “VP-2848” )
- Face To Face TV MAX (Bt848) (PCB “VP-8482 Rev1.3” )
- Genie TV (Bt878) (PCB “VP-8790 Rev 2.1” )
- Genie Wonder Pro

## AVerMedia

- AVer FunTV Lite (ISA, AV3001 chipset) “M101.C”
- AVerTV
- AVerTV Stereo
- AVerTV Studio (w/FM)
- AVerMedia TV98 with Remote
- AVerMedia TV/FM98 Stereo
- AVerMedia TVCAM98
- TVCapture (Bt848)
- TVPhone (Bt848)
- TVCapture98 (=” AVerMedia TV98” in USA) (Bt878)
- TVPhone98 (Bt878, w/FM)

PCB	PCI-ID	Model-Name	Eep-rom	Tuner	Sound	Country
M101.C	ISA !					
M108-B	Bt848		-	FR1236		US <sup>2,3</sup>
M1A8-A	Bt848	AVer TV-Phone		FM1216 -		
M168-T	1461:0003	AVerTV Studio	48:17	FM1216	TDA9840TD <sup>1</sup>	w/FM w/Remote
M168-U	1461:0004	TVCap-ture98	40:11	FI1216	-	D w/Remote
M168II-B	1461:0003	Medion MD9592	48:16	FM1216	TDA9873HD	w/FM

- US site has different drivers for (as of 09/2002):
  - EZ Capture/InterCam PCI (BT-848 chip)
  - EZ Capture/InterCam PCI (BT-878 chip)
  - TV-Phone (BT-848 chip)
  - TV98 (BT-848 chip)
  - TV98 With Remote (BT-848 chip)
  - TV98 (BT-878 chip)
  - TV98 With Remote (BT-878)
  - TV/FM98 (BT-878 chip)
  - AVerTV
  - AVerTV Stereo
  - AVerTV Studio

DE hat diverse Treiber fuer diese Modelle (Stand 09/2002):

- TVPhone (848) mit Philips tuner FR12X6 (w/ FM radio)
- TVPhone (848) mit Philips tuner FM12X6 (w/ FM radio)
- TVCapture (848) w/Philips tuner FI12X6
- TVCapture (848) non-Philips tuner
- TVCapture98 (Bt878)
- TVPhone98 (Bt878)
- AVerTV und TVCapture98 w/VCR (Bt 878)
- AVerTVStudio und TVPhone98 w/VCR (Bt878)
- AVerTV GO Serie (Kein SVideo Input)
- AVerTV98 (BT-878 chip)
- AVerTV98 mit Fernbedienung (BT-878 chip)
- AVerTV/FM98 (BT-878 chip)
- VDOmate ([www.averm.com.cn](http://www.averm.com.cn)) = M168U ?

---

<sup>2</sup> Sony NE41S soldered (stereo sound?)

<sup>3</sup> Daughterboard M118-A w/ pic 16c54 and 4 MHz quartz

<sup>1</sup> Daughterboard MB68-A with TDA9820T and TDA9840T

## **Aimslab**

Models:

- Video Highway or “Video Highway TR200” (ISA)
- Video Highway Xtreme (aka “VHX” ) (Bt848, FM w/ TEA5757)

## **IXMicro (former: IMS=Integrated Micro Solutions)**

Models:

- IXTV BT848 (=TurboTV)
- IXTV BT878
- IMS TurboTV (Bt848)

## **Lifetec/Medion/Tevion/Aldi**

Models:

- LT9306/MD9306 = CPH061
- LT9415/MD9415 = LR90 Rev.F or Rev.G
- MD9592 = Avermedia TVphone98 (PCI\_ID=1461:0003), PCB-Rev=M168II-B (w/TDA9873H)
- MD9717 = KNC One (Rev D4, saa7134, FM1216 MK2 tuner)
- MD5044 = KNC One (Rev D4, saa7134, FM1216ME MK3 tuner)

## **Modular Technologies ([www.modulartech.com](http://www.modulartech.com)) UK**

Models:

- MM100 PCTV (Bt848)
- MM201 PCTV (Bt878, Bt832) w/ Quartzsight camera
- MM202 PCTV (Bt878, Bt832, tda9874)
- MM205 PCTV (Bt878)
- MM210 PCTV (Bt878) (Galaxy TV, Galaxymedia ?)

### Terratec

#### Models:

- Terra TV+ Version 1.0 (Bt848), “ceb105.PCB” printed on the PCB, TDA9821
- Terra TV+ Version 1.1 (Bt878), “LR74 Rev.E” printed on the PCB, TDA9821
- Terra TValueRadio, “LR102 Rev.C” printed on the PCB
- Terra TV/Radio+ Version 1.0, “80-CP2830100-0” TTTV3 printed on the PCB, “CPH010-E83” on the back, SAA6588T, TDA9873H
- Terra TValue Version BT878, “80-CP2830110-0 TTTV4” printed on the PCB, “CPH011-D83” on back
- Terra TValue Version 1.0 “ceb105.PCB” (really identical to Terra TV+ Version 1.0)
- Terra TValue New Revision “LR102 Rec.C”
- Terra Active Radio Upgrade (tea5757h, saa6588t)
- LR74 is a newer PCB revision of ceb105 (both incl. connector for Active Radio Upgrade)
- Cinergy 400 (saa7134), “E877 11(S)” , “PM820092D” printed on PCB
- Cinergy 600 (saa7134)

### Technisat

#### Models:

- Discos ADR PC-Karte ISA (no TV!)
- Discos ADR PC-Karte PCI (probably no TV?)
- Techni-PC-Sat (Sat. analog) Rev 1.2 (zr36120, vpx3220, stv0030, saa5246, BSJE3-494A)
- Mediafocus I (zr36120/zr36125, drp3510, Sat. analog + ADR Radio)
- Mediafocus II (saa7146, Sat. analog)
- SatADR Rev 2.1 (saa7146a, saa7113h, stv0056a, msp3400c, drp3510a, BSKE3-307A)
- SkyStar 1 DVB (AV7110) = Technotrend Premium
- SkyStar 2 DVB (B2C2) (=Sky2PC)

## Siemens

Multimedia eXtension Board (MXB) (SAA7146, SAA7111)

## Powercolor

Models:

- **MTV878** Package comes with different contents:
  - a) pcb “MTV878” (CARD=75)
  - b) Pixelview Rev. 4\_
- MTV878R w/Remote Control
- MTV878F w/Remote Control w/FM radio

## Pinnacle

PCTV models:

- Mirovideo PCTV (Bt848)
- Mirovideo PCTV SE (Bt848)
- Mirovideo PCTV Pro (Bt848 + Daughterboard for TV Stereo and FM)
- Studio PCTV Rave (Bt848 Version = Mirovideo PCTV)
- Studio PCTV Rave (Bt878 package w/o infrared)
- Studio PCTV (Bt878)
- Studio PCTV Pro (Bt878 stereo w/ FM)
- Pinnacle PCTV (Bt878, MT2032)
- Pinnacle PCTV Pro (Bt878, MT2032)
- Pinncale PCTV Sat (bt878a, HM1821/1221) [ “Conexant CX24110 with CX24108 tuner, aka HM1221/HM1811” ]
- Pinnacle PCTV Sat XE

M(J)PEG capture and playback models:

- DC1+ (ISA)
- DC10 (zr36057, zr36060, saa7110, adv7176)
- DC10+ (zr36067, zr36060, saa7110, adv7176)
- DC20 (ql16x24b,zr36050, zr36016, saa7110, saa7187 ...)
- DC30 (zr36057, zr36050, zr36016, vpx3220, adv7176, ad1843, tea6415, miro FST97A1)
- DC30+ (zr36067, zr36050, zr36016, vpx3220, adv7176)

- DC50 (zr36067, zr36050, zr36016, saa7112, adv7176 (2 pcs.?), ad1843, miro FST97A1, Lattice ???)

### Lenco

Models:

- MXR-9565 (=Technisat Mediafocus?)
- MXR-9571 (Bt848) (=CPH031?)
- MXR-9575
- MXR-9577 (Bt878) (=Prolink 878TV Rev.3x)
- MXTV-9578CP (Bt878) (= Prolink PV-BT878P+4E)

### lomega

Buz (zr36067, zr36060, saa7111, saa7185)

### LML

LML33 (zr36067, zr36060, bt819, bt856)

### Grandtec

Models:

- Grand Video Capture (Bt848)
- Multi Capture Card (Bt878)

### Koutech

Models:

- KW-606 (Bt848)
- KW-607 (Bt848 capture only)
- KW-606RSF
- KW-607A (capture only)
- KW-608 (Zoran capture only)

## **IODATA (jp)**

Models:

- GV-BCTV/PCI
- GV-BCTV2/PCI
- GV-BCTV3/PCI
- GV-BCTV4/PCI
- GV-VCP/PCI (capture only)
- GV-VCP2/PCI (capture only)

## **Canopus (jp)**

WinDVR = Kworld “KW-TVL878RF”

**[www.sigmacom.co.kr](http://www.sigmacom.co.kr)**

Sigma Cyber TV II

**[www.sasem.co.kr](http://www.sasem.co.kr)**

Lite OnAir TV

## **hama**

TV/Radio-Tuner Card, PCI (Model 44677) = CPH051

## **Sigma Designs**

Hollywood plus (em8300, em9010, adv7175), (PCB “M340-10” ) MPEG DVD decoder

## **Formac**

Models:

- iProTV (Card for iMac Mezzanine slot, Bt848+SCSI)
- ProTV (Bt848)
- ProTV II = ProTV Stereo (Bt878) [ “stereo” means FM stereo, tv is still mono]

### ATI

Models:

- TV-Wonder
- TV-Wonder VE

### Diamond Multimedia

DTV2000 (Bt848, tda9875)

### Aopen

- VA1000 Plus (w/ Stereo)
- VA1000 Lite
- VA1000 (=LR90)

### Intel

Models:

- Smart Video Recorder (ISA full-length)
- Smart Video Recorder pro (ISA half-length)
- Smart Video Recorder III (Bt848)

### STB

Models:

- STB Gateway 6000704 (bt878)
- STB Gateway 6000699 (bt848)
- STB Gateway 6000402 (bt848)
- STB TV130 PCI

### Videologic

Models:

- Captivator Pro/TV (ISA?)
- Captivator PCI/VC (Bt848 bundled with camera) (capture only)

## Technotrend

Models:

- TT-SAT PCI (PCB “Sat-PCI Rev.:1.3.1” ; zr36125, vpx3225d, stc0056a, Tuner:BSKE6-155A
- **TT-DVB-Sat**
  - revisions 1.1, 1.3, 1.5, 1.6 and 2.1
  - This card is sold as OEM from:
    - \* Siemens DVB-s Card
    - \* Hauppauge WinTV DVB-S
    - \* Technisat SkyStar 1 DVB
    - \* Galaxis DVB Sat
  - Now this card is called TT-PCline Premium Family
  - TT-Budget (saa7146, bsru6-701a) This card is sold as OEM from:
    - \* Hauppauge WinTV Nova
    - \* Satelco Standard PCI (DVB-S)
  - TT-DVB-C PCI

## Teles

DVB-s (Rev. 2.2, BSRV2-301A, data only?)

## Remote Vision

MX RV605 (Bt848 capture only)

## Boeder

Models:

- PC ChatCam (Model 68252) (Bt848 capture only)
- Tv/Fm Capture Card (Model 68404) = PV951

### Media-Surfer ([esc-kathrein.de](http://esc-kathrein.de))

Models:

- Sat-Surfer (ISA)
- Sat-Surfer PCI = Techni-PC-Sat
- Cable-Surfer 1
- Cable-Surfer 2
- Cable-Surfer PCI (zr36120)
- Audio-Surfer (ISA Radio card)

### Jetway ([www.jetway.com.tw](http://www.jetway.com.tw))

Models:

- JW-TV 878M
- JW-TV 878 = KWorld KW-TV878RF

### Galaxis

Models:

- Galaxis DVB Card S CI
- Galaxis DVB Card C CI
- Galaxis DVB Card S
- Galaxis DVB Card C
- Galaxis plug.in S [neuer Name: Galaxis DVB Card S CI]

### Hauppauge

Models:

- many many WinTV models ...
- WinTV DVBS = Technotrend Premium 1.3
- WinTV NOVA = Technotrend Budget 1.1 “S-DVB DATA”
- WinTV NOVA-CI “SDVBACI”
- WinTV Nova USB (=Technotrend USB 1.0)
- WinTV-Nexus-s (=Technotrend Premium 2.1 or 2.2)
- WinTV PVR
- WinTV PVR 250
- WinTV PVR 450

### US models

-990 WinTV-PVR-350 (249USD) (iTVC15 chipset + radio) -980 WinTV-PVR-250 (149USD) (iTVC15 chipset) -880 WinTV-PVR-PCI (199USD) (KFIR chipset + bt878) -881 WinTV-PVR-USB -190 WinTV-GO -191 WinTV-GO-FM -404 WinTV -401 WinTV-radio -495 WinTV-Theater -602 WinTV-USB -621 WinTV-USB-FM -600 USB-Live -698 WinTV-HD -697 WinTV-D -564 WinTV-Nexus-S

### Deutsche Modelle:

-603 WinTV GO -719 WinTV Primio-FM -718 WinTV PCI-FM -497 WinTV Theater -569 WinTV USB -568 WinTV USB-FM -882 WinTV PVR -981 WinTV PVR 250 -891 WinTV-PVR-USB -541 WinTV Nova -488 WinTV Nova-Ci -564 WinTV-Nexus-s -727 WinTV-DVB-c -545 Common Interface -898 WinTV-Nova-USB

### UK models:

-607 WinTV Go -693,793 WinTV Primio FM -647,747 WinTV PCI FM -498 WinTV Theater -883 WinTV PVR -893 WinTV PVR USB (Duplicate entry) -566 WinTV USB (UK) -573 WinTV USB FM -429 Impact VCB (bt848) -600 USB Live (Video-In 1x Comp, 1xSVHS) -542 WinTV Nova -717 WinTV DVB-S -909 Nova-t PCI -893 Nova-t USB (Duplicate entry) -802 MyTV -804 MyView -809 MyVideo -872 MyTV2Go FM -546 WinTV Nova-S CI -543 WinTV Nova -907 Nova-S USB -908 Nova-T USB -717 WinTV Nexus-S -157 DEC3000-s Standalone + USB

### Spain:

-685 WinTV-Go -690 WinTV-PrimioFM -416 WinTV-PCI Nicam Estereo -677 WinTV-PCI-FM -699 WinTV-Theater -683 WinTV-USB -678 WinTV-USB-FM -983 WinTV-PVR-250 -883 WinTV-PVR-PCI -993 WinTV-PVR-350 -893 WinTV-PVR-USB -728 WinTV-DVB-C PCI -832 MyTV2Go -869 MyTV2Go-FM -805 MyVideo (USB)

## **Matrix-Vision**

### Models:

- MATRIX-Vision MV-Delta
- MATRIX-Vision MV-Delta 2
- MVsigma-SLC (Bt848)

## **Conceptronic (.net)**

### Models:

- TVCON FM, TV card w/ FM = CPH05x
- TVCON = CPH06x

### BestData

Models:

- HCC100 = VCC100rev1 + camera
- VCC100 rev1 (bt848)
- VCC100 rev2 (bt878)

### Gallant ([www.gallantcom.com](http://www.gallantcom.com)) [www.minton.com.tw](http://www.minton.com.tw)

Models:

- Intervision IV-510 (capture only bt8x8)
- Intervision IV-550 (bt8x8)
- Intervision IV-100 (zoran)
- Intervision IV-1000 (bt8x8)

### Asonic ([www.asonic.com.cn](http://www.asonic.com.cn)) (website down)

SkyEye tv 878

### Hoontech

878TV/FM

### Teppro ([www.itcteppro.com.tw](http://www.itcteppro.com.tw))

Models:

- ITC PCITV (Card Ver 1.0) “Teppro TV1/TVFM1 Card”
- ITC PCITV (Card Ver 2.0)
- ITC PCITV (Card Ver 3.0) = “PV-BT878P+ (REV.9D)”
- ITC PCITV (Card Ver 4.0)
- TEPPRO IV-550 (For BT848 Main Chip)
- ITC DSTTV (bt878, satellite)
- ITC VideoMaker (saa7146, StreamMachine sm2110, tvtuner) “PV-SM2210P+ (REV:1C)”

### **Kworld ([www.kworld.com.tw](http://www.kworld.com.tw))**

PC TV Station:

- KWORLD KW-TV878R TV (no radio)
- KWORLD KW-TV878RF TV (w/ radio)
- KWORLD KW-TVL878RF (low profile)
- KWORLD KW-TV713XRF (saa7134)

MPEG TV Station (same cards as above plus WinDVR Software MPEG en/decoder)

- KWORLD KW-TV878R -Pro TV (no Radio)
- KWORLD KW-TV878RF-Pro TV (w/ Radio)
- KWORLD KW-TV878R -Ultra TV (no Radio)
- KWORLD KW-TV878RF-Ultra TV (w/ Radio)

### **JTT/ Justy Corp.(<http://www.jtt.ne.jp/>)**

JTT-02 (JTT TV) “TV watchmate pro” (bt848)

### **ADS [www.adstech.com](http://www.adstech.com)**

Models:

- Channel Surfer TV ( CHX-950 )
- Channel Surfer TV+FM ( CHX-960FM )

### **AVEC [www.prochips.com](http://www.prochips.com)**

AVEC Intercapture (bt848, tea6320)

### **NoBrand**

TV Excel = Australian Name for “PV-BT878P+ 8E” or “878TV Rev.3\_”

### **Mach [www.machspeed.com](http://www.machspeed.com)**

Mach TV 878

### **Eline [www.eline-net.com/](http://www.eline-net.com/)**

Models:

- Eline Vision TVMaster / TVMaster FM (ELV-TVM/ ELV-TVM-FM) = LR26 (bt878)
- Eline Vision TVMaster-2000 (ELV-TVM-2000, ELV-TVM-2000-FM)= LR138 (saa713x)

### **Spirit**

- Spirit TV Tuner/Video Capture Card (bt848)

### **Boser [www.boser.com.tw](http://www.boser.com.tw)**

Models:

- HS-878 Mini PCI Capture Add-on Card
- HS-879 Mini PCI 3D Audio and Capture Add-on Card (w/ ES1938 Solo-1)

### **Satelco [www.citycom-gmbh.de](http://www.citycom-gmbh.de), [www.satelco.de](http://www.satelco.de)**

Models:

- TV-FM =KNC1 saa7134
- Standard PCI (DVB-S) = Technotrend Budget
- Standard PCI (DVB-S) w/ CI
- Satelco Highend PCI (DVB-S) = Technotrend Premium

### **Sensoray [www.sensoray.com](http://www.sensoray.com)**

Models:

- Sensoray 311 (PC/104 bus)
- Sensoray 611 (PCI)

## **CEI (Chartered Electronics Industries Pte Ltd [CEI] [FCC ID HBY])**

Models:

- TV Tuner - HBY-33A-RAFFLES Brooktree Bt848KPF + Philips
- TV Tuner MG9910 - HBY33A-TVO CEI + Philips SAA7110 + OKI M548262 + ST STV8438CV
- Primetime TV (ISA)
  - acquired by Singapore Technologies
  - now operating as Chartered Semiconductor Manufacturing
  - Manufacturer of video cards is listed as:
    - \* Cogent Electronics Industries [CEI]

## **AITech**

Models:

- Wavewatcher TV (ISA)
- AITech WaveWatcher TV-PCI = can be LR26 (Bt848) or LR50 (BT878)
- WaveWatcher TVR-202 TV/FM Radio Card (ISA)

## **MAXRON**

Maxron MaxTV/FM Radio (KW-TV878-FNT) = Kworld or JW-TV878-FBK

## **[www.ids-imaging.de](http://www.ids-imaging.de)**

Models:

- Falcon Series (capture only)

In USA: <http://www.theimagingsource.com/> - DFG/LC1

## **[www.sknet-web.co.jp](http://www.sknet-web.co.jp)**

SKnet Monster TV (saa7134)

### **A-Max [www.amaxhk.com](http://www.amaxhk.com) (Colormax, Amax, Napa)**

APAC Viewcomp 878

### **Cybertainment**

Models:

- CyberMail AV Video Email Kit w/ PCI Capture Card (capture only)
- CyberMail Xtreme

These are Flyvideo

### **VCR (<http://www.vcrinc.com/>)**

Video Catcher 16

### **Twinhan**

Models:

- DST Card/DST-IP (bt878, twinhan asic) VP-1020 - Sold as:
  - KWorld DVBS Satellite TV-Card
  - Powercolor DSTV Satellite Tuner Card
  - Prolink Pixelview DTV2000
  - Provideo PV-911 Digital Satellite TV Tuner Card With Common Interface ?
- DST-CI Card (DVB Satellite) VP-1030
- DCT Card (DVB cable)

### **MSI**

Models:

- MSI [TV@nywhere](#) Tuner Card (MS-8876) (CX23881/883) Not Bt878 compatible.
- MS-8401 DVB-S

**Focus [www.focusinfo.com](http://www.focusinfo.com)**

InVideo PCI (bt878)

**Sdisilk [www.sdisilk.com/](http://www.sdisilk.com/)**

Models:

- SDI Silk 100
- SDI Silk 200 SDI Input Card

**[www.euresys.com](http://www.euresys.com)**

PICOLO series

**PMC/Pace**

[www.pacecom.co.uk](http://www.pacecom.co.uk) website closed

**Mercury [www.kobian.com](http://www.kobian.com) (UK and FR)**

Models:

- LR50
- LR138RBG-Rx == LR138

**TEC sound**

TV-Mate = Zoltrix VP-8482

Though educated googling found: [www.techmakers.com](http://www.techmakers.com)

(package and manuals don' t have any other manufacturer info) TecSound

**Lorenzen [www.lorenzen.de](http://www.lorenzen.de)**

SL DVB-S PCI = Technotrend Budget PCI (su1278 or bsru version)

### Origo (.uk) [www.origo2000.com](http://www.origo2000.com)

PC TV Card = LR50

### I/O Magic [www.iomagic.com](http://www.iomagic.com)

PC PVR - Desktop TV Personal Video Recorder DR-PCTV100 = Pinnacle ROB2D-51009464 4.0 + Cyberlink PowerVCR II

### Arowana

TV-Karte / Poso Power TV (?) = Zoltrix VP-8482 (?)

### iTVC15 boards

kuroutoshikou.com iTVC15 yuan.com MPG160 PCI TV (Internal PCI MPEG2 encoder card plus TV-tuner)

### Asus [www.asuscom.com](http://www.asuscom.com)

Models:

- Asus TV Tuner Card 880 NTSC (low profile, cx23880)
- Asus TV (saa7134)

### Hoontech

<http://www.hoontech.de/>

- HART Vision 848 (H-ART Vision 848)
- HART Vision 878 (H-Art Vision 878)

### Chips used at bttv devices

- all boards:
  - Brooktree Bt848/848A/849/878/879: video capture chip
- Board specific
  - Miro PCTV:
    - \* Philips or Temic Tuner
  - Hauppauge Win/TV pci (version 405):
    - \* Microchip 24LC02B or Philips 8582E2Y:
      - 256 Byte EEPROM with configuration information

- I2C 0xa0-0xa1, (24LC02B also responds to 0xa2-0xaf)
- \* Philips SAA5246AGP/E: Videotext decoder chip, I2C 0x22-0x23
- \* TDA9800: sound decoder
- \* Winbond W24257AS-35: 32Kx8 CMOS static RAM (Videotext buffer mem)
- \* 14052B: analog switch for selection of sound source
- PAL:
  - TDA5737: VHF, hyperband and UHF mixer/oscillator for TV and VCR 3-band tuners
  - TSA5522: 1.4 GHz I2C-bus controlled synthesizer, I2C 0xc2-0xc3
- NTSC:
  - TDA5731: VHF, hyperband and UHF mixer/oscillator for TV and VCR 3-band tuners
  - TSA5518: no datasheet available on Philips site
- STB TV pci:
  - ???
  - if you want better support for STB cards send me info! Look at the board! What chips are on it?

## Specs

Philips <http://www.Semiconductors.COM/pip/>

Conexant <http://www.conexant.com/>

Micronas <http://www.micronas.com/en/home/index.html>

## Thanks

Many thanks to:

- Markus Schroeder <[schroedm@uni-duesseldorf.de](mailto:schroedm@uni-duesseldorf.de)> for information on the Bt848 and tuner programming and his control program xtvc.
- Martin Buck <[martin-2.buck@student.uni-ulm.de](mailto:martin-2.buck@student.uni-ulm.de)> for his great Videotext package.
- Gerd Hoffmann for the MSP3400 support and the modular I2C, tuner, ...support.
- MATRIX Vision for giving us 2 cards for free, which made support of single crystal operation possible.
- MIRO for providing a free PCTV card and detailed information about the components on their cards. (E.g. how the tuner type is detected) Without their card I could not have debugged the NTSC mode.

- Hauppauge for telling how the sound input is selected and what components they do and will use on their radio cards. Also many thanks for faxing me the FM1216 data sheet.

### Contributors

**Michael Chu** <[mmchu@pobox.com](mailto:mmchu@pobox.com)> AverMedia fix and more flexible card recognition

**Alan Cox** <[alan@lxorguk.ukuu.org.uk](mailto:alan@lxorguk.ukuu.org.uk)> Video4Linux interface and 2.1.x kernel adaptation

**Chris Kleitsch** Hardware I2C

**Gerd Hoffmann** Radio card (ITT sound processor)

bigfoot <[bigfoot@net-way.net](mailto:bigfoot@net-way.net)>

**Ragnar Hojland Espinosa** <[ragnar@macula.net](mailto:ragnar@macula.net)> ConferenceTV card

- **many more (please mail me if you are missing in this list and would like to be mentioned)**

### The `cafe_ccic` driver

Author: Jonathan Corbet <[corbet@lwn.net](mailto:corbet@lwn.net)>

### Introduction

“`cafe_ccic`” is a driver for the Marvell 88ALP01 “cafe” CMOS camera controller. This is the controller found in first-generation OLPC systems, and this driver was written with support from the OLPC project.

Current status: the core driver works. It can generate data in YUV422, RGB565, and RGB444 formats. (Anybody looking at the code will see RGB32 as well, but that is a debugging aid which will be removed shortly). VGA and QVGA modes work; CIF is there but the colors remain funky. Only the OV7670 sensor is known to work with this controller at this time.

To try it out: either of these commands will work:

```
$ mplayer tv:// -tv driver=v4l2:width=640:height=480 -nosound
$ mplayer tv:// -tv driver=v4l2:width=640:height=480:outfmt=bgr16 -nosound
```

The “`xawtv`” utility also works; `gqcam` does not, for unknown reasons.

## Load time options

There are a few load-time options, most of which can be changed after loading via sysfs as well:

- `alloc_bufs_at_load`: Normally, the driver will not allocate any DMA buffers until the time comes to transfer data. If this option is set, then worst-case-sized buffers will be allocated at module load time. This option nails down the memory for the life of the module, but perhaps decreases the chances of an allocation failure later on.
- `dma_buf_size`: The size of DMA buffers to allocate. Note that this option is only consulted for load-time allocation; when buffers are allocated at run time, they will be sized appropriately for the current camera settings.
- `n_dma_bufs`: The controller can cycle through either two or three DMA buffers. Normally, the driver tries to use three buffers; on faster systems, however, it will work well with only two.
- `min_buffers`: The minimum number of streaming I/O buffers that the driver will consent to work with. Default is one, but, on slower systems, better behavior with mplayer can be achieved by setting to a higher value (like six).
- `max_buffers`: The maximum number of streaming I/O buffers; default is ten. That number was carefully picked out of a hat and should not be assumed to actually mean much of anything.
- `flip`: If this boolean parameter is set, the sensor will be instructed to invert the video image. Whether it makes sense is determined by how your particular camera is mounted.

## The cpia2 driver

Authors: Peter Pregler <[Peter\\_Pregler@email.com](mailto:Peter_Pregler@email.com)>, Scott J. Bertin <[scottbertin@yahoo.com](mailto:scottbertin@yahoo.com)>, and Jarl Totland <[Jarl.Totland@bdc.no](mailto:Jarl.Totland@bdc.no)> for the original cpia driver, which this one was modelled from.

## Introduction

This is a driver for STMicroelectronics' s CPiA2 (second generation Colour Processor Interface ASIC) based cameras. This camera outputs an MJPEG stream at up to vga size. It implements the Video4Linux interface as much as possible. Since the V4L interface does not support compressed formats, only an mjpeg enabled application can be used with the camera. We have modified the gqcam application to view this stream.

The driver is implemented as two kernel modules. The cpia2 module contains the camera functions and the V4L interface. The cpia2\_usb module contains usb specific functions. The main reason for this was the size of the module was getting out of hand, so I separated them. It is not likely that there will be a parallel port version.

### Features

- Supports cameras with the Vision stv6410 (CIF) and stv6500 (VGA) cmos sensors. I only have the vga sensor, so can't test the other.
- Image formats: VGA, QVGA, CIF, QCIF, and a number of sizes in between. VGA and QVGA are the native image sizes for the VGA camera. CIF is done in the coprocessor by scaling QVGA. All other sizes are done by clipping.
- Palette: YCrCb, compressed with MJPEG.
- Some compression parameters are settable.
- Sensor framerate is adjustable (up to 30 fps CIF, 15 fps VGA).
- Adjust brightness, color, contrast while streaming.
- Flicker control settable for 50 or 60 Hz mains frequency.

### Making and installing the stv672 driver modules

#### Requirements

Video4Linux must be either compiled into the kernel or available as a module. Video4Linux2 is automatically detected and made available at compile time.

#### Setup

Use `modprobe cpia2` to load and `modprobe -r cpia2` to unload. This may be done automatically by your distribution.

#### Driver options

Option	Description
<code>video_nr</code>	video device to register (0=/dev/video0, etc) range -1 to 64. default is -1 (first available) If you have more than 1 camera, this MUST be -1.
<code>buffer_size</code>	Size for each frame buffer in bytes (default 68k)
<code>num_buffers</code>	Number of frame buffers (1-32, default 3)
<code>alternate</code>	USB Alternate (2-7, default 7)
<code>flicker_freq</code>	Frequency for flicker reduction(50 or 60, default 60)
<code>flicker_mode</code>	0 to disable, or 1 to enable flicker reduction. (default 0). This is only effective if the camera uses a stv0672 coprocessor.

## Setting the options

If you are using modules, edit `/etc/modules.conf` and add an options line like this:

```
options cpia2 num_buffers=3 buffer_size=65535
```

If the driver is compiled into the kernel, at boot time specify them like this:

```
cpia2.num_buffers=3 cpia2.buffer_size=65535
```

## What buffer size should I use?

The maximum image size depends on the alternate you choose, and the frame rate achieved by the camera. If the compression engine is able to keep up with the frame rate, the maximum image size is given by the table below.

The compression engine starts out at maximum compression, and will increase image quality until it is close to the size in the table. As long as the compression engine can keep up with the frame rate, after a short time the images will all be about the size in the table, regardless of resolution.

At low alternate settings, the compression engine may not be able to compress the image enough and will reduce the frame rate by producing larger images.

The default of 68k should be good for most users. This will handle any alternate at frame rates down to 15fps. For lower frame rates, it may be necessary to increase the buffer size to avoid having frames dropped due to insufficient space.

Alternate	bytes/ms	15fps	30fps
2	128	8533	4267
3	384	25600	12800
4	640	42667	21333
5	768	51200	25600
6	896	59733	29867
7	1023	68200	34100

Table: Image size(bytes)

## How many buffers should I use?

For normal streaming, 3 should give the best results. With only 2, it is possible for the camera to finish sending one image just after a program has started reading the other. If this happens, the driver must drop a frame. The exception to this is if you have a heavily loaded machine. In this case use 2 buffers. You are probably not reading at the full frame rate. If the camera can send multiple images before a read finishes, it could overwrite the third buffer before the read finishes, leading to a corrupt image. Single and double buffering have extra checks to avoid overwriting.

### Using the camera

We are providing a modified `gqcam` application to view the output. In order to avoid confusion, here it is called `mview`. There is also the `qx5view` program which can also control the lights on the `qx5` microscope. `MJPEG Tools` (<http://mjpeg.sourceforge.net>) can also be used to record from the camera.

### The `cx88` driver

Author: Gerd Hoffmann

This is a v4l2 device driver for the `cx2388x` chip.

### Current status

#### video

- Works.
- Overlay isn't supported.

#### audio

- Works. The TV standard detection is made by the driver, as the hardware has bugs to auto-detect.
- audio data dma (i.e. recording without loopback cable to the sound card) is supported via `cx88-alsa`.

#### vbi

- Works.

### How to add support for new cards

The driver needs some config info for the TV cards. This stuff is in `cx88-cards.c`. If the driver doesn't work well you likely need a new entry for your card in that file. Check the kernel log (using `dmesg`) to see whenever the driver knows your card or not. There is a line like this one:

```
cx8800[0]: subsystem: 0070:3400, board: Hauppauge WinTV \
          34xxx models [card=1,autodetected]
```

If your card is listed as “board: UNKNOWN/GENERIC” it is unknown to the driver. What to do then?

- 1) Try upgrading to the latest snapshot, maybe it has been added meanwhile.
- 2) You can try to create a new entry yourself, have a look at `cx88-cards.c`. If that worked, mail me your changes as unified diff ( “diff -u” ).
- 3) Or you can mail me the config information. We need at least the following information to add the card:

- the PCI Subsystem ID ( “0070:3400” from the line above, “lspci -v” output is fine too).
- the tuner type used by the card. You can try to find one by trial-and-error using the tuner=<n> insmod option. If you know which one the card has you can also have a look at the list in CARDLIST.tuner

## **The VPBE V4L2 driver design**

### **Functional partitioning**

Consists of the following:

1. V4L2 display driver

Implements creation of video2 and video3 device nodes and provides v4l2 device interface to manage VID0 and VID1 layers.

2. Display controller

Loads up VENC, OSD and external encoders such as ths8200. It provides a set of API calls to V4L2 drivers to set the output/standards in the VENC or external sub devices. It also provides a device object to access the services from OSD subdevice using sub device ops. The connection of external encoders to VENC LCD controller port is done at init time based on default output and standard selection or at run time when application change the output through V4L2 IOCTLs.

When connected to an external encoder, vpbe controller is also responsible for setting up the interface between VENC and external encoders based on board specific settings (specified in board-xxx-evm.c). This allows interfacing external encoders such as ths8200. The setup\_if\_config() is implemented for this as well as configure\_venc() (part of the next patch) API to set timings in VENC for a specific display resolution. As of this patch series, the interconnection and enabling and setting of the external encoders is not present, and would be a part of the next patch series.

3. VENC subdevice module

Responsible for setting outputs provided through internal DACs and also setting timings at LCD controller port when external encoders are connected at the port or LCD panel timings required. When external encoder/LCD panel is connected, the timings for a specific standard/preset is retrieved from the board specific table and the values are used to set the timings in venc using non-standard timing mode.

Support LCD Panel displays using the VENC. For example to support a Logic PD display, it requires setting up the LCD controller port with a set of timings for the resolution supported and setting the dot clock. So we could add the available outputs as a board specific entry (i.e add the “LogicPD” output name to board-xxx-evm.c). A table of timings for various LCDs supported can be maintained in the board specific setup file to support various LCD displays. As of this patch a basic driver is present, and this support for external encoders and displays forms a part of the next patch series.

### 4. OSD module

OSD module implements all OSD layer management and hardware specific features. The VPBE module interacts with the OSD for enabling and disabling appropriate features of the OSD.

### Current status

A fully functional working version of the V4L2 driver is available. This driver has been tested with NTSC and PAL standards and buffer streaming.

### The Samsung S5P/EXYNOS4 FIMC driver

Copyright © 2012 - 2013 Samsung Electronics Co., Ltd.

The FIMC (Fully Interactive Mobile Camera) device available in Samsung SoC Application Processors is an integrated camera host interface, color space converter, image resizer and rotator. It's also capable of capturing data from LCD controller (FIMD) through the SoC internal writeback data path. There are multiple FIMC instances in the SoCs (up to 4), having slightly different capabilities, like pixel alignment constraints, rotator availability, LCD writeback support, etc. The driver is located at `drivers/media/platform/exynos4-is` directory.

### Supported SoCs

S5PC100 (mem-to-mem only), S5PV210, EXYNOS4210

### Supported features

- camera parallel interface capture (ITU-R.BT601/565);
- camera serial interface capture (MIPI-CSI2);
- memory-to-memory processing (color space conversion, scaling, mirror and rotation);
- dynamic pipeline re-configuration at runtime (re-attachment of any FIMC instance to any parallel video input or any MIPI-CSI front-end);
- runtime PM and system wide suspend/resume

## **Not currently supported**

- LCD writeback input
- per frame clock gating (mem-to-mem)

## **User space interfaces**

### **Media device interface**

The driver supports Media Controller API as defined at `media_controller`. The media device driver name is “SAMSUNG S5P FIMC” .

The purpose of this interface is to allow changing assignment of FIMC instances to the SoC peripheral camera input at runtime and optionally to control internal connections of the MIPI-CSIS device(s) to the FIMC entities.

The media device interface allows to configure the SoC for capturing image data from the sensor through more than one FIMC instance (e.g. for simultaneous viewfinder and still capture setup).

Reconfiguration is done by enabling/disabling media links created by the driver during initialization. The internal device topology can be easily discovered through media entity and links enumeration.

### **Memory-to-memory video node**

V4L2 memory-to-memory interface at `/dev/video?` device node. This is standalone video device, it has no media pads. However please note the mem-to-mem and capture video node operation on same FIMC instance is not allowed. The driver detects such cases but the applications should prevent them to avoid an undefined behaviour.

### **Capture video node**

The driver supports V4L2 Video Capture Interface as defined at `devices`.

At the capture and mem-to-mem video nodes only the multi-planar API is supported. For more details see: `planar-apis`.

### **Camera capture subdevs**

Each FIMC instance exports a sub-device node (`/dev/v4l-subdev?`), a sub-device node is also created per each available and enabled at the platform level MIPI-CSI receiver device (currently up to two).

### sysfs

In order to enable more precise camera pipeline control through the sub-device API the driver creates a sysfs entry associated with “s5p-fimc-md” platform device. The entry path is: /sys/platform/devices/s5p-fimc-md/subdev\_conf\_mode.

In typical use case there could be a following capture pipeline configuration: sensor subdev -> mipi-csi subdev -> fimc subdev -> video node

When we configure these devices through sub-device API at user space, the configuration flow must be from left to right, and the video node is configured as last one.

When we don't use sub-device user space API the whole configuration of all devices belonging to the pipeline is done at the video node driver. The sysfs entry allows to instruct the capture node driver not to configure the sub-devices (format, crop), to avoid resetting the subdevs' configuration when the last configuration steps at the video node is performed.

For full sub-device control support (subdevs configured at user space before starting streaming):

```
# echo "sub-dev" > /sys/platform/devices/s5p-fimc-md/subdev_conf_mode
```

For V4L2 video node control only (subdevs configured internally by the host driver):

```
# echo "vid-dev" > /sys/platform/devices/s5p-fimc-md/subdev_conf_mode
```

This is a default option.

## 5. Device mapping to video and subdev device nodes

There are associated two video device nodes with each device instance in hardware - video capture and mem-to-mem and additionally a subdev node for more precise FIMC capture subsystem control. In addition a separate v4l2 sub-device node is created per each MIPI-CSIS device.

How to find out which /dev/video? or /dev/v4l-subdev? is assigned to which device?

You can either grep through the kernel log to find relevant information, i.e.

```
# dmesg | grep -i fimc
```

(note that udev, if present, might still have rearranged the video nodes),

or retrieve the information from /dev/media? with help of the media-ctl tool:

```
# media-ctl -p
```

## 7. Build

If the driver is built as a loadable kernel module (CONFIG\_VIDEO\_SAMSUNG\_S5P\_FIMC=m) two modules are created (in addition to the core v4l2 modules): s5p-fimc.ko and optional s5p-csis.ko (MIPI-CSI receiver subdev).

### i.MX Video Capture Driver

#### Introduction

The Freescale i.MX5/6 contains an Image Processing Unit (IPU), which handles the flow of image frames to and from capture devices and display devices.

For image capture, the IPU contains the following internal subunits:

- Image DMA Controller (IDMAC)
- Camera Serial Interface (CSI)
- Image Converter (IC)
- Sensor Multi-FIFO Controller (SMFC)
- Image Rotator (IRT)
- Video De-Interlacing or Combining Block (VDIC)

The IDMAC is the DMA controller for transfer of image frames to and from memory. Various dedicated DMA channels exist for both video capture and display paths. During transfer, the IDMAC is also capable of vertical image flip, 8x8 block transfer (see IRT description), pixel component re-ordering (for example UYVY to YUYV) within the same colorspace, and packed <-> planar conversion. The IDMAC can also perform a simple de-interlacing by interweaving even and odd lines during transfer (without motion compensation which requires the VDIC).

The CSI is the backend capture unit that interfaces directly with camera sensors over Parallel, BT.656/1120, and MIPI CSI-2 buses.

The IC handles color-space conversion, resizing (downscaling and upscaling), horizontal flip, and 90/270 degree rotation operations.

There are three independent “tasks” within the IC that can carry out conversions concurrently: pre-process encoding, pre-process viewfinder, and post-processing. Within each task, conversions are split into three sections: downsizing section, main section (upsizing, flip, colorspace conversion, and graphics plane combining), and rotation section.

The IPU time-shares the IC task operations. The time-slice granularity is one burst of eight pixels in the downsizing section, one image line in the main processing section, one image frame in the rotation section.

The SMFC is composed of four independent FIFOs that each can transfer captured frames from sensors directly to memory concurrently via four IDMAC channels.

The IRT carries out 90 and 270 degree image rotation operations. The rotation operation is carried out on 8x8 pixel blocks at a time. This operation is supported

by the IDMAC which handles the 8x8 block transfer along with block reordering, in coordination with vertical flip.

The VDIC handles the conversion of interlaced video to progressive, with support for different motion compensation modes (low, medium, and high motion). The deinterlaced output frames from the VDIC can be sent to the IC pre-process viewfinder task for further conversions. The VDIC also contains a Combiner that combines two image planes, with alpha blending and color keying.

In addition to the IPU internal subunits, there are also two units outside the IPU that are also involved in video capture on i.MX:

- MIPI CSI-2 Receiver for camera sensors with the MIPI CSI-2 bus interface. This is a Synopsys DesignWare core.
- Two video multiplexers for selecting among multiple sensor inputs to send to a CSI.

For more info, refer to the latest versions of the i.MX5/6 reference manuals<sup>1</sup> and<sup>2</sup>.

### Features

Some of the features of this driver include:

- Many different pipelines can be configured via media controller API, that correspond to the hardware video capture pipelines supported in the i.MX.
- Supports parallel, BT.565, and MIPI CSI-2 interfaces.
- Concurrent independent streams, by configuring pipelines to multiple video capture interfaces using independent entities.
- Scaling, color-space conversion, horizontal and vertical flip, and image rotation via IC task subdevs.
- Many pixel formats supported (RGB, packed and planar YUV, partial planar YUV).
- The VDIC subdev supports motion compensated de-interlacing, with three motion compensation modes: low, medium, and high motion. Pipelines are defined that allow sending frames to the VDIC subdev directly from the CSI. There is also support in the future for sending frames to the VDIC from memory buffers via a output/mem2mem devices.
- Includes a Frame Interval Monitor (FIM) that can correct vertical sync problems with the ADV718x video decoders.

---

<sup>1</sup> <http://www.nxp.com/assets/documents/data/en/reference-manuals/IMX6DQRM.pdf>

<sup>2</sup> <http://www.nxp.com/assets/documents/data/en/reference-manuals/IMX6SDLRM.pdf>

## Topology

The following shows the media topologies for the i.MX6Q SabreSD and i.MX6Q SabreAuto. Refer to these diagrams in the entity descriptions in the next section.

The i.MX5/6 topologies can differ upstream from the IPUv3 CSI video multiplexers, but the internal IPUv3 topology downstream from there is common to all i.MX5/6 platforms. For example, the SabreSD, with the MIPI CSI-2 OV5640 sensor, requires the i.MX6 MIPI CSI-2 receiver. But the SabreAuto has only the ADV7180 decoder on a parallel bt.656 bus, and therefore does not require the MIPI CSI-2 receiver, so it is missing in its graph.

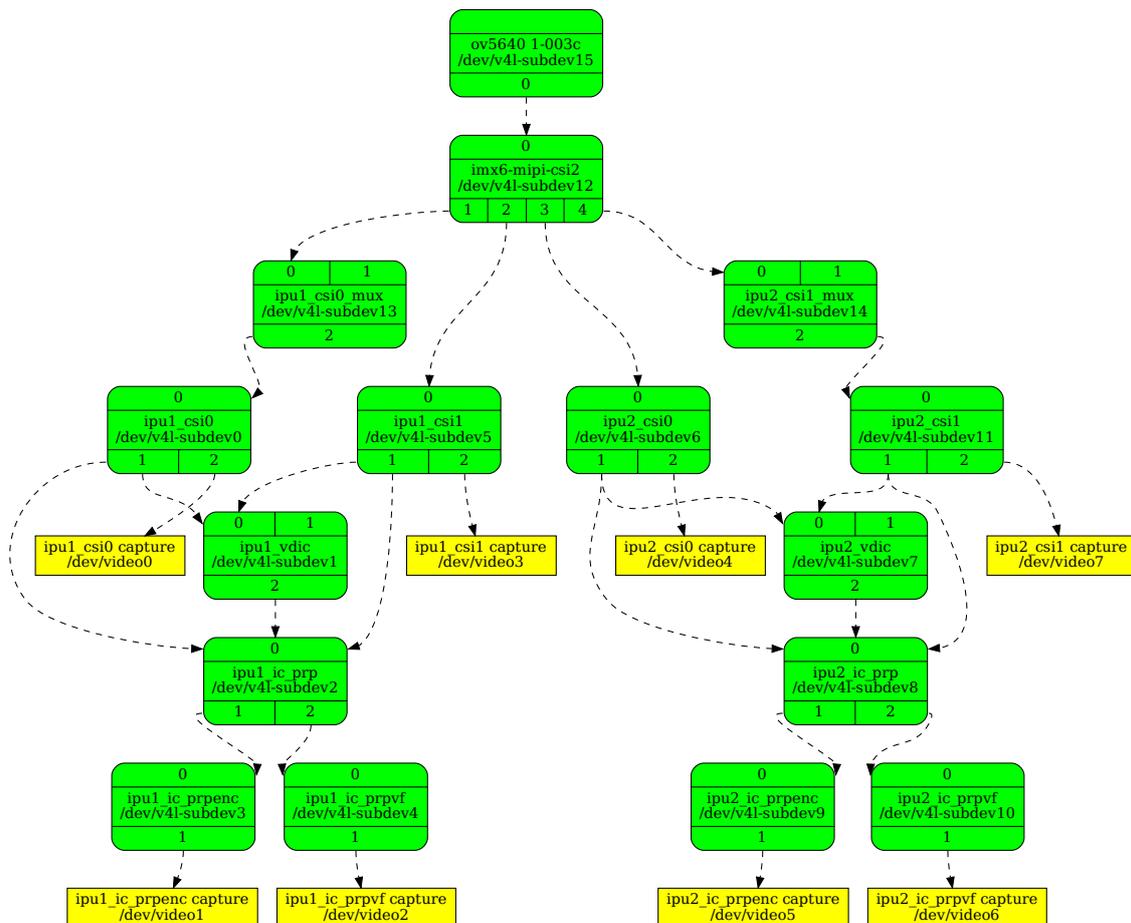


Fig. 1: Media pipeline graph on i.MX6Q SabreSD



## Entities

### **imx6-mipi-csi2**

This is the MIPI CSI-2 receiver entity. It has one sink pad to receive the MIPI CSI-2 stream (usually from a MIPI CSI-2 camera sensor). It has four source pads, corresponding to the four MIPI CSI-2 demuxed virtual channel outputs. Multiple source pads can be enabled to independently stream from multiple virtual channels.

This entity actually consists of two sub-blocks. One is the MIPI CSI-2 core. This is a Synopsys Designware MIPI CSI-2 core. The other sub-block is a “CSI-2 to IPU gasket” . The gasket acts as a demultiplexer of the four virtual channels streams, providing four separate parallel buses containing each virtual channel that are routed to CSIs or video multiplexers as described below.

On i.MX6 solo/dual-lite, all four virtual channel buses are routed to two video multiplexers. Both CSI0 and CSI1 can receive any virtual channel, as selected by the video multiplexers.

On i.MX6 Quad, virtual channel 0 is routed to IPU1-CSI0 (after selected by a video mux), virtual channels 1 and 2 are hard-wired to IPU1-CSI1 and IPU2-CSI0, respectively, and virtual channel 3 is routed to IPU2-CSI1 (again selected by a video mux).

### **ipuX\_csiY\_mux**

These are the video multiplexers. They have two or more sink pads to select from either camera sensors with a parallel interface, or from MIPI CSI-2 virtual channels from imx6-mipi-csi2 entity. They have a single source pad that routes to a CSI (ipuX\_csiY entities).

On i.MX6 solo/dual-lite, there are two video mux entities. One sits in front of IPU1-CSI0 to select between a parallel sensor and any of the four MIPI CSI-2 virtual channels (a total of five sink pads). The other mux sits in front of IPU1-CSI1, and again has five sink pads to select between a parallel sensor and any of the four MIPI CSI-2 virtual channels.

On i.MX6 Quad, there are two video mux entities. One sits in front of IPU1-CSI0 to select between a parallel sensor and MIPI CSI-2 virtual channel 0 (two sink pads). The other mux sits in front of IPU2-CSI1 to select between a parallel sensor and MIPI CSI-2 virtual channel 3 (two sink pads).

### **ipuX\_csiY**

These are the CSI entities. They have a single sink pad receiving from either a video mux or from a MIPI CSI-2 virtual channel as described above.

This entity has two source pads. The first source pad can link directly to the ipuX\_vdic entity or the ipuX\_ic\_prp entity, using hardware links that require no IDMAC memory buffer transfer.

When the direct source pad is routed to the `ipuX_ic_prp` entity, frames from the CSI can be processed by one or both of the IC pre-processing tasks.

When the direct source pad is routed to the `ipuX_vdic` entity, the VDIC will carry out motion-compensated de-interlace using “high motion” mode (see description of `ipuX_vdic` entity).

The second source pad sends video frames directly to memory buffers via the SMFC and an IDMAC channel, bypassing IC pre-processing. This source pad is routed to a capture device node, with a node name of the format “`ipuX_csiY capture`” .

Note that since the IDMAC source pad makes use of an IDMAC channel, pixel reordering within the same colorspace can be carried out by the IDMAC channel. For example, if the CSI sink pad is receiving in UYVY order, the capture device linked to the IDMAC source pad can capture in YUYV order. Also, if the CSI sink pad is receiving a packed YUV format, the capture device can capture a planar YUV format such as YUV420.

The IDMAC channel at the IDMAC source pad also supports simple interweave without motion compensation, which is activated if the source pad’s field type is sequential top-bottom or bottom-top, and the requested capture interface field type is set to interlaced (t-b, b-t, or unqualified interlaced). The capture interface will enforce the same field order as the source pad field order (interlaced-bt if source pad is seq-bt, interlaced-tb if source pad is seq-tb).

For events produced by `ipuX_csiY`, see `ref:imx_api_ipuX_csiY`.

### Cropping in `ipuX_csiY`

The CSI supports cropping the incoming raw sensor frames. This is implemented in the `ipuX_csiY` entities at the sink pad, using the crop selection subdev API.

The CSI also supports fixed divide-by-two downscaling independently in width and height. This is implemented in the `ipuX_csiY` entities at the sink pad, using the compose selection subdev API.

The output rectangle at the `ipuX_csiY` source pad is the same as the compose rectangle at the sink pad. So the source pad rectangle cannot be negotiated, it must be set using the compose selection API at sink pad (if /2 downscale is desired, otherwise source pad rectangle is equal to incoming rectangle).

To give an example of crop and /2 downscale, this will crop a 1280x960 input frame to 640x480, and then /2 downscale in both dimensions to 320x240 (assumes `ipu1_csi0` is linked to `ipu1_csi0_mux`):

```
media-ctl -V "'ipu1_csi0_mux':2[fmt:UYVY2X8/1280x960]"
media-ctl -V "'ipu1_csi0':0[crop:(0,0)/640x480]"
media-ctl -V "'ipu1_csi0':0[compose:(0,0)/320x240]"
```

## Frame Skipping in ipuX\_csiY

The CSI supports frame rate decimation, via frame skipping. Frame rate decimation is specified by setting the frame intervals at sink and source pads. The ipuX\_csiY entity then applies the best frame skip setting to the CSI to achieve the desired frame rate at the source pad.

The following example reduces an assumed incoming 60 Hz frame rate by half at the IDMAC output source pad:

```
media-ctl -V "'ipu1_csi0':0[fmt:UYVY2X8/640x480@1/60]"
media-ctl -V "'ipu1_csi0':2[fmt:UYVY2X8/640x480@1/30]"
```

## Frame Interval Monitor in ipuX\_csiY

See ref:imx\_api\_FIM.

## ipuX\_vdic

The VDIC carries out motion compensated de-interlacing, with three motion compensation modes: low, medium, and high motion. The mode is specified with the menu control V4L2\_CID\_DEINTERLACING\_MODE. The VDIC has two sink pads and a single source pad.

The direct sink pad receives from an ipuX\_csiY direct pad. With this link the VDIC can only operate in high motion mode.

When the IDMAC sink pad is activated, it receives from an output or mem2mem device node. With this pipeline, the VDIC can also operate in low and medium modes, because these modes require receiving frames from memory buffers. Note that an output or mem2mem device is not implemented yet, so this sink pad currently has no links.

The source pad routes to the IC pre-processing entity ipuX\_ic\_prp.

## ipuX\_ic\_prp

This is the IC pre-processing entity. It acts as a router, routing data from its sink pad to one or both of its source pads.

This entity has a single sink pad. The sink pad can receive from the ipuX\_csiY direct pad, or from ipuX\_vdic.

This entity has two source pads. One source pad routes to the pre-process encode task entity (ipuX\_ic\_prpenc), the other to the pre-process viewfinder task entity (ipuX\_ic\_prpvf). Both source pads can be activated at the same time if the sink pad is receiving from ipuX\_csiY. Only the source pad to the pre-process viewfinder task entity can be activated if the sink pad is receiving from ipuX\_vdic (frames from the VDIC can only be processed by the pre-process viewfinder task).

### ipuX\_ic\_prpenc

This is the IC pre-processing encode entity. It has a single sink pad from ipuX\_ic\_prp, and a single source pad. The source pad is routed to a capture device node, with a node name of the format “ipuX\_ic\_prpenc capture” .

This entity performs the IC pre-process encode task operations: color-space conversion, resizing (downscaling and upscaling), horizontal and vertical flip, and 90/270 degree rotation. Flip and rotation are provided via standard V4L2 controls.

Like the ipuX\_csiY IDMAC source, this entity also supports simple de-interlace without motion compensation, and pixel reordering.

### ipuX\_ic\_prpvf

This is the IC pre-processing viewfinder entity. It has a single sink pad from ipuX\_ic\_prp, and a single source pad. The source pad is routed to a capture device node, with a node name of the format “ipuX\_ic\_prpvf capture” .

This entity is identical in operation to ipuX\_ic\_prpenc, with the same resizing and CSC operations and flip/rotation controls. It will receive and process de-interlaced frames from the ipuX\_vdic if ipuX\_ic\_prp is receiving from ipuX\_vdic.

Like the ipuX\_csiY IDMAC source, this entity supports simple interweaving without motion compensation. However, note that if the ipuX\_vdic is included in the pipeline (ipuX\_ic\_prp is receiving from ipuX\_vdic), it's not possible to use interweave in ipuX\_ic\_prpvf, since the ipuX\_vdic has already carried out de-interlacing (with motion compensation) and therefore the field type output from ipuX\_vdic can only be none (progressive).

## Capture Pipelines

The following describe the various use-cases supported by the pipelines.

The links shown do not include the backend sensor, video mux, or mipi csi-2 receiver links. This depends on the type of sensor interface (parallel or mipi csi-2). So these pipelines begin with:

```
sensor -> ipuX_csiY_mux -> ...
```

for parallel sensors, or:

```
sensor -> imx6-mipi-csi2 -> (ipuX_csiY_mux) -> ...
```

for mipi csi-2 sensors. The imx6-mipi-csi2 receiver may need to route to the video mux (ipuX\_csiY\_mux) before sending to the CSI, depending on the mipi csi-2 virtual channel, hence ipuX\_csiY\_mux is shown in parenthesis.

### **Unprocessed Video Capture:**

Send frames directly from sensor to camera device interface node, with no conversions, via ipuX\_csiY IDMAC source pad:

```
-> ipuX_csiY:2 -> ipuX_csiY capture
```

### **IC Direct Conversions:**

This pipeline uses the preprocess encode entity to route frames directly from the CSI to the IC, to carry out scaling up to 1024x1024 resolution, CSC, flipping, and image rotation:

```
-> ipuX_csiY:1 -> 0:ipuX_ic_prp:1 -> 0:ipuX_ic_prpenc:1 -> ipuX_ic_prpenc capture
```

### **Motion Compensated De-interlace:**

This pipeline routes frames from the CSI direct pad to the VDIC entity to support motion-compensated de-interlacing (high motion mode only), scaling up to 1024x1024, CSC, flip, and rotation:

```
-> ipuX_csiY:1 -> 0:ipuX_vdic:2 -> 0:ipuX_ic_prp:2 -> 0:ipuX_ic_prpvf:1 -> ipuX_ic_prpvf capture
```

### **Usage Notes**

To aid in configuration and for backward compatibility with V4L2 applications that access controls only from video device nodes, the capture device interfaces inherit controls from the active entities in the current pipeline, so controls can be accessed either directly from the subdev or from the active capture device interface. For example, the FIM controls are available either from the ipuX\_csiY subdevs or from the active capture device.

The following are specific usage notes for the Sabre\* reference boards:

#### **i.MX6Q SabreLite with OV5642 and OV5640**

This platform requires the OmniVision OV5642 module with a parallel camera interface, and the OV5640 module with a MIPI CSI-2 interface. Both modules are available from Boundary Devices:

- [https://boundarydevices.com/product/nit6x\\_5mp](https://boundarydevices.com/product/nit6x_5mp)
- [https://boundarydevices.com/product/nit6x\\_5mp\\_mipi](https://boundarydevices.com/product/nit6x_5mp_mipi)

Note that if only one camera module is available, the other sensor node can be disabled in the device tree.

The OV5642 module is connected to the parallel bus input on the i.MX internal video mux to IPU1 CSI0. It's i2c bus connects to i2c bus 2.

The MIPI CSI-2 OV5640 module is connected to the i.MX internal MIPI CSI-2 receiver, and the four virtual channel outputs from the receiver are routed as follows: vc0 to the IPU1 CSI0 mux, vc1 directly to IPU1 CSI1, vc2 directly to IPU2 CSI0, and vc3 to the IPU2 CSI1 mux. The OV5640 is also connected to i2c bus 2 on the SabreLite, therefore the OV5642 and OV5640 must not share the same i2c slave address.

The following basic example configures unprocessed video capture pipelines for both sensors. The OV5642 is routed to ipu1\_csi0, and the OV5640, transmitting on MIPI CSI-2 virtual channel 1 (which is imx6-mipi-csi2 pad 2), is routed to ipu1\_csi1. Both sensors are configured to output 640x480, and the OV5642 outputs YUYV2X8, the OV5640 UYVY2X8:

```
# Setup links for OV5642
media-ctl -l "'ov5642 1-0042':0 -> 'ipu1_csi0_mux':1[1]"
media-ctl -l "'ipu1_csi0_mux':2 -> 'ipu1_csi0':0[1]"
media-ctl -l "'ipu1_csi0':2 -> 'ipu1_csi0 capture':0[1]"
# Setup links for OV5640
media-ctl -l "'ov5640 1-0040':0 -> 'imx6-mipi-csi2':0[1]"
media-ctl -l "'imx6-mipi-csi2':2 -> 'ipu1_csi1':0[1]"
media-ctl -l "'ipu1_csi1':2 -> 'ipu1_csi1 capture':0[1]"
# Configure pads for OV5642 pipeline
media-ctl -V "'ov5642 1-0042':0 [fmt:YUYV2X8/640x480 field:none]"
media-ctl -V "'ipu1_csi0_mux':2 [fmt:YUYV2X8/640x480 field:none]"
media-ctl -V "'ipu1_csi0':2 [fmt:AYUV32/640x480 field:none]"
# Configure pads for OV5640 pipeline
media-ctl -V "'ov5640 1-0040':0 [fmt:UYVY2X8/640x480 field:none]"
media-ctl -V "'imx6-mipi-csi2':2 [fmt:UYVY2X8/640x480 field:none]"
media-ctl -V "'ipu1_csi1':2 [fmt:AYUV32/640x480 field:none]"
```

Streaming can then begin independently on the capture device nodes “ipu1\_csi0 capture” and “ipu1\_csi1 capture”. The v4l2-ctl tool can be used to select any supported YUV pixelformat on the capture device nodes, including planar.

### i.MX6Q SabreAuto with ADV7180 decoder

On the i.MX6Q SabreAuto, an on-board ADV7180 SD decoder is connected to the parallel bus input on the internal video mux to IPU1 CSI0.

The following example configures a pipeline to capture from the ADV7180 video decoder, assuming NTSC 720x480 input signals, using simple interweave (unconverted and without motion compensation). The adv7180 must output sequential or alternating fields (field type ‘seq-bt’ for NTSC, or ‘alternate’):

```
# Setup links
media-ctl -l "'adv7180 3-0021':0 -> 'ipu1_csi0_mux':1[1]"
media-ctl -l "'ipu1_csi0_mux':2 -> 'ipu1_csi0':0[1]"
media-ctl -l "'ipu1_csi0':2 -> 'ipu1_csi0 capture':0[1]"
# Configure pads
media-ctl -V "'adv7180 3-0021':0 [fmt:UYVY2X8/720x480 field:seq-bt]"
media-ctl -V "'ipu1_csi0_mux':2 [fmt:UYVY2X8/720x480]"
media-ctl -V "'ipu1_csi0':2 [fmt:AYUV32/720x480]"
# Configure "ipu1_csi0 capture" interface (assumed at /dev/video4)
v4l2-ctl -d4 --set-fmt-video=field=interlaced_bt
```

Streaming can then begin on /dev/video4. The v4l2-ctl tool can also be used to select any supported YUV pixelformat on /dev/video4.

This example configures a pipeline to capture from the ADV7180 video decoder, assuming PAL 720x576 input signals, with Motion Compensated de-interlacing. The adv7180 must output sequential or alternating fields (field type 'seq-tb' for PAL, or 'alternate' ).

```
# Setup links
media-ctl -l "'adv7180 3-0021':0 -> 'ipu1_csi0_mux':1[1]"
media-ctl -l "'ipu1_csi0_mux':2 -> 'ipu1_csi0':0[1]"
media-ctl -l "'ipu1_csi0':1 -> 'ipu1_vdic':0[1]"
media-ctl -l "'ipu1_vdic':2 -> 'ipu1_ic_prp':0[1]"
media-ctl -l "'ipu1_ic_prp':2 -> 'ipu1_ic_prpvf':0[1]"
media-ctl -l "'ipu1_ic_prpvf':1 -> 'ipu1_ic_prpvf capture':0[1]"
# Configure pads
media-ctl -V "'adv7180 3-0021':0 [fmt:UYVY2X8/720x576 field:seq-tb]"
media-ctl -V "'ipu1_csi0_mux':2 [fmt:UYVY2X8/720x576]"
media-ctl -V "'ipu1_csi0':1 [fmt:AYUV32/720x576]"
media-ctl -V "'ipu1_vdic':2 [fmt:AYUV32/720x576 field:none]"
media-ctl -V "'ipu1_ic_prp':2 [fmt:AYUV32/720x576 field:none]"
media-ctl -V "'ipu1_ic_prpvf':1 [fmt:AYUV32/720x576 field:none]"
# Configure "ipu1_ic_prpvf capture" interface (assumed at /dev/video2)
v4l2-ctl -d2 --set-fmt-video=field=none
```

Streaming can then begin on /dev/video2. The v4l2-ctl tool can also be used to select any supported YUV pixelformat on /dev/video2.

This platform accepts Composite Video analog inputs to the ADV7180 on Ain1 (connector J42).

### i.MX6DL SabreAuto with ADV7180 decoder

On the i.MX6DL SabreAuto, an on-board ADV7180 SD decoder is connected to the parallel bus input on the internal video mux to IPU1 CSI0.

The following example configures a pipeline to capture from the ADV7180 video decoder, assuming NTSC 720x480 input signals, using simple interweave (unconverted and without motion compensation). The adv7180 must output sequential or alternating fields (field type 'seq-bt' for NTSC, or 'alternate' ):

```
# Setup links
media-ctl -l "'adv7180 4-0021':0 -> 'ipu1_csi0_mux':4[1]"
media-ctl -l "'ipu1_csi0_mux':5 -> 'ipu1_csi0':0[1]"
media-ctl -l "'ipu1_csi0':2 -> 'ipu1_csi0 capture':0[1]"
# Configure pads
media-ctl -V "'adv7180 4-0021':0 [fmt:UYVY2X8/720x480 field:seq-bt]"
media-ctl -V "'ipu1_csi0_mux':5 [fmt:UYVY2X8/720x480]"
media-ctl -V "'ipu1_csi0':2 [fmt:AYUV32/720x480]"
# Configure "ipu1_csi0 capture" interface (assumed at /dev/video0)
v4l2-ctl -d0 --set-fmt-video=field=interlaced_bt
```

Streaming can then begin on /dev/video0. The v4l2-ctl tool can also be used to select any supported YUV pixelformat on /dev/video0.

This example configures a pipeline to capture from the ADV7180 video decoder,

assuming PAL 720x576 input signals, with Motion Compensated de-interlacing. The adv7180 must output sequential or alternating fields (field type 'seq-tb' for PAL, or 'alternate' ).

```
# Setup links
media-ctl -l "'adv7180 4-0021':0 -> 'ipu1_csi0_mux':4[1]"
media-ctl -l "'ipu1_csi0_mux':5 -> 'ipu1_csi0':0[1]"
media-ctl -l "'ipu1_csi0':1 -> 'ipu1_vdic':0[1]"
media-ctl -l "'ipu1_vdic':2 -> 'ipu1_ic_prp':0[1]"
media-ctl -l "'ipu1_ic_prp':2 -> 'ipu1_ic_prpvf':0[1]"
media-ctl -l "'ipu1_ic_prpvf':1 -> 'ipu1_ic_prpvf capture':0[1]"
# Configure pads
media-ctl -v "'adv7180 4-0021':0 [fmt:UYVY2X8/720x576 field:seq-tb]"
media-ctl -v "'ipu1_csi0_mux':5 [fmt:UYVY2X8/720x576]"
media-ctl -v "'ipu1_csi0':1 [fmt:AYUV32/720x576]"
media-ctl -v "'ipu1_vdic':2 [fmt:AYUV32/720x576 field:none]"
media-ctl -v "'ipu1_ic_prp':2 [fmt:AYUV32/720x576 field:none]"
media-ctl -v "'ipu1_ic_prpvf':1 [fmt:AYUV32/720x576 field:none]"
# Configure "ipu1_ic_prpvf capture" interface (assumed at /dev/video2)
v4l2-ctl -d2 --set-fmt-video=field=none
```

Streaming can then begin on /dev/video2. The v4l2-ctl tool can also be used to select any supported YUV pixelformat on /dev/video2.

This platform accepts Composite Video analog inputs to the ADV7180 on Ain1 (connector J42).

### i.MX6Q SabreSD with MIPI CSI-2 OV5640

Similarly to i.MX6Q SabreLite, the i.MX6Q SabreSD supports a parallel interface OV5642 module on IPU1 CSI0, and a MIPI CSI-2 OV5640 module. The OV5642 connects to i2c bus 1 and the OV5640 to i2c bus 2.

The device tree for SabreSD includes OF graphs for both the parallel OV5642 and the MIPI CSI-2 OV5640, but as of this writing only the MIPI CSI-2 OV5640 has been tested, so the OV5642 node is currently disabled. The OV5640 module connects to MIPI connector J5. The NXP part number for the OV5640 module that connects to the SabreSD board is H120729.

The following example configures unprocessed video capture pipeline to capture from the OV5640, transmitting on MIPI CSI-2 virtual channel 0:

```
# Setup links
media-ctl -l "'ov5640 1-003c':0 -> 'imx6-mipi-csi2':0[1]"
media-ctl -l "'imx6-mipi-csi2':1 -> 'ipu1_csi0_mux':0[1]"
media-ctl -l "'ipu1_csi0_mux':2 -> 'ipu1_csi0':0[1]"
media-ctl -l "'ipu1_csi0':2 -> 'ipu1_csi0 capture':0[1]"
# Configure pads
media-ctl -v "'ov5640 1-003c':0 [fmt:UYVY2X8/640x480]"
media-ctl -v "'imx6-mipi-csi2':1 [fmt:UYVY2X8/640x480]"
media-ctl -v "'ipu1_csi0_mux':0 [fmt:UYVY2X8/640x480]"
media-ctl -v "'ipu1_csi0':0 [fmt:AYUV32/640x480]"
```

Streaming can then begin on "ipu1\_csi0 capture" node. The v4l2-ctl tool can be used to select any supported pixelformat on the capture device node.

To determine what is the /dev/video node correspondent to “ipu1\_csi0 capture” :

```
media-ctl -e "ipu1_csi0 capture"
/dev/video0
```

/dev/video0 is the streaming element in this case.

Starting the streaming via v4l2-ctl:

```
v4l2-ctl --stream-mmap -d /dev/video0
```

Starting the streaming via Gstreamer and sending the content to the display:

```
gst-launch-1.0 v4l2src device=/dev/video0 ! kmssink
```

The following example configures a direct conversion pipeline to capture from the OV5640, transmitting on MIPI CSI-2 virtual channel 0. It also shows colorspace conversion and scaling at IC output.

```
# Setup links
media-ctl -l "'ov5640 1-003c':0 -> 'imx6-mipi-csi2':0[1]"
media-ctl -l "'imx6-mipi-csi2':1 -> 'ipu1_csi0_mux':0[1]"
media-ctl -l "'ipu1_csi0_mux':2 -> 'ipu1_csi0':0[1]"
media-ctl -l "'ipu1_csi0':1 -> 'ipu1_ic_prp':0[1]"
media-ctl -l "'ipu1_ic_prp':1 -> 'ipu1_ic_prpenc':0[1]"
media-ctl -l "'ipu1_ic_prpenc':1 -> 'ipu1_ic_prpenc capture':0[1]"
# Configure pads
media-ctl -V "'ov5640 1-003c':0 [fmt:UYVY2X8/640x480]"
media-ctl -V "'imx6-mipi-csi2':1 [fmt:UYVY2X8/640x480]"
media-ctl -V "'ipu1_csi0_mux':2 [fmt:UYVY2X8/640x480]"
media-ctl -V "'ipu1_csi0':1 [fmt:AYUV32/640x480]"
media-ctl -V "'ipu1_ic_prp':1 [fmt:AYUV32/640x480]"
media-ctl -V "'ipu1_ic_prpenc':1 [fmt:ARGB8888_1X32/800x600]"
# Set a format at the capture interface
v4l2-ctl -d /dev/video1 --set-fmt-video=pixelformat=RGB3
```

Streaming can then begin on “ipu1\_ic\_prpenc capture” node.

To determine what is the /dev/video node correspondent to “ipu1\_ic\_prpenc capture” :

```
media-ctl -e "ipu1_ic_prpenc capture"
/dev/video1
```

/dev/video1 is the streaming element in this case.

Starting the streaming via v4l2-ctl:

```
v4l2-ctl --stream-mmap -d /dev/video1
```

Starting the streaming via Gstreamer and sending the content to the display:

```
gst-launch-1.0 v4l2src device=/dev/video1 ! kmssink
```

### Known Issues

1. When using 90 or 270 degree rotation control at capture resolutions near the IC resizer limit of 1024x1024, and combined with planar pixel formats (YUV420, YUV422p), frame capture will often fail with no end-of-frame interrupts from the IDMAC channel. To work around this, use lower resolution and/or packed formats (YUYV, RGB3, etc.) when 90 or 270 rotations are needed.

### File list

drivers/staging/media/imx/ include/media/imx.h include/linux/imx-media.h

### References

### Authors

- Steve Longerbeam <steve\_longerbeam@mentor.com>
- Philipp Zabel <kernel@pengutronix.de>
- Russell King <linux@armlinux.org.uk>

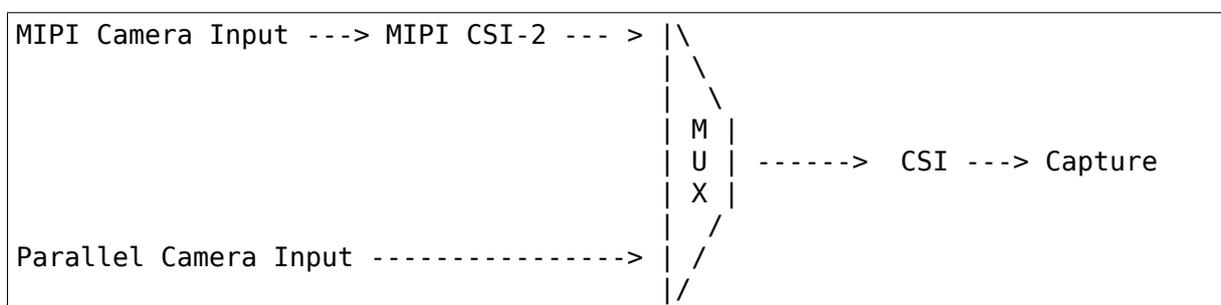
Copyright (C) 2012-2017 Mentor Graphics Inc.

## i.MX7 Video Capture Driver

### Introduction

The i.MX7 contrary to the i.MX5/6 family does not contain an Image Processing Unit (IPU); because of that the capabilities to perform operations or manipulation of the capture frames are less feature rich.

For image capture the i.MX7 has three units: - CMOS Sensor Interface (CSI) - Video Multiplexer - MIPI CSI-2 Receiver



For additional information, please refer to the latest versions of the i.MX7 reference manual<sup>1</sup>.

<sup>1</sup> <https://www.nxp.com/docs/en/reference-manual/IMX7SRM.pdf>

## Entities

### imx7-mipi-csi2

This is the MIPI CSI-2 receiver entity. It has one sink pad to receive the pixel data from MIPI CSI-2 camera sensor. It has one source pad, corresponding to the virtual channel 0. This module is compliant to previous version of Samsung D-phy, and supports two D-PHY Rx Data lanes.

### csi-mux

This is the video multiplexer. It has two sink pads to select from either camera sensor with a parallel interface or from MIPI CSI-2 virtual channel 0. It has a single source pad that routes to the CSI.

### csi

The CSI enables the chip to connect directly to external CMOS image sensor. CSI can interface directly with Parallel and MIPI CSI-2 buses. It has 256 x 64 FIFO to store received image pixel data and embedded DMA controllers to transfer data from the FIFO through AHB bus.

This entity has one sink pad that receives from the csi-mux entity and a single source pad that routes video frames directly to memory buffers. This pad is routed to a capture device node.

## Usage Notes

To aid in configuration and for backward compatibility with V4L2 applications that access controls only from video device nodes, the capture device interfaces inherit controls from the active entities in the current pipeline, so controls can be accessed either directly from the subdev or from the active capture device interface. For example, the sensor controls are available either from the sensor subdevs or from the active capture device.

### Warp7 with OV2680

On this platform an OV2680 MIPI CSI-2 module is connected to the internal MIPI CSI-2 receiver. The following example configures a video capture pipeline with an output of 800x600, and BGGR 10 bit bayer format:

```
# Setup links
media-ctl -l "'ov2680 1-0036':0 -> 'imx7-mipi-csis.0':0[1]"
media-ctl -l "'imx7-mipi-csis.0':1 -> 'csi-mux':1[1]"
media-ctl -l "'csi-mux':2 -> 'csi':0[1]"
media-ctl -l "'csi':1 -> 'csi capture':0[1]"

# Configure pads for pipeline
```

(continues on next page)

(continued from previous page)

```
media-ctl -V "'ov2680 1-0036':0 [fmt:SBGGR10_1X10/800x600 field:none]"
media-ctl -V "'csi-mux':1 [fmt:SBGGR10_1X10/800x600 field:none]"
media-ctl -V "'csi-mux':2 [fmt:SBGGR10_1X10/800x600 field:none]"
media-ctl -V "'imx7-mipi-csis.0':0 [fmt:SBGGR10_1X10/800x600 field:none]"
media-ctl -V "'csi':0 [fmt:SBGGR10_1X10/800x600 field:none]"
```

After this streaming can start. The v4l2-ctl tool can be used to select any of the resolutions supported by the sensor.

```
# media-ctl -p
Media controller API version 5.2.0

Media device information
-----
driver          imx7-csi
model           imx-media
serial
bus info
hw revision     0x0
driver version  5.2.0

Device topology
- entity 1: csi (2 pads, 2 links)
  type V4L2 subdev subtype Unknown flags 0
  device node name /dev/v4l-subdev0
  pad0: Sink
    [fmt:SBGGR10_1X10/800x600 field:none colorspace:srgb
  ↪xfer:srgb ycbcr:601 quantization:full-range]
    <- "csi-mux":2 [ENABLED]
  pad1: Source
    [fmt:SBGGR10_1X10/800x600 field:none colorspace:srgb
  ↪xfer:srgb ycbcr:601 quantization:full-range]
    -> "csi capture":0 [ENABLED]

- entity 4: csi capture (1 pad, 1 link)
  type Node subtype V4L flags 0
  device node name /dev/video0
  pad0: Sink
    <- "csi":1 [ENABLED]

- entity 10: csi-mux (3 pads, 2 links)
  type V4L2 subdev subtype Unknown flags 0
  device node name /dev/v4l-subdev1
  pad0: Sink
    [fmt:Y8_1X8/1x1 field:none]
  pad1: Sink
    [fmt:SBGGR10_1X10/800x600 field:none]
    <- "imx7-mipi-csis.0":1 [ENABLED]
  pad2: Source
    [fmt:SBGGR10_1X10/800x600 field:none]
    -> "csi":0 [ENABLED]

- entity 14: imx7-mipi-csis.0 (2 pads, 2 links)
  type V4L2 subdev subtype Unknown flags 0
  device node name /dev/v4l-subdev2
```

(continues on next page)

(continued from previous page)

```

pad0: Sink
      [fmt:SBGGR10_1X10/800x600 field:none]
      <- "ov2680 1-0036":0 [ENABLED]
pad1: Source
      [fmt:SBGGR10_1X10/800x600 field:none]
      -> "csi-mux":1 [ENABLED]
- entity 17: ov2680 1-0036 (1 pad, 1 link)
      type V4L2 subdev subtype Sensor flags 0
      device node name /dev/v4l-subdev3
      pad0: Source
            [fmt:SBGGR10_1X10/800x600@1/30 field:none colorspace:srgb]
            -> "imx7-mipi-csis.0":0 [ENABLED]

```

## References

### Intel Image Processing Unit 3 (IPU3) Imaging Unit (ImgU) driver

Copyright © 2018 Intel Corporation

## Introduction

This file documents the Intel IPU3 (3rd generation Image Processing Unit) Imaging Unit drivers located under `drivers/media/pci/intel/ipu3` (CIO2) as well as under `drivers/staging/media/ipu3` (ImgU).

The Intel IPU3 found in certain Kaby Lake (as well as certain Sky Lake) platforms (U/Y processor lines) is made up of two parts namely the Imaging Unit (ImgU) and the CIO2 device (MIPI CSI2 receiver).

The CIO2 device receives the raw Bayer data from the sensors and outputs the frames in a format that is specific to the IPU3 (for consumption by the IPU3 ImgU). The CIO2 driver is available as `drivers/media/pci/intel/ipu3/ipu3-cio2*` and is enabled through the `CONFIG_VIDEO_IPU3_CIO2` config option.

The Imaging Unit (ImgU) is responsible for processing images captured by the IPU3 CIO2 device. The ImgU driver sources can be found under `drivers/staging/media/ipu3` directory. The driver is enabled through the `CONFIG_VIDEO_IPU3_IMGU` config option.

The two driver modules are named `ipu3_csi2` and `ipu3_imgu`, respectively.

The drivers has been tested on Kaby Lake platforms (U/Y processor lines).

Both of the drivers implement V4L2, Media Controller and V4L2 sub-device interfaces. The IPU3 CIO2 driver supports camera sensors connected to the CIO2 MIPI CSI-2 interfaces through V4L2 sub-device sensor drivers.

### CIO2

The CIO2 is represented as a single V4L2 subdev, which provides a V4L2 subdev interface to the user space. There is a video node for each CSI-2 receiver, with a single media controller interface for the entire device.

The CIO2 contains four independent capture channel, each with its own MIPI CSI-2 receiver and DMA engine. Each channel is modelled as a V4L2 sub-device exposed to userspace as a V4L2 sub-device node and has two pads:

pad	direction	purpose
0	sink	MIPI CSI-2 input, connected to the sensor subdev
1	source	Raw video capture, connected to the V4L2 video interface

The V4L2 video interfaces model the DMA engines. They are exposed to userspace as V4L2 video device nodes.

### Capturing frames in raw Bayer format

CIO2 MIPI CSI2 receiver is used to capture frames (in packed raw Bayer format) from the raw sensors connected to the CSI2 ports. The captured frames are used as input to the ImgU driver.

Image processing using IPU3 ImgU requires tools such as `raw2pnm`<sup>2</sup>, and `yavta`<sup>3</sup> due to the following unique requirements and / or features specific to IPU3.

- The IPU3 CSI2 receiver outputs the captured frames from the sensor in packed raw Bayer format that is specific to IPU3.
- Multiple video nodes have to be operated simultaneously.

Let us take the example of `ov5670` sensor connected to CSI2 port 0, for a 2592x1944 image capture.

Using the media controller APIs, the `ov5670` sensor is configured to send frames in packed raw Bayer format to IPU3 CSI2 receiver.

```
# This example assumes /dev/media0 as the CIO2 media device
```

```
export MDEV=/dev/media0
```

```
# and that ov5670 sensor is connected to i2c bus 10 with address 0x36
```

```
export SDEV=$(mediactl -d $MDEV -e "ov5670 10-0036" )
```

```
# Establish the link for the media devices using mediactl4 mediactl -d $MDEV -l "ov5670:0 -> ipu3-csi2 0:0[1]"
```

---

<sup>2</sup> <https://github.com/intel/nvt>

<sup>3</sup> <http://git.ideasonboard.org/yavta.git>

<sup>4</sup> <http://git.ideasonboard.org/?p=media-ctl.git;a=summary>

```
# Set the format for the media devices media-ctl -d $MDEV -V "ov5670:0 [fmt:SGRBG10/2592x1944]"
```

```
media-ctl -d $MDEV -V "ipu3-csi2 0:0 [fmt:SGRBG10/2592x1944]"
```

```
media-ctl -d $MDEV -V "ipu3-csi2 0:1 [fmt:SGRBG10/2592x1944]"
```

Once the media pipeline is configured, desired sensor specific settings (such as exposure and gain settings) can be set, using the yavta tool.

e.g

```
yavta -w 0x009e0903 444 $SDEV
```

```
yavta -w 0x009e0913 1024 $SDEV
```

```
yavta -w 0x009e0911 2046 $SDEV
```

Once the desired sensor settings are set, frame captures can be done as below.

e.g

```
yavta -data-prefix -u -c10 -n5 -I -s2592x1944 -file=/tmp/frame-#.bin -f  
IPU3_SGRBG10 $(media-ctl -d $MDEV -e "ipu3-cio2 0" )
```

With the above command, 10 frames are captured at 2592x1944 resolution, with sGRBG10 format and output as IPU3\_SGRBG10 format.

The captured frames are available as /tmp/frame-#.bin files.

## ImgU

The ImgU is represented as two V4L2 subdevs, each of which provides a V4L2 subdev interface to the user space.

Each V4L2 subdev represents a pipe, which can support a maximum of 2 streams. This helps to support advanced camera features like Continuous View Finder (CVF) and Snapshot During Video(SDV).

The ImgU contains two independent pipes, each modelled as a V4L2 sub-device exposed to userspace as a V4L2 sub-device node.

Each pipe has two sink pads and three source pads for the following purpose:

pad	direction	purpose
0	sink	Input raw video stream
1	sink	Processing parameters
2	source	Output processed video stream
3	source	Output viewfinder video stream
4	source	3A statistics

Each pad is connected to a corresponding V4L2 video interface, exposed to userspace as a V4L2 video device node.

### Device operation

With ImgU, once the input video node ( “ipu3-imgu 0/1” :0, in <entity>:<pad-number> format) is queued with buffer (in packed raw Bayer format), ImgU starts processing the buffer and produces the video output in YUV format and statistics output on respective output nodes. The driver is expected to have buffers ready for all of parameter, output and statistics nodes, when input video node is queued with buffer.

At a minimum, all of input, main output, 3A statistics and viewfinder video nodes should be enabled for IPU3 to start image processing.

Each ImgU V4L2 subdev has the following set of video nodes.

### input, output and viewfinder video nodes

The frames (in packed raw Bayer format specific to the IPU3) received by the input video node is processed by the IPU3 Imaging Unit and are output to 2 video nodes, with each targeting a different purpose (main output and viewfinder output).

Details on and the Bayer format specific to the IPU3 can be found in v4l2-pix-fmt-ipu3-sbggr10.

The driver supports V4L2 Video Capture Interface as defined at devices.

Only the multi-planar API is supported. More details can be found at planar-apis.

### Parameters video node

The parameters video node receives the ImgU algorithm parameters that are used to configure how the ImgU algorithms process the image.

Details on processing parameters specific to the IPU3 can be found in v4l2-meta-fmt-params.

### 3A statistics video node

3A statistics video node is used by the ImgU driver to output the 3A (auto focus, auto exposure and auto white balance) statistics for the frames that are being processed by the ImgU to user space applications. User space applications can use this statistics data to compute the desired algorithm parameters for the ImgU.

## Configuring the Intel IPU3

The IPU3 ImgU pipelines can be configured using the Media Controller, defined at `media_controller`.

### Running mode and firmware binary selection

ImgU works based on firmware, currently the ImgU firmware support run 2 pipes in time-sharing with single input frame data. Each pipe can run at certain mode - “VIDEO” or “STILL”, “VIDEO” mode is commonly used for video frames capture, and “STILL” is used for still frame capture. However, you can also select “VIDEO” to capture still frames if you want to capture images with less system load and power. For “STILL” mode, ImgU will try to use smaller BDS factor and output larger bayer frame for further YUV processing than “VIDEO” mode to get high quality images. Besides, “STILL” mode need XNR3 to do noise reduction, hence “STILL” mode will need more power and memory bandwidth than “VIDEO” mode. TNR will be enabled in “VIDEO” mode and bypassed by “STILL” mode. ImgU is running at “VIDEO” mode by default, the user can use v4l2 control `V4L2_CID_INTEL_IPU3_MODE` (currently defined in `drivers/staging/media/ipu3/include/intel-ipu3.h`) to query and set the running mode. For user, there is no difference for buffer queueing between the “VIDEO” and “STILL” mode, mandatory input and main output node should be enabled and buffers need be queued, the statistics and the view-finder queues are optional.

The firmware binary will be selected according to current running mode, such log “using binary `if_to_osys_stripped`” or “using binary `if_to_osys_primary_stripped`” could be observed if you enable the ImgU dynamic debug, the binary `if_to_osys_stripped` is selected for “VIDEO” and the binary “`if_to_osys_primary_stripped`” is selected for “STILL” .

### Processing the image in raw Bayer format

#### Configuring ImgU V4L2 subdev for image processing

The ImgU V4L2 subdevs have to be configured with media controller APIs to have all the video nodes setup correctly.

Let us take “`ipu3-imgu 0`” subdev as an example.

```
media-ctl -d $MDEV -r
```

```
media-ctl -d $MDEV -l “ipu3-imgu 0 input” :0 -> “ipu3-imgu 0” :0[1]
```

```
media-ctl -d $MDEV -l “ipu3-imgu 0” :2 -> “ipu3-imgu 0 output” :0[1]
```

```
media-ctl -d $MDEV -l “ipu3-imgu 0” :3 -> “ipu3-imgu 0 viewfinder” :0[1]
```

```
media-ctl -d $MDEV -l “ipu3-imgu 0” :4 -> “ipu3-imgu 0 3a stat” :0[1]
```

Also the pipe mode of the corresponding V4L2 subdev should be set as desired (e.g 0 for video mode or 1 for still mode) through the control id `0x009819a1` as below.

```
yavta -w “0x009819A1 1” /dev/v4l-subdev7
```

Certain hardware blocks in ImgU pipeline can change the frame resolution by cropping or scaling, these hardware blocks include Input Feeder(IF), Bayer Down Scaler (BDS) and Geometric Distortion Correction (GDC). There is also a block which can change the frame resolution - YUV Scaler, it is only applicable to the secondary output.

RAW Bayer frames go through these ImgU pipeline hardware blocks and the final processed image output to the DDR memory.

### **Input Feeder**

Input Feeder gets the Bayer frame data from the sensor, it can enable cropping of lines and columns from the frame and then store pixels into device' s internal pixel buffer which are ready to readout by following blocks.

### **Bayer Down Scaler**

Bayer Down Scaler is capable of performing image scaling in Bayer domain, the downscale factor can be configured from 1X to 1/4X in each axis with configuration steps of 0.03125 (1/32).

### **Geometric Distortion Correction**

Geometric Distortion Correction is used to performe correction of distortions and image filtering. It needs some extra filter and envelop padding pixels to work, so the input resolution of GDC should be larger than the output resolution.

### **YUV Scaler**

YUV Scaler which similar with BDS, but it is mainly do image down scaling in YUV domain, it can support up to 1/12X down scaling, but it can not be applied to the main output.

The ImgU V4L2 subdev has to be configured with the supported resolutions in all the above hardware blocks, for a given input resolution. For a given supported resolution for an input frame, the Input Feeder, Bayer Down Scaler and GDC blocks should be configured with the supported resolutions as each hardware block has its own alignment requirement.

You must configure the output resolution of the hardware blocks smartly to meet the hardware requirement along with keeping the maximum field of view. The intermediate resolutions can be generated by specific tool -

<https://github.com/intel/intel-ipu3-pipecfg>

This tool can be used to generate intermediate resolutions. More information can be obtained by looking at the following IPU3 ImgU configuration table.

[https://chromium.googlesource.com/chromiumos/overlays/board-overlays/+/\\_master](https://chromium.googlesource.com/chromiumos/overlays/board-overlays/+/_master)

Under `baseboard-poppy/media-libs/cros-camera-hal-configs-poppy/files/gcss` directory, `graph_settings_ov5670.xml` can be used as an example.

The following steps prepare the ImgU pipeline for the image processing.

1. The ImgU V4L2 subdev data format should be set by using the VID-IOC\_SUBDEV\_S\_FMT on pad 0, using the GDC width and height obtained above.

```

<?xml version="1.0" encoding="UTF-8"?>
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/
  ↪1999/xlink" width="774pt" height="152pt" viewBox="0 0 774 152" version=
  ↪"1.1">
<defs>
<g>
<symbol overflow="visible" id="glyph0-0">
<path style="stroke:none;" d="M 1 0 L 1 -15 L 9 -15 L 9 0 Z M 8 -1 L 8 -14
  ↪L 2 -14 L 2 -1 Z M 8 -1 "/>
</symbol>
<symbol overflow="visible" id="glyph0-1">
<path style="stroke:none;" d="M 4.6875 -1.15625 C 5.519531 -1.15625 6.
  ↪15625 -1.316406 6.59375 -1.640625 C 7.039062 -1.960938 7.265625 -2.
  ↪441406 7.265625 -3.078125 C 7.265625 -3.460938 7.179688 -3.789062 7.
  ↪015625 -4.0625 C 6.859375 -4.34375 6.644531 -4.582031 6.375 -4.78125 C 6.
  ↪113281 -4.988281 5.816406 -5.171875 5.484375 -5.328125 C 5.148438 -5.
  ↪484375 4.804688 -5.628906 4.453125 -5.765625 C 4.054688 -5.921875 3.
  ↪675781 -6.097656 3.3125 -6.296875 C 2.945312 -6.492188 2.617188 -6.
  ↪726562 2.328125 -7 C 2.046875 -7.269531 1.820312 -7.582031 1.65625 -7.
  ↪9375 C 1.488281 -8.300781 1.40625 -8.726562 1.40625 -9.21875 C 1.40625 -
  ↪10.300781 1.742188 -11.144531 2.421875 -11.75 C 3.097656 -12.351562 4.
  ↪046875 -12.65625 5.265625 -12.65625 C 5.597656 -12.65625 5.925781 -12.
  ↪628906 6.25 -12.578125 C 6.570312 -12.535156 6.875 -12.476562 7.15625 -
  ↪12.40625 C 7.4375 -12.34375 7.6875 -12.265625 7.90625 -12.171875 C 8.125
  ↪-12.085938 8.300781 -12 8.4375 -11.90625 L 7.921875 -10.515625 C 7.
  ↪648438 -10.679688 7.28125 -10.84375 6.8125 -11 C 6.351562 -11.15625 5.
  ↪835938 -11.234375 5.265625 -11.234375 C 4.660156 -11.234375 4.140625 -11.
  ↪082031 3.703125 -10.78125 C 3.265625 -10.488281 3.046875 -10.039062 3.
  ↪046875 -9.4375 C 3.046875 -9.09375 3.109375 -8.800781 3.234375 -8.5625 C
  ↪3.359375 -8.320312 3.53125 -8.109375 3.75 -7.921875 C 3.96875 -7.742188
  ↪4.222656 -7.582031 4.515625 -7.4375 C 4.804688 -7.289062 5.128906 -7.
  ↪144531 5.484375 -7 C 5.984375 -6.789062 6.441406 -6.578125 6.859375 -6.
  ↪359375 C 7.285156 -6.148438 7.648438 -5.894531 7.953125 -5.59375 C 8.
  ↪253906 -5.300781 8.488281 -4.953125 8.65625 -4.546875 C 8.820312 -4.
  ↪148438 8.90625 -3.664062 8.90625 -3.09375 C 8.90625 -2.019531 8.539062 -
  ↪1.191406 7.8125 -0.609375 C 7.082031 -0.0234375 6.039062 0.265625 4.6875
  ↪0.265625 C 4.238281 0.265625 3.820312 0.234375 3.4375 0.171875 C 3.
  ↪050781 0.109375 2.707031 0.03125 2.40625 -0.0625 C 2.101562 -0.15625 1.
  ↪835938 -0.25 1.609375 -0.34375 C 1.390625 -0.4375 1.21875 -0.519531 1.
  ↪09375 -0.59375 L 1.59375 -1.953125 C 1.863281 -1.804688 2.257812 -1.
  ↪632812 2.78125 -1.4375 C 3.300781 -1.25 3.9375 -1.15625 4.6875 -1.15625
  ↪Z M 4.6875 -1.15625 "/>
</symbol>
<symbol overflow="visible" id="glyph0-2">
<path style="stroke:none;" d="M 5.1875 -9.5 C 6.4375 -9.5 7.398438 -9.
  ↪109375 8.078125 -8.328125 C 8.753906 -7.546875 9.09375 -6.363281 9.09375
  ↪-4.78125 L 9.09375 -4.203125 L 2.453125 -4.203125 C 2.523438 -3.242188 2.
  ↪84375 -2.515625 3.40625 -2.015625 C 3.976562 -1.515625 4.773438 -1.
  ↪265625 5.796875 -1.265625 C 6.390625 -1.265625 6.890625 -1.3125 7.296875
  ↪-1.40625 C 7.710938 -1.5 8.023438 -1.597656 8.234375 -1.703125 L 8.
  ↪453125 -0.296875 C 8.253906 -0.191406 7.894531 -0.0820312 7.375 0.03125
  ↪C 6.851562 0.15625 6.269531 0.21875 5.625 0.21875 C 4.820312 0.21875 4.
  ↪113281 0.0976562 3.5 -0.140625 C 2.894531 -0.390625 2.394531 -0.726562 2
  ↪-1.15625 C 1.601562 -1.582031 1.300781 -2.09375 1.09375 -2.6875 C 0.
  ↪894531 -3.28125 0.796875 -3.925781 0.796875 -4.625 C 0.796875 -5.445312
  ↪0.921875 -6.164062 1.171875 -6.78125 C 1.429688 -7.394531 1.765625 -7.
  ↪898438 2.171875 -8.296875 C 2.585938 -8.703125 3.054688 -9.003906 3.
  ↪578125 -9.203125 C 4.097656 -9.398438 4.632812 -9.5 5.1875 -9.5 Z M 7.
  ↪421875 -5.546875 C 7.421875 -6.328125 7.210938 -6.945312 6.796875 -7.
  ↪40625 C 6.390625 -7.863281 5.84375 -8.09375 5.15625 -8.09375 C 4.769531 -
  ↪8.09375 4.421875 -8.019531 4.109375 -7.875 C 3.796875 -7.726562 3.523438
  ↪-7.535156 3.296875 -7.296875 C 3.066406 -7.054688 2.882812 -6.78125 2.75
  ↪-6.46875 C 2.625 -6.164062 2.539062 -5.859375 2.5 -5.546875 Z M 7.421875
  ↪-5.546875 "/>

```

2. The ImgU V4L2 subdev cropping should be set by using the VIDIOC\_SUBDEV\_S\_SELECTION on pad 0, with V4L2\_SEL\_TGT\_CROP as the target, using the input feeder height and width.

3. The ImgU V4L2 subdev composing should be set by using the VIDIOC\_SUBDEV\_S\_SELECTION on pad 0, with V4L2\_SEL\_TGT\_COMPOSE as the target, using the BDS height and width.

For the ov5670 example, for an input frame with a resolution of 2592x1944 (which is input to the ImgU subdev pad 0), the corresponding resolutions for input feeder, BDS and GDC are 2592x1944, 2592x1944 and 2560x1920 respectively.

Once this is done, the received raw Bayer frames can be input to the ImgU V4L2 subdev as below, using the open source application v4l2n<sup>2</sup>.

For an image captured with 2592x1944<sup>5</sup> resolution, with desired output resolution as 2560x1920 and viewfinder resolution as 2560x1920, the following v4l2n command can be used. This helps process the raw Bayer frames and produces the desired results for the main output image and the viewfinder output, in NV12 format.

```
v4l2n -pipe=4 -load=/tmp/frame-#.bin -open=/dev/video4 -
fmt=type:VIDEO_OUTPUT_MPLANE,width=2592,height=1944,pixelformat=0X47337069
-reqbufs=type:VIDEO_OUTPUT_MPLANE,count:1 -pipe=1 -
output=/tmp/frames.out -open=/dev/video5 -fmt=type:VIDEO_CAPTURE_MPLANE,width=2560
-reqbufs=type:VIDEO_CAPTURE_MPLANE,count:1 -pipe=2 -
output=/tmp/frames.vf -open=/dev/video6 -fmt=type:VIDEO_CAPTURE_MPLANE,width=2560
-reqbufs=type:VIDEO_CAPTURE_MPLANE,count:1 -pipe=3 -
open=/dev/video7 -output=/tmp/frames.3A -fmt=type:META_CAPTURE,? -
reqbufs=count:1,type:META_CAPTURE -pipe=1,2,3,4 -stream=5
```

You can also use yavta<sup>3</sup> command to do same thing as above:

```
yavta --data-prefix -Bcapture-mplane -c10 -n5 -I -s2592x1944 \
--file=frame-#.out -f NV12 /dev/video5 & \
yavta --data-prefix -Bcapture-mplane -c10 -n5 -I -s2592x1944 \
--file=frame-#.vf -f NV12 /dev/video6 & \
yavta --data-prefix -Bmeta-capture -c10 -n5 -I \
--file=frame-#.3a /dev/video7 & \
yavta --data-prefix -Boutput-mplane -c10 -n5 -I -s2592x1944 \
--file=/tmp/frame-in.cio2 -f IPU3_SGRBG10 /dev/video4
```

where /dev/video4, /dev/video5, /dev/video6 and /dev/video7 devices point to input, output, viewfinder and 3A statistics video nodes respectively.

<sup>5</sup> ImgU limitation requires an additional 16x16 for all input resolutions

## Converting the raw Bayer image into YUV domain

The processed images after the above step, can be converted to YUV domain as below.

### Main output frames

```
raw2pnm -x2560 -y1920 -fNV12 /tmp/frames.out /tmp/frames.out.ppm
```

where 2560x1920 is output resolution, NV12 is the video format, followed by input frame and output PNM file.

### Viewfinder output frames

```
raw2pnm -x2560 -y1920 -fNV12 /tmp/frames.vf /tmp/frames.vf.ppm
```

where 2560x1920 is output resolution, NV12 is the video format, followed by input frame and output PNM file.

### Example user space code for IPU3

User space code that configures and uses IPU3 is available here.

[https://chromium.googlesource.com/chromiumos/platform/arc-camera/+/\\_/master/](https://chromium.googlesource.com/chromiumos/platform/arc-camera/+/_/master/)

The source can be located under hal/intel directory.

### Overview of IPU3 pipeline

IPU3 pipeline has a number of image processing stages, each of which takes a set of parameters as input. The major stages of pipelines are shown here:

The table below presents a description of the above algorithms.

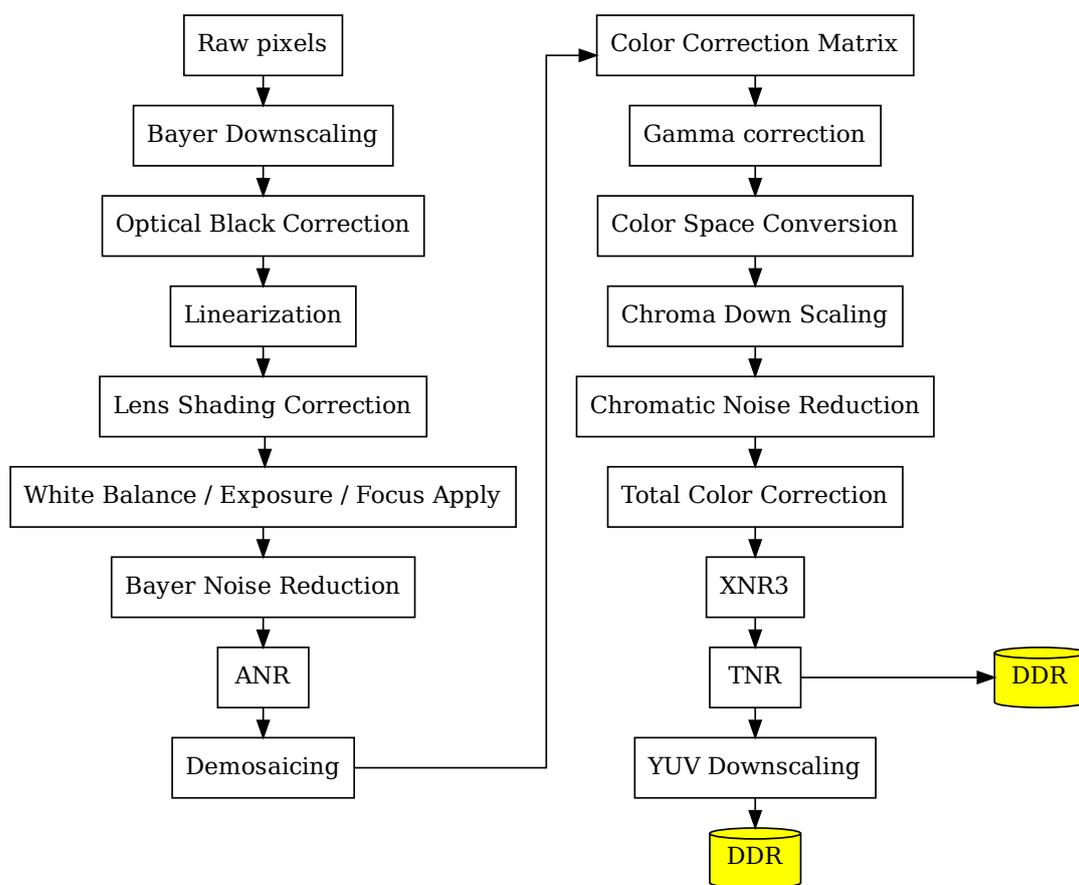


Fig. 4: IPU3 ImgU Pipeline Diagram

Name	Description
Optical Black Correction	Optical Black Correction block subtracts a pre-defined value from the respective pixel values to obtain better image quality. Defined in <code>ipu3_uapi_obgrid_param</code> .
Linearization	This algo block uses linearization parameters to address non-linearity sensor effects. The Lookup table table is defined in <code>ipu3_uapi_isp_lin_vmem_params</code> .
SHD	Lens shading correction is used to correct spatial non-uniformity of the pixel response due to optical lens shading. This is done by applying a different gain for each pixel. The gain, black level etc are configured in <code>ipu3_uapi_shd_config_static</code> .
BNR	Bayer noise reduction block removes image noise by applying a bilateral filter. See <code>ipu3_uapi_bnr_static_config</code> for details.
ANR	Advanced Noise Reduction is a block based algorithm that performs noise reduction in the Bayer domain. The convolution matrix etc can be found in <code>ipu3_uapi_anr_config</code> .
DM	Demosaicing converts raw sensor data in Bayer format into RGB (Red, Green, Blue) presentation. Then add outputs of estimation of Y channel for following stream processing by Firmware. The struct is defined as <code>ipu3_uapi_dm_config</code> .
Color Correction	Color Correction algo transforms sensor specific color space to the standard "sRGB" color space. This is done by applying 3x3 matrix defined in <code>ipu3_uapi_ccm_mat_config</code> .
Gamma correction	Gamma correction <code>ipu3_uapi_gamma_config</code> is a basic non-linear tone mapping correction that is applied per pixel for each pixel component.
CSC	Color space conversion transforms each pixel from the RGB primary presentation to YUV (Y: brightness, UV: Luminance) presentation. This is done by applying a 3x3 matrix defined in <code>ipu3_uapi_csc_mat_config</code>
CDS	Chroma down sampling After the CSC is performed, the Chroma Down Sampling is applied for a UV plane down sampling by a factor of 2 in each direction for YUV 4:2:0 using a 4x2 configurable filter <code>ipu3_uapi_cds_params</code> .
CHNR	Chroma noise reduction This block processes only the chrominance pixels and performs noise reduction by cleaning the high frequency noise. See struct <code>ipu3_uapi_yuvp1_chnr_config</code> .
TCC	Total color correction as defined in struct <code>ipu3_uapi_yuvp2_tcc_static_config</code> .
XNR3	eXtreme Noise Reduction V3 is the third revision of noise reduction algorithm used to improve image quality. This removes the low frequency noise in the captured image. Two related structs are being defined, <code>ipu3_uapi_isp_xnr3_params</code> for ISP data memory and <code>ipu3_uapi_isp_xnr3_vmem_params</code> for vector memory.
TNR	Temporal Noise Reduction block compares successive frames in time to remove anomalies / noise in pixel values. <code>ipu3_uapi_isp_tnr3_vmem_params</code> and <code>ipu3_uapi_isp_tnr3_params</code> are defined for ISP vector and data memory respectively.

Other often encountered acronyms not listed in above table:

- ACC** Accelerator cluster
- AWB\_FR** Auto white balance filter response statistics
- BDS** Bayer downscaler parameters
- CCM** Color correction matrix coefficients
- IEFd** Image enhancement filter directed
- Obgrid** Optical black level compensation
- OSYS** Output system configuration
- ROI** Region of interest
- YDS** Y down sampling
- YTM** Y-tone mapping

A few stages of the pipeline will be executed by firmware running on the ISP processor, while many others will use a set of fixed hardware blocks also called accelerator cluster (ACC) to crunch pixel data and produce statistics.

ACC parameters of individual algorithms, as defined by `ipu3_uapi_acc_param`, can be chosen to be applied by the user space through struct `ipu3_uapi_flags` embedded in `ipu3_uapi_params` structure. For parameters that are configured as not enabled by the user space, the corresponding structs are ignored by the driver, in which case the existing configuration of the algorithm will be preserved.

## References

### The ivtv driver

Author: Hans Verkuil <[hverkuil@xs4all.nl](mailto:hverkuil@xs4all.nl)>

This is a v4l2 device driver for the Conexant cx23415/6 MPEG encoder/decoder. The cx23415 can do both encoding and decoding, the cx23416 can only do MPEG encoding. Currently the only card featuring full decoding support is the Hauppauge PVR-350.

---

### Note:

- 1) This driver requires the latest encoder firmware (version 2.06.039, size 376836 bytes). Get the firmware from here:  
<https://linuxtv.org/downloads/firmware/#conexant>
  - 2) ‘normal’ TV applications do not work with this driver, you need an application that can handle MPEG input such as mplayer, xine, MythTV, etc.
- 

The primary goal of the IVTV project is to provide a “clean room” Linux Open Source driver implementation for video capture cards based on the iCompression iTVC15 or Conexant CX23415/CX23416 MPEG Codec.

## Features

- Hardware mpeg2 capture of broadcast video (and sound) via the tuner or S-Video/Composite and audio line-in.
- Hardware mpeg2 capture of FM radio where hardware support exists
- Supports NTSC, PAL, SECAM with stereo sound
- Supports SAP and bilingual transmissions.
- Supports raw VBI (closed captions and teletext).
- Supports sliced VBI (closed captions and teletext) and is able to insert this into the captured MPEG stream.
- Supports raw YUV and PCM input.

## Additional features for the PVR-350 (CX23415 based)

- Provides hardware mpeg2 playback
- Provides comprehensive OSD (On Screen Display: ie. graphics overlaying the video signal)
- Provides a framebuffer (allowing X applications to appear on the video device)
- Supports raw YUV output.

**IMPORTANT: In case of problems first read this page:** [https://help.ubuntu.com/community/Install\\_IVTV\\_Troubleshooting](https://help.ubuntu.com/community/Install_IVTV_Troubleshooting)

## See also

<https://linuxtv.org>

## IRC

<irc://irc.freenode.net/#v4l>

---

## Devices

A maximum of 12 ivtv boards are allowed at the moment.

Cards that don't have a video output capability (i.e. non PVR350 cards) lack the vbi8, vbi16, video16 and video48 devices. They also do not support the framebuffer device /dev/fbx for OSD.

The radio0 device may or may not be present, depending on whether the card has a radio tuner or not.

Here is a list of the base v4l devices:

```
crw-rw----  1 root    video    81,   0 Jun 19 22:22 /dev/video0
crw-rw----  1 root    video    81,  16 Jun 19 22:22 /dev/video16
crw-rw----  1 root    video    81,  24 Jun 19 22:22 /dev/video24
crw-rw----  1 root    video    81,  32 Jun 19 22:22 /dev/video32
crw-rw----  1 root    video    81,  48 Jun 19 22:22 /dev/video48
crw-rw----  1 root    video    81,  64 Jun 19 22:22 /dev/radio0
crw-rw----  1 root    video    81, 224 Jun 19 22:22 /dev/vbi0
crw-rw----  1 root    video    81, 228 Jun 19 22:22 /dev/vbi8
crw-rw----  1 root    video    81, 232 Jun 19 22:22 /dev/vbi16
```

### Base devices

For every extra card you have the numbers increased by one. For example, /dev/video0 is listed as the 'base' encoding capture device so we have:

- /dev/video0 is the encoding capture device for the first card (card 0)
- /dev/video1 is the encoding capture device for the second card (card 1)
- /dev/video2 is the encoding capture device for the third card (card 2)

Note that if the first card doesn't have a feature (eg no decoder, so no video16, the second card will still use video17. The simple rule is 'add the card number to the base device number'. If you have other capture cards (e.g. WinTV PCI) that are detected first, then you have to tell the ivtv module about it so that it will start counting at 1 (or 2, or whatever). Otherwise the device numbers can get confusing. The ivtv 'ivtv\_first\_minor' module option can be used for that.

- /dev/video0

The encoding capture device(s).

Read-only.

Reading from this device gets you the MPEG1/2 program stream. Example:

```
cat /dev/video0 > my.mpg (you need to hit ctrl-c to exit)
```

- /dev/video16

The decoder output device(s)

Write-only. Only present if the MPEG decoder (i.e. CX23415) exists.

An mpeg2 stream sent to this device will appear on the selected video display, audio will appear on the line-out/audio out. It is only available for cards that support video out. Example:

```
cat my.mpg >/dev/video16
```

- /dev/video24

The raw audio capture device(s).

Read-only

The raw audio PCM stereo stream from the currently selected tuner or audio line-in. Reading from this device results in a raw (signed 16 bit Little Endian, 48000 Hz, stereo pcm) capture. This device only captures audio. This should be replaced by an ALSA device in the future. Note that there is no corresponding raw audio output device, this is not supported in the decoder firmware.

- `/dev/video32`

The raw video capture device(s)

Read-only

The raw YUV video output from the current video input. The YUV format is non-standard (V4L2\_PIX\_FMT\_HM12).

Note that the YUV and PCM streams are not synchronized, so they are of limited use.

- `/dev/video48`

The raw video display device(s)

Write-only. Only present if the MPEG decoder (i.e. CX23415) exists.

Writes a YUV stream to the decoder of the card.

- `/dev/radio0`

The radio tuner device(s)

Cannot be read or written.

Used to enable the radio tuner and tune to a frequency. You cannot read or write audio streams with this device. Once you use this device to tune the radio, use `/dev/video24` to read the raw pcm stream or `/dev/video0` to get an mpeg2 stream with black video.

- `/dev/vbi0`

The ‘vertical blank interval’ (Teletext, CC, WSS etc) capture device(s)

Read-only

Captures the raw (or sliced) video data sent during the Vertical Blank Interval. This data is used to encode teletext, closed captions, VPS, widescreen signalling, electronic program guide information, and other services.

- `/dev/vbi8`

Processed vbi feedback device(s)

Read-only. Only present if the MPEG decoder (i.e. CX23415) exists.

The sliced VBI data embedded in an MPEG stream is reproduced on this device. So while playing back a recording on `/dev/video16`, you can read the embedded VBI data from `/dev/vbi8`.

- `/dev/vbi16`

The vbi ‘display’ device(s)

Write-only. Only present if the MPEG decoder (i.e. CX23415) exists.

Can be used to send sliced VBI data to the video-out connector.

### Vaio Picturebook Motion Eye Camera Driver

Copyright © 2001-2004 Stelian Pop <[stelian@popies.net](mailto:stelian@popies.net)>

Copyright © 2001-2002 Alcôve <[www.alcove.com](http://www.alcove.com)>

Copyright © 2000 Andrew Tridgell <[tridge@samba.org](mailto:tridge@samba.org)>

This driver enable the use of video4linux compatible applications with the Motion Eye camera. This driver requires the “Sony Laptop Extras” driver (which can be found in the “Misc devices” section of the kernel configuration utility) to be compiled and installed (using its “camera=1” parameter).

It can do at maximum 30 fps @ 320x240 or 15 fps @ 640x480.

Grabbing is supported in packed YUV colorspace only.

MJPEG hardware grabbing is supported via a private API (see below).

### Hardware supported

This driver supports the ‘second’ version of the MotionEye camera :)

The first version was connected directly on the video bus of the Neomagic video card and is unsupported.

The second one, made by Kawasaki Steel is fully supported by this driver (PCI vendor/device is 0x136b/0xff01)

The third one, present in recent (more or less last year) Picturebooks (C1M\* models), is not supported. The manufacturer has given the specs to the developers under a NDA (which allows the development of a GPL driver however), but things are not moving very fast (see <http://r-engine.sourceforge.net/>) (PCI vendor/device is 0x10cf/0x2011).

There is a fourth model connected on the USB bus in TR1\* Vaio laptops. This camera is not supported at all by the current driver, in fact little information if any is available for this camera (USB vendor/device is 0x054c/0x0107).

### Driver options

Several options can be passed to the meye driver using the standard module argument syntax (<param>=<value> when passing the option to the module or meye.<param>=<value> on the kernel boot line when meye is statically linked into the kernel). Those options are:

gbuffers:	number of capture buffers, default is 2 (32 max)
gbufsize:	size of each capture buffer, default is 614400
video_nr:	video device to register (0 = /dev/video0, etc)

## Module use

In order to automatically load the meye module on use, you can put those lines in your `/etc/modprobe.d/meye.conf` file:

```
alias char-major-81 videodev
alias char-major-81-0 meye
options meye gbuffers=32
```

## Usage:

```
xawtv >= 3.49 (<http://bytesex.org/xawtv/>)
    for display and uncompressed video capture:

        xawtv -c /dev/video0 -geometry 640x480
            or
        xawtv -c /dev/video0 -geometry 320x240

motioneye (<http://popies.net/meye/>)
    for getting ppm or jpg snapshots, mjpeg video
```

## Bugs / Todo

- ‘motioneye’ still uses the meye private v4l1 API extensions.

## OMAP 3 Image Signal Processor (ISP) driver

Copyright © 2010 Nokia Corporation

Copyright © 2009 Texas Instruments, Inc.

Contacts: Laurent Pinchart <[laurent.pinchart@ideasonboard.com](mailto:laurent.pinchart@ideasonboard.com)>, Sakari Ailus <[sakari.ailus@iki.fi](mailto:sakari.ailus@iki.fi)>, David Cohen <[dacohen@gmail.com](mailto:dacohen@gmail.com)>

## Introduction

This file documents the Texas Instruments OMAP 3 Image Signal Processor (ISP) driver located under `drivers/media/platform/omap3isp`. The original driver was written by Texas Instruments but since that it has been rewritten (twice) at Nokia.

The driver has been successfully used on the following versions of OMAP 3:

- 3430
- 3530
- 3630

The driver implements V4L2, Media controller and v4l2\_subdev interfaces. Sensor, lens and flash drivers using the v4l2\_subdev interface in the kernel are supported.

### Split to subdevs

The OMAP 3 ISP is split into V4L2 subdevs, each of the blocks inside the ISP having one subdev to represent it. Each of the subdevs provide a V4L2 subdev interface to userspace.

- OMAP3 ISP CCP2
- OMAP3 ISP CSI2a
- OMAP3 ISP CCDC
- OMAP3 ISP preview
- OMAP3 ISP resizer
- OMAP3 ISP AEWB
- OMAP3 ISP AF
- OMAP3 ISP histogram

Each possible link in the ISP is modelled by a link in the Media controller interface. For an example program see<sup>1</sup>.

### Controlling the OMAP 3 ISP

In general, the settings given to the OMAP 3 ISP take effect at the beginning of the following frame. This is done when the module becomes idle during the vertical blanking period on the sensor. In memory-to-memory operation the pipe is run one frame at a time. Applying the settings is done between the frames.

All the blocks in the ISP, excluding the CSI-2 and possibly the CCP2 receiver, insist on receiving complete frames. Sensors must thus never send the ISP partial frames.

Autoidle does have issues with some ISP blocks on the 3430, at least. Autoidle is only enabled on 3630 when the omap3isp module parameter autoidle is non-zero.

### Technical reference manuals (TRMs) and other documentation

OMAP 3430 TRM: <URL:[http://focus.ti.com/pdfs/wtbu/OMAP34xx\\_ES3.1.x\\_PUBLIC\\_TRM\\_vZM](http://focus.ti.com/pdfs/wtbu/OMAP34xx_ES3.1.x_PUBLIC_TRM_vZM)> Referenced 2011-03-05.

OMAP 35xx TRM: <URL:<http://www.ti.com/litv/pdf/spruf98o>> Referenced 2011-03-05.

OMAP 3630 TRM: <URL:[http://focus.ti.com/pdfs/wtbu/OMAP36xx\\_ES1.x\\_PUBLIC\\_TRM\\_vQ.zip](http://focus.ti.com/pdfs/wtbu/OMAP36xx_ES1.x_PUBLIC_TRM_vQ.zip)> Referenced 2011-03-05.

DM 3730 TRM: <URL:<http://www.ti.com/litv/pdf/sprugn4h>> Referenced 2011-03-06.

---

<sup>1</sup> <http://git.ideasonboard.org/?p=media-ctl.git;a=summary>

## References

### OMAP4 ISS Driver

Author: Sergio Aguirre <[sergio.a.aguirre@gmail.com](mailto:sergio.a.aguirre@gmail.com)>

Copyright (C) 2012, Texas Instruments

## Introduction

The OMAP44XX family of chips contains the Imaging SubSystem (a.k.a. ISS), Which contains several components that can be categorized in 3 big groups:

- Interfaces (2 Interfaces: CSI2-A & CSI2-B/CCP2)
- ISP (Image Signal Processor)
- SIMCOP (Still Image Coprocessor)

For more information, please look in<sup>1</sup> for latest version of: “OMAP4430 Multimedia Device Silicon Revision 2.x”

As of Revision AB, the ISS is described in detail in section 8.

This driver is supporting **only** the CSI2-A/B interfaces for now.

It makes use of the Media Controller framework<sup>2</sup>, and inherited most of the code from OMAP3 ISP driver (found under `drivers/media/platform/omap3isp/*`), except that it doesn't need an IOMMU now for ISS buffers memory mapping.

Supports usage of MMAP buffers only (for now).

## Tested platforms

- OMAP4430SDP, w/ ES2.1 GP & SEVM4430-CAM-V1-0 (Contains IMX060 & OV5640, in which only the last one is supported, outputting YUV422 frames).
- TI Blaze MDP, w/ OMAP4430 ES2.2 EMU (Contains 1 IMX060 & 2 OV5650 sensors, in which only the OV5650 are supported, outputting RAW10 frames).
- PandaBoard, Rev. A2, w/ OMAP4430 ES2.1 GP & OV adapter board, tested with following sensors: \* OV5640 \* OV5650
- Tested on mainline kernel:

<http://git.kernel.org/?p=linux/kernel/git/torvalds/linux.git;a=summary>

Tag: v3.3 (commit c16fa4f2ad19908a47c63d8fa436a1178438c7e7)

---

<sup>1</sup> <http://focus.ti.com/general/docs/wtbu/wtbudocumentcenter.tsp?navigationId=12037&templateId=6123#62>

<sup>2</sup> <http://lwn.net/Articles/420485/>

### File list

drivers/staging/media/omap4iss/ include/linux/platform\_data/media/omap4iss.h

### References

#### Philips webcams (pwc driver)

This file contains some additional information for the Philips and OEM webcams. E-mail: [webcam@smcc.demon.nl](mailto:webcam@smcc.demon.nl) Last updated: 2004-01-19 Site: <http://www.smcc.demon.nl/webcam/>

As of this moment, the following cameras are supported:

- Philips PCA645
- Philips PCA646
- Philips PCVC675
- Philips PCVC680
- Philips PCVC690
- Philips PCVC720/40
- Philips PCVC730
- Philips PCVC740
- Philips PCVC750
- Askey VC010
- Creative Labs Webcam 5
- Creative Labs Webcam Pro Ex
- Logitech QuickCam 3000 Pro
- Logitech QuickCam 4000 Pro
- Logitech QuickCam Notebook Pro
- Logitech QuickCam Zoom
- Logitech QuickCam Orbit
- Logitech QuickCam Sphere
- Samsung MPC-C10
- Samsung MPC-C30
- Sotec Afina Eye
- AME CU-001
- Visionite VCS-UM100
- Visionite VCS-UC300

The main webpage for the Philips driver is at the address above. It contains a lot of extra information, a FAQ, and the binary plugin 'PWCX'. This plugin contains decompression routines that allow you to use higher image sizes and framerates; in addition the webcam uses less bandwidth on the USB bus (handy if you want to run more than 1 camera simultaneously). These routines fall under a NDA, and may therefore not be distributed as source; however, its use is completely optional.

You can build this code either into your kernel, or as a module. I recommend the latter, since it makes troubleshooting a lot easier. The built-in microphone is supported through the USB Audio class.

When you load the module you can set some default settings for the camera; some programs depend on a particular image-size or -format and don't know how to set it properly in the driver. The options are:

**size** Can be one of 'sqcif', 'qsif', 'qcif', 'sif', 'cif' or 'vga', for an image size of resp. 128x96, 160x120, 176x144, 320x240, 352x288 and 640x480 (of course, only for those cameras that support these resolutions).

**fps** Specifies the desired framerate. Is an integer in the range of 4-30.

**fbufs** This parameter specifies the number of internal buffers to use for storing frames from the cam. This will help if the process that reads images from the cam is a bit slow or momentarily busy. However, on slow machines it only introduces lag, so choose carefully. The default is 3, which is reasonable. You can set it between 2 and 5.

**mbufs** This is an integer between 1 and 10. It will tell the module the number of buffers to reserve for mmap(), VIDIOCCGMBUF, VIDIOCMCAPTURE and friends. The default is 2, which is adequate for most applications (double buffering).

Should you experience a lot of 'Dumping frame...' messages during grabbing with a tool that uses mmap(), you might want to increase it. However, it doesn't really buffer images, it just gives you a bit more slack when your program is behind. But you need a multi-threaded or forked program to really take advantage of these buffers.

The absolute maximum is 10, but don't set it too high! Every buffer takes up 460 KB of RAM, so unless you have a lot of memory setting this to something more than 4 is an absolute waste. This memory is only allocated during open(), so nothing is wasted when the camera is not in use.

**power\_save** When power\_save is enabled (set to 1), the module will try to shut down the cam on close() and re-activate on open(). This will save power and turn off the LED. Not all cameras support this though (the 645 and 646 don't have power saving at all), and some models don't work either (they will shut down, but never wake up). Consider this experimental. By default this option is disabled.

**compression (only useful with the plugin)** With this option you can control the compression factor that the camera uses to squeeze the image through the USB bus. You can set the parameter between 0 and 3:

0 = prefer uncompressed images; if the requested mode is not available in an uncompressed format, the driver will silently switch to low
--

(continues on next page)

(continued from previous page)

```
compression.  
1 = low compression.  
2 = medium compression.  
3 = high compression.
```

High compression takes less bandwidth of course, but it could also introduce some unwanted artefacts. The default is 2, medium compression. See the FAQ on the website for an overview of which modes require compression.

The compression parameter does not apply to the 645 and 646 cameras and OEM models derived from those (only a few). Most cams honour this parameter.

**leds** This settings takes 2 integers, that define the on/off time for the LED (in milliseconds). One of the interesting things that you can do with this is let the LED blink while the camera is in use. This:

```
leds=500,500
```

will blink the LED once every second. But with:

```
leds=0,0
```

the LED never goes on, making it suitable for silent surveillance.

By default the camera's LED is on solid while in use, and turned off when the camera is not used anymore.

This parameter works only with the ToUCam range of cameras (720, 730, 740, 750) and OEMs. For other cameras this command is silently ignored, and the LED cannot be controlled.

Finally: this parameters does not take effect UNTIL the first time you open the camera device. Until then, the LED remains on.

**dev\_hint** A long standing problem with USB devices is their dynamic nature: you never know what device a camera gets assigned; it depends on module load order, the hub configuration, the order in which devices are plugged in, and the phase of the moon (i.e. it can be random). With this option you can give the driver a hint as to what video device node (/dev/videoX) it should use with a specific camera. This is also handy if you have two cameras of the same model.

A camera is specified by its type (the number from the camera model, like PCA645, PCVC750VC, etc) and optionally the serial number (visible in /sys/kernel/debug/usb/devices). A hint consists of a string with the following format:

```
[type[.serialnumber]:]node
```

The square brackets mean that both the type and the serialnumber are optional, but a serialnumber cannot be specified without a type (which would be rather pointless). The serialnumber is separated from the type by a '.'; the node number by a ':'.

This somewhat cryptic syntax is best explained by a few examples:

<code>dev_hint=3,5</code>	The first detected cam gets assigned /dev/video3, the second /dev/video5. Any other cameras will get the first free available slot (see below).
<code>dev_hint=645:1,680:2</code>	The PCA645 camera will get /dev/video1, and a PCVC680 /dev/video2.
<code>dev_hint=645.0123:3,645.4567:0</code>	The PCA645 camera with serialnumber 0123 goes to /dev/video3, the same camera model with the 4567 serial gets /dev/video0.
<code>dev_hint=750:1,4,5,6</code> ↪the	The PCVC750 camera will get /dev/video1, the next 3 Philips cams will use /dev/video4 through /dev/video6.

Some points worth knowing:

- Serialnumbers are case sensitive and must be written full, including leading zeroes (it's treated as a string).
- If a device node is already occupied, registration will fail and the webcam is not available.
- You can have up to 64 video devices; be sure to make enough device nodes in /dev if you want to spread the numbers. After /dev/video9 comes /dev/video10 (not /dev/videoA).
- If a camera does not match any dev\_hint, it will simply get assigned the first available device node, just as it used to be.

**trace** In order to better detect problems, it is now possible to turn on a 'trace' of some of the calls the module makes; it logs all items in your kernel log at debug level.

The trace variable is a bitmask; each bit represents a certain feature. If you want to trace something, look up the bit value(s) in the table below, add the values together and supply that to the trace variable.

Value (dec)	Value (hex)	Description	Default
1	0x1	Module initialization; this will log messages while loading and unloading the module	On
2	0x2	probe() and disconnect() traces	On
4	0x4	Trace open() and close() calls	Off
8	0x8	read(), mmap() and associated ioctl() calls	Off
16	0x10	Memory allocation of buffers, etc.	Off
32	0x20	Showing underflow, overflow and Dumping frame messages	On
64	0x40	Show viewport and image sizes	Off
128	0x80	PWCX debugging	Off

For example, to trace the `open()` & `read()` functions, sum  $8 + 4 = 12$ , so you would supply `trace=12` during `insmod` or `modprobe`. If you want to turn the initialization and probing tracing off, set `trace=0`. The default value for `trace` is 35 (0x23).

Example:

```
# modprobe pwc size=cif fps=15 power_save=1
```

The `fbufs`, `mbufs` and `trace` parameters are global and apply to all connected cameras. Each camera has its own set of buffers.

`size` and `fps` only specify defaults when you `open()` the device; this is to accommodate some tools that don't set the size. You can change these settings after `open()` with the `Video4Linux ioctl()` calls. The default of defaults is QCIF size at 10 fps.

The compression parameter is semiglobal; it sets the initial compression preference for all camera's, but this parameter can be set per camera with the `VID-IOCPWCSCQUAL ioctl()` call.

All parameters are optional.

## Qualcomm Camera Subsystem driver

### Introduction

This file documents the Qualcomm Camera Subsystem driver located under `drivers/media/platform/qcom/camss`.

The current version of the driver supports the Camera Subsystem found on Qualcomm MSM8916/APQ8016 and MSM8996/APQ8096 processors.

The driver implements V4L2, Media controller and V4L2 subdev interfaces. Camera sensor using V4L2 subdev interface in the kernel is supported.

The driver is implemented using as a reference the Qualcomm Camera Subsystem driver for Android as found in Code Aurora<sup>1,2</sup>.

### Qualcomm Camera Subsystem hardware

The Camera Subsystem hardware found on 8x16 / 8x96 processors and supported by the driver consists of:

- 2 / 3 CSIPHY modules. They handle the Physical layer of the CSI2 receivers. A separate camera sensor can be connected to each of the CSIPHY module;
- 2 / 4 CSID (CSI Decoder) modules. They handle the Protocol and Application layer of the CSI2 receivers. A CSID can decode data stream from any of the CSIPHY. Each CSID also contains a TG (Test Generator) block which can generate artificial input data for test purposes;

---

<sup>1</sup> <https://source.codeaurora.org/quic/la/kernel/msm-3.10/>

<sup>2</sup> <https://source.codeaurora.org/quic/la/kernel/msm-3.18/>

- ISPIF (ISP Interface) module. Handles the routing of the data streams from the CSIDs to the inputs of the VFE;
- 1 / 2 VFE (Video Front End) module(s). Contain a pipeline of image processing hardware blocks. The VFE has different input interfaces. The PIX (Pixel) input interface feeds the input data to the image processing pipeline. The image processing pipeline contains also a scale and crop module at the end. Three RDI (Raw Dump Interface) input interfaces bypass the image processing pipeline. The VFE also contains the AXI bus interface which writes the output data to memory.

## Supported functionality

The current version of the driver supports:

- Input from camera sensor via CSIPHY;
- Generation of test input data by the TG in CSID;
- RDI interface of VFE
  - Raw dump of the input data to memory.

Supported formats:

- \* YUYV/UYYV/YVYU/VYUY (packed YUV 4:2:2 - V4L2\_PIX\_FMT\_YUYV / V4L2\_PIX\_FMT\_UYYV / V4L2\_PIX\_FMT\_YVYU / V4L2\_PIX\_FMT\_VYUY);
- \* MIPI RAW8 (8bit Bayer RAW - V4L2\_PIX\_FMT\_SRGGB8 / V4L2\_PIX\_FMT\_SGRBG8 / V4L2\_PIX\_FMT\_SBGGR8);
- \* MIPI RAW10 (10bit packed Bayer RAW - V4L2\_PIX\_FMT\_SBGGR10P / V4L2\_PIX\_FMT\_SGBRG10P / V4L2\_PIX\_FMT\_SGRBG10P / V4L2\_PIX\_FMT\_SRGGB10P / V4L2\_PIX\_FMT\_Y10P);
- \* MIPI RAW12 (12bit packed Bayer RAW - V4L2\_PIX\_FMT\_SRGGB12P / V4L2\_PIX\_FMT\_SGBRG12P / V4L2\_PIX\_FMT\_SGRBG12P / V4L2\_PIX\_FMT\_SRGGB12P).
- \* (8x96 only) MIPI RAW14 (14bit packed Bayer RAW - V4L2\_PIX\_FMT\_SRGGB14P / V4L2\_PIX\_FMT\_SGBRG14P / V4L2\_PIX\_FMT\_SGRBG14P / V4L2\_PIX\_FMT\_SRGGB14P).

- (8x96 only) Format conversion of the input data.

Supported input formats:

- \* MIPI RAW10 (10bit packed Bayer RAW - V4L2\_PIX\_FMT\_SBGGR10P / V4L2\_PIX\_FMT\_Y10P).

Supported output formats:

- \* Plain16 RAW10 (10bit unpacked Bayer RAW - V4L2\_PIX\_FMT\_SBGGR10 / V4L2\_PIX\_FMT\_Y10).

- PIX interface of VFE

- Format conversion of the input data.

Supported input formats:

- \* YUYV/UYVY/YVYU/VYUY (packed YUV 4:2:2 - V4L2\_PIX\_FMT\_YUYV / V4L2\_PIX\_FMT\_UYVY / V4L2\_PIX\_FMT\_YVYU / V4L2\_PIX\_FMT\_VYUY).

Supported output formats:

- \* NV12/NV21 (two plane YUV 4:2:0 - V4L2\_PIX\_FMT\_NV12 / V4L2\_PIX\_FMT\_NV21);
  - \* NV16/NV61 (two plane YUV 4:2:2 - V4L2\_PIX\_FMT\_NV16 / V4L2\_PIX\_FMT\_NV61).
  - \* (8x96 only) YUYV/UYVY/YVYU/VYUY (packed YUV 4:2:2 - V4L2\_PIX\_FMT\_YUYV / V4L2\_PIX\_FMT\_UYVY / V4L2\_PIX\_FMT\_YVYU / V4L2\_PIX\_FMT\_VYUY).
- Scaling support. Configuration of the VFE Encoder Scale module for downscaling with ratio up to 16x.
  - Cropping support. Configuration of the VFE Encoder Crop module.
- Concurrent and independent usage of two (8x96: three) data inputs - could be camera sensors and/or TG.

### Driver Architecture and Design

The driver implements the V4L2 subdev interface. With the goal to model the hardware links between the modules and to expose a clean, logical and usable interface, the driver is split into V4L2 sub-devices as follows (8x16 / 8x96):

- 2 / 3 CSIPHY sub-devices - each CSIPHY is represented by a single sub-device;
- 2 / 4 CSID sub-devices - each CSID is represented by a single sub-device;
- 2 / 4 ISPIF sub-devices - ISPIF is represented by a number of sub-devices equal to the number of CSID sub-devices;
- 4 / 8 VFE sub-devices - VFE is represented by a number of sub-devices equal to the number of the input interfaces (3 RDI and 1 PIX for each VFE).

The considerations to split the driver in this particular way are as follows:

- representing CSIPHY and CSID modules by a separate sub-device for each module allows to model the hardware links between these modules;
- representing VFE by a separate sub-devices for each input interface allows to use the input interfaces concurrently and independently as this is supported by the hardware;
- representing ISPIF by a number of sub-devices equal to the number of CSID sub-devices allows to create linear media controller pipelines when using two cameras simultaneously. This avoids branches in the pipelines which otherwise will require a) userspace and b) media framework (e.g. power on/off operations) to make assumptions about the data flow from a sink pad to a source pad on a single media entity.

Each VFE sub-device is linked to a separate video device node.

The media controller pipeline graph is as follows (with connected two / three OV5645 camera sensors):

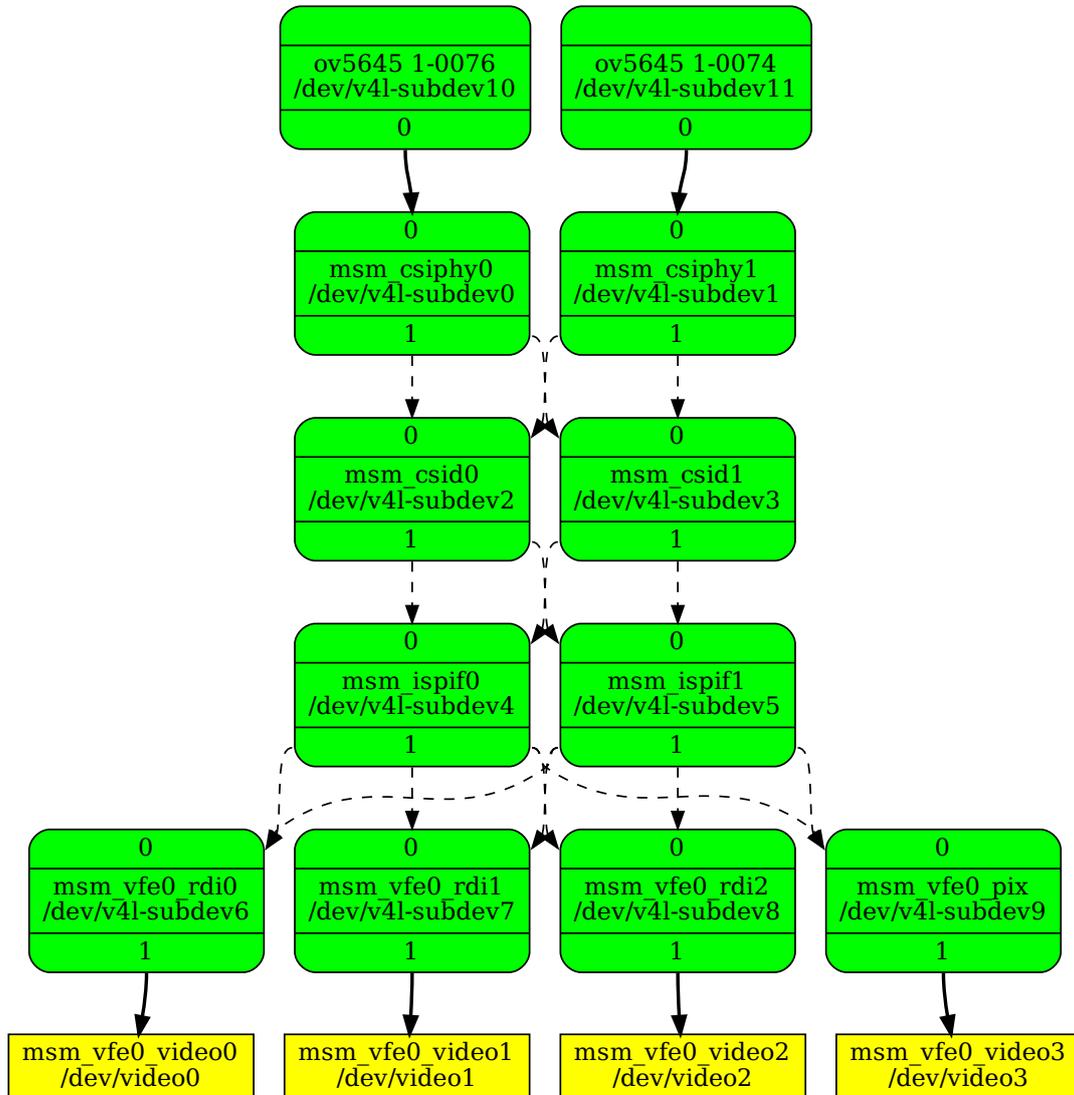


Fig. 5: Media pipeline graph 8x16

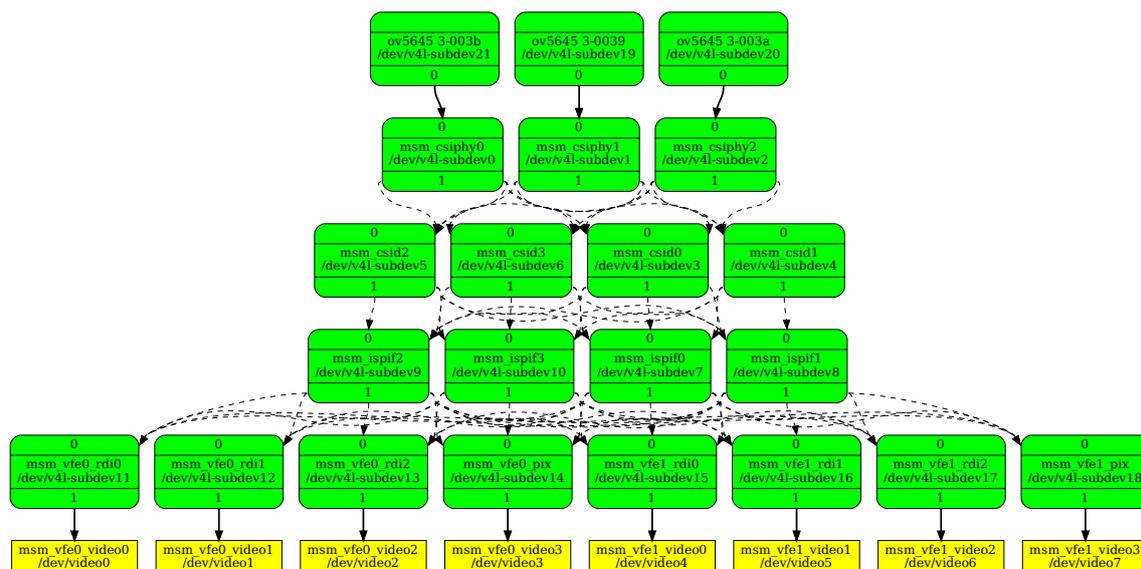


Fig. 6: Media pipeline graph 8x96

## Implementation

Runtime configuration of the hardware (updating settings while streaming) is not required to implement the currently supported functionality. The complete configuration on each hardware module is applied on STREAMON ioctl based on the current active media links, formats and controls set.

The output size of the scaler module in the VFE is configured with the actual compose selection rectangle on the sink pad of the 'msm\_vfe0\_pix' entity.

The crop output area of the crop module in the VFE is configured with the actual crop selection rectangle on the source pad of the 'msm\_vfe0\_pix' entity.

## Documentation

APQ8016 Specification: <https://developer.qualcomm.com/download/sd410/snapdragon-410-processor-device-specification.pdf> Referenced 2016-11-24.

APQ8096 Specification: <https://developer.qualcomm.com/download/sd820e/qualcomm-snapdragon-820e-processor-apq8096sge-device-specification.pdf> Referenced 2018-06-22.

## References

### Renesas R-Car Fine Display Processor (FDP1) Driver

The R-Car FDP1 driver implements driver-specific controls as follows.

**V4L2\_CID\_DEINTERLACING\_MODE (menu)** The video deinterlacing mode (such as Bob, Weave, ...). The R-Car FDP1 driver implements the following modes.

"Progressive" (0)	The input image video stream is progressive (not interlaced). No deinterlacing is performed. Apart from (optional) format and encoding conversion output frames are identical to the input frames.
"Adaptive 2D/3D" (1)	Motion adaptive version of 2D and 3D deinterlacing. Use 3D deinterlacing in the presence of fast motion and 2D deinterlacing with diagonal interpolation otherwise.
"Fixed 2D" (2)	The current field is scaled vertically by averaging adjacent lines to recover missing lines. This method is also known as blending or Line Averaging (LAV).
"Fixed 3D" (3)	The previous and next fields are averaged to recover lines missing from the current field. This method is also known as Field Averaging (FAV).
"Previous field" (4)	The current field is weaved with the previous field, i.e. the previous field is used to fill missing lines from the current field. This method is also known as weave deinterlacing.
"Next field" (5)	The current field is weaved with the next field, i.e. the next field is used to fill missing lines from the current field. This method is also known as weave deinterlacing.

### The saa7134 driver

Author Gerd Hoffmann

This is a v4l2/oss device driver for saa7130/33/34/35 based capture / TV boards.

### Status

Almost everything is working. video, sound, tuner, radio, mpeg ts, ...

As with bttv, card-specific tweaks are needed. Check CARDLIST for a list of known TV cards and saa7134-cards.c for the drivers card configuration info.

### Build

Once you pick up a Kernel source, you should configure, build, install and boot the new kernel. You' ll need at least these config options:

```
./scripts/config -e PCI
./scripts/config -e INPUT
./scripts/config -m I2C
./scripts/config -m MEDIA_SUPPORT
./scripts/config -e MEDIA_PCI_SUPPORT
./scripts/config -e MEDIA_ANALOG_TV_SUPPORT
./scripts/config -e MEDIA_DIGITAL_TV_SUPPORT
./scripts/config -e MEDIA_RADIO_SUPPORT
./scripts/config -e RC_CORE
./scripts/config -e MEDIA_SUBDRV_AUTOSELECT
./scripts/config -m VIDEO_SAA7134
./scripts/config -e SAA7134_ALSA
./scripts/config -e VIDEO_SAA7134_RC
./scripts/config -e VIDEO_SAA7134_DVB
./scripts/config -e VIDEO_SAA7134_G07007
```

To build and install, you should run:

```
make && make modules_install && make install
```

Once the new Kernel is booted, saa7134 driver should be loaded automatically.

Depending on the card you might have to pass card=<nr> as insmod option. If so, please check SAA7134 cards list for valid choices.

Once you have your card type number, you can pass a modules configuration via a file (usually, it is either /etc/modules.conf or some file at /etc/modules-load.d/, but the actual place depends on your distribution), with this content:

```
options saa7134 card=13 # Assuming that your card type is #13
```

### Changes / Fixes

Please mail to linux-media AT vger.kernel.org unified diffs against the linux media git tree:

[https://git.linuxtv.org/media\\_tree.git/](https://git.linuxtv.org/media_tree.git/)

This is done by committing a patch at a clone of the git tree and submitting the patch using git send-email. Don' t forget to describe at the lots what it changes / which problem it fixes / whatever it is good for ...

## Known Problems

- The tuner for the flyvideos isn't detected automatically and the default might not work for you depending on which version you have. There is a tuner=insmod option to override the driver's default.

## Credits

[andrew.stevens@philips.com](mailto:andrew.stevens@philips.com) + [werner.leebe@philips.com](mailto:werner.leebe@philips.com) for providing saa7134 hardware specs and sample board.

## The Silicon Labs Si470x FM Radio Receivers driver

Copyright © 2009 Tobias Lorenz <[tobias.lorenz@gmx.net](mailto:tobias.lorenz@gmx.net)>

## Information from Silicon Labs

Silicon Laboratories is the manufacturer of the radio ICs, that nowadays are the most often used radio receivers in cell phones. Usually they are connected with I2C. But SiLabs also provides a reference design, which integrates this IC, together with a small microcontroller C8051F321, to form a USB radio. Part of this reference design is also a radio application in binary and source code. The software also contains an automatic firmware upgrade to the most current version. Information on these can be downloaded here: <http://www.silabs.com/usbradio>

## Supported ICs

The following ICs have a very similar register set, so that they are or will be supported somewhen by the driver:

- Si4700: FM radio receiver
- Si4701: FM radio receiver, RDS Support
- Si4702: FM radio receiver
- Si4703: FM radio receiver, RDS Support
- Si4704: FM radio receiver, no external antenna required
- Si4705: FM radio receiver, no external antenna required, RDS support, Dig I/O
- **Si4706: Enhanced FM RDS/TMC radio receiver, no external antenna required, RDS Support**
- Si4707: Dedicated weather band radio receiver with SAME decoder, RDS Support
- Si4708: Smallest FM receivers
- Si4709: Smallest FM receivers, RDS Support

More information on these can be downloaded here: <http://www.silabs.com/products/mcu/Pages/USBFMRadioRD.aspx>

### Supported USB devices

Currently the following USB radios (vendor:product) with the Silicon Labs si470x chips are known to work:

- 10c4:818a: Silicon Labs USB FM Radio Reference Design
- 06e1:a155: ADS/Tech FM Radio Receiver (formerly Instant FM Music) (RDX-155-EF)
- 1b80:d700: KWorld USB FM Radio SnapMusic Mobile 700 (FM700)
- 10c5:819a: Sanei Electric, Inc. FM USB Radio (sold as DealExtreme.com PCear)

### Software

Testing is usually done with most application under Debian/testing:

- `fmtools` - Utility for managing FM tuner cards
- `gnomeradio` - FM-radio tuner for the GNOME desktop
- `gradio` - GTK FM radio tuner
- `kradio` - Comfortable Radio Application for KDE
- `radio` - ncurses-based radio application
- `mplayer` - The Ultimate Movie Player For Linux
- `v4l2-ctl` - Collection of command line video4linux utilities

For example, you can use:

```
v4l2-ctl -d /dev/radio0 --set-ctrl=volume=10,mute=0 --set-freq=95.21 --all
```

There is also a library `libv4l`, which can be used. It's going to have a function for frequency seeking, either by using hardware functionality as in `radio-si470x` or by implementing a function as we currently have in every of the mentioned programs. Somewhen the radio programs should make use of `libv4l`.

For processing RDS information, there is a project ongoing at: <http://rdsd.berlios.de/>

There is currently no project for making TMC sentences human readable.

## Audio Listing

USB Audio is provided by the ALSA `snd_usb_audio` module. It is recommended to also select `SND_USB_AUDIO`, as this is required to get sound from the radio. For listing you have to redirect the sound, for example using one of the following commands. Please adjust the audio devices to your needs (`/dev/dsp*` and `hw:x,x`).

If you just want to test audio (very poor quality):

```
cat /dev/dsp1 > /dev/dsp
```

If you use `sox` + `OSS` try:

```
sox -2 --endian little -r 96000 -t oss /dev/dsp1 -t oss /dev/dsp
```

or using `sox` + `alsa`:

```
sox --endian little -c 2 -S -r 96000 -t alsa hw:1 -t alsa -r 96000 hw:0
```

If you use `arts` try:

```
arecord -D hw:1,0 -r96000 -c2 -f S16_LE | artsdsp aplay -B -
```

If you use `mplayer` try:

```
mplayer -radio adevice=hw=1.0:arate=96000 \  
        -rawaudio rate=96000 \  
        radio://<frequency>/capture
```

## Module Parameters

After loading the module, you still have access to some of them in the `sysfs` mount under `/sys/module/radio_si470x/parameters`. The contents of read-only files (0444) are not updated, even if `space`, `band` and `de` are changed using private video controls. The others are runtime changeable.

## Errors

Increase `tune_timeout`, if you often get `-EIO` errors.

When timed out or band limit is reached, `hw_freq_seek` returns `-EAGAIN`.

If you get any errors from `snd_usb_audio`, please report them to the ALSA people.

### Open Issues

V4L minor device allocation and parameter setting is not perfect. A solution is currently under discussion.

There is an USB interface for downloading/uploading new firmware images. Support for it can be implemented using the `request_firmware` interface.

There is a RDS interrupt mode. The driver is already using the same interface for polling RDS information, but is currently not using the interrupt mode.

There is a LED interface, which can be used to override the LED control programmed in the firmware. This can be made available using the LED support functions in the kernel.

### Other useful information and links

<http://www.silabs.com/usbradio>

### The Silicon Labs Si4713 FM Radio Transmitter Driver

Copyright © 2009 Nokia Corporation

Contact: Eduardo Valentin <[eduardo.valentin@nokia.com](mailto:eduardo.valentin@nokia.com)>

### Information about the Device

This chip is a Silicon Labs product. It is a I2C device, currently on 0x63 address. Basically, it has transmission and signal noise level measurement features.

The Si4713 integrates transmit functions for FM broadcast stereo transmission. The chip also allows integrated receive power scanning to identify low signal power FM channels.

The chip is programmed using commands and responses. There are also several properties which can change the behavior of this chip.

Users must comply with local regulations on radio frequency (RF) transmission.

### Device driver description

There are two modules to handle this device. One is a I2C device driver and the other is a platform driver.

The I2C device driver exports a v4l2-subdev interface to the kernel. All properties can also be accessed by v4l2 extended controls interface, by using the v4l2-subdev calls (`g_ext_ctrls`, `s_ext_ctrls`).

The platform device driver exports a v4l2 radio device interface to user land. So, it uses the I2C device driver as a sub device in order to send the user commands to the actual device. Basically it is a wrapper to the I2C device driver.

Applications can use v4l2 radio API to specify frequency of operation, mute state, etc. But mostly of its properties will be present in the extended controls.

When the v4l2 mute property is set to 1 (true), the driver will turn the chip off.

## Properties description

The properties can be accessed using v4l2 extended controls. Here is an output from v4l2-ctl util:

```
/ # v4l2-ctl -d /dev/radio0 --all -L
Driver Info:
  Driver name      : radio-si4713
  Card type       : Silicon Labs Si4713 Modulator
  Bus info        :
  Driver version  : 0
  Capabilities    : 0x00080800
                   RDS Output
                   Modulator
Audio output: 0 (FM Modulator Audio Out)
Frequency: 1408000 (88.000000 MHz)
Video Standard = 0x00000000
Modulator:
  Name             : FM Modulator
  Capabilities     : 62.5 Hz stereo rds
  Frequency range  : 76.0 MHz - 108.0 MHz
  Subchannel modulation: stereo+rds

User Controls

                mute (bool) : default=1 value=0

FM Radio Modulator Controls

  rds_signal_deviation (int) : min=0 max=90000 step=10 default=200
↪value=200 flags=slider
  rds_program_id (int)      : min=0 max=65535 step=1 default=0
↪value=0
  rds_program_type (int)    : min=0 max=31 step=1 default=0 value=0
  rds_ps_name (str)        : min=0 max=96 step=8 value='si4713 '
  rds_radio_text (str)     : min=0 max=384 step=32 value=''
audio_limiter_feature_enabled (bool) : default=1 value=1
audio_limiter_release_time (int) : min=250 max=102390 step=50
↪default=5010 value=5010 flags=slider
  audio_limiter_deviation (int) : min=0 max=90000 step=10
↪default=66250 value=66250 flags=slider
audio_compression_feature_enabl (bool) : default=1 value=1
  audio_compression_gain (int) : min=0 max=20 step=1 default=15
↪value=15 flags=slider
audio_compression_threshold (int) : min=-40 max=0 step=1 default=-40
↪value=-40 flags=slider
audio_compression_attack_time (int) : min=0 max=5000 step=500 default=0
↪value=0 flags=slider
audio_compression_release_time (int) : min=100000 max=1000000 step=100000
↪default=1000000 value=1000000 flags=slider
pilot_tone_feature_enabled (bool) : default=1 value=1
```

(continues on next page)

(continued from previous page)

```
    pilot_tone_deviation (int) : min=0 max=90000 step=10 default=6750 ↵
↵value=6750 flags=slider
    pilot_tone_frequency (int) : min=0 max=19000 step=1 default=19000 ↵
↵value=19000 flags=slider
    pre_emphasis_settings (menu) : min=0 max=2 default=1 value=1
    tune_power_level (int) : min=0 max=120 step=1 default=88 value=88 ↵
↵flags=slider
    tune_antenna_capacitor (int) : min=0 max=191 step=1 default=0 ↵
↵value=110 flags=slider
```

Here is a summary of them:

- Pilot is an audible tone sent by the device.
- `pilot_frequency` - Configures the frequency of the stereo pilot tone.
- `pilot_deviation` - Configures pilot tone frequency deviation level.
- `pilot_enabled` - Enables or disables the pilot tone feature.
- The si4713 device is capable of applying audio compression to the transmitted signal.
- `acomp_enabled` - Enables or disables the audio dynamic range control feature.
- `acomp_gain` - Sets the gain for audio dynamic range control.
- `acomp_threshold` - Sets the threshold level for audio dynamic range control.
- `acomp_attack_time` - Sets the attack time for audio dynamic range control.
- `acomp_release_time` - Sets the release time for audio dynamic range control.
- Limiter setups audio deviation limiter feature. Once a over deviation occurs, it is possible to adjust the front-end gain of the audio input and always prevent over deviation.
- `limiter_enabled` - Enables or disables the limiter feature.
- `limiter_deviation` - Configures audio frequency deviation level.
- `limiter_release_time` - Sets the limiter release time.
- Tuning power
- `power_level` - Sets the output power level for signal transmission. `antenna_capacitor` - This selects the value of antenna tuning capacitor manually or automatically if set to zero.
- RDS related
- `rds_ps_name` - Sets the RDS ps name field for transmission.
- `rds_radio_text` - Sets the RDS radio text for transmission.
- `rds_pi` - Sets the RDS PI field for transmission.
- `rds_pty` - Sets the RDS PTY field for transmission.
- Region related
- `preemphasis` - sets the preemphasis to be applied for transmission.

## RNL

This device also has an interface to measure received noise level. To do that, you should ioctl the device node. Here is an code of example:

```
int main (int argc, char *argv[])
{
    struct si4713_rnl rnl;
    int fd = open("/dev/radio0", O_RDWR);
    int rval;

    if (argc < 2)
        return -EINVAL;

    if (fd < 0)
        return fd;

    sscanf(argv[1], "%d", &rnl.frequency);

    rval = ioctl(fd, SI4713_IOC_MEASURE_RNL, &rnl);
    if (rval < 0)
        return rval;

    printf("received noise level: %d\n", rnl.rnl);

    close(fd);
}
```

The struct `si4713_rnl` and `SI4713_IOC_MEASURE_RNL` are defined under `include/linux/platform_data/media/si4713.h`.

## Stereo/Mono and RDS subchannels

The device can also be configured using the available sub channels for transmission. To do that use `S/G_MODULATOR` ioctl and configure `txsubchans` properly. Refer to the V4L2 API specification for proper use of this ioctl.

## Testing

Testing is usually done with `v4l2-ctl` utility for managing FM tuner cards. The tool can be found in `v4l-dvb` repository under `v4l2-apps/util` directory.

Example for setting `rds ps` name:

```
# v4l2-ctl -d /dev/radio0 --set-ctrl=rds_ps_name="Dummy"
```

### The SI476x Driver

Copyright © 2013 Andrey Smirnov <[andrew.smirnov@gmail.com](mailto:andrew.smirnov@gmail.com)>

#### TODO for the driver

- According to the SiLabs' datasheet it is possible to update the firmware of the radio chip in the run-time, thus bringing it to the most recent version. Unfortunately I couldn't find any mentioning of the said firmware update for the old chips that I tested the driver against, so for chips like that the driver only exposes the old functionality.

#### Parameters exposed over debugfs

SI476x allow user to get multiple characteristics that can be very useful for EoL testing/RF performance estimation, parameters that have very little to do with V4L2 subsystem. Such parameters are exposed via debugfs and can be accessed via regular file I/O operations.

The drivers exposes following files:

- `/sys/kernel/debug/<device-name>/acf` This file contains ACF(Automatically Controlled Features) status information. The contents of the file is binary data of the following layout:

Offset	Name	Description
0x00	blend_int	Flag, set when stereo separation has crossed below the blend threshold
0x01	hblend_int	Flag, set when HiBlend cutoff frequency is lower than threshold
0x02	hicut_int	Flag, set when HiCut cutoff frequency is lower than threshold
0x03	chbw_int	Flag, set when channel filter bandwidth is less than threshold
0x04	softmute_int	Flag indicating that softmute attenuation has increased above softmute threshold
0x05	smute	0 - Audio is not soft muted 1 - Audio is soft muted
0x06	smattn	Soft mute attenuation level in dB
0x07	chbw	Channel filter bandwidth in kHz
0x08	hicut	HiCut cutoff frequency in units of 100Hz
0x09	hblend	HiBlend cutoff frequency in units of 100 Hz
0x10	pilot	0 - Stereo pilot is not present 1 - Stereo pilot is present
0x11	stblend	Stereo blend in %

- `/sys/kernel/debug/<device-name>/rds_blkcnt` This file contains statistics about RDS receptions. It's binary data has the following layout:

Offset	Name	Description
0x00	expected	Number of expected RDS blocks
0x02	received	Number of received RDS blocks
0x04	uncorrectable	Number of uncorrectable RDS blocks

- /sys/kernel/debug/<device-name>/agc This file contains information about parameters pertaining to AGC(Automatic Gain Control)

The layout is:

Offset	Name	Description
0x00	mxhi	0 - FM Mixer PD high threshold is not tripped 1 - FM Mixer PD high threshold is tripped
0x01	mxlo	ditto for FM Mixer PD low
0x02	lnahi	ditto for FM LNA PD high
0x03	lnalo	ditto for FM LNA PD low
0x04	fmagc1	FMAGC1 attenuator resistance (see datasheet for more detail)
0x05	fmagc2	ditto for FMAGC2
0x06	pgagain	PGA gain in dB
0x07	fmwblang	FM/WB LNA Gain in dB

- /sys/kernel/debug/<device-name>/rsq This file contains information about parameters pertaining to RSQ(Received Signal Quality)

The layout is:

Offset	Name	Description
0x00	multhint	0 - multipath value has not crossed the Multipath high threshold 1 - multipath value has crossed the Multipath high threshold
0x01	multlint	ditto for Multipath low threshold
0x02	snrhint	0 - received signal' s SNR has not crossed high threshold 1 - received signal' s SNR has crossed high threshold
0x03	snrlint	ditto for low threshold
0x04	rssihint	ditto for RSSI high threshold
0x05	rssilint	ditto for RSSI low threshold
0x06	bltf	Flag indicating if seek command reached/wrapped seek band limit
0x07	snr_ready	Indicates that SNR metrics is ready
0x08	rssiready	ditto for RSSI metrics
0x09	injside	0 - Low-side injection is being used 1 - High-side injection is used
0x10	afcr1	Flag indicating if AFC rails
0x11	valid	Flag indicating if channel is valid
0x12	readfreq	Current tuned frequency
0x14	freqoff	Signed frequency offset in units of 2ppm
0x15	rssi	Signed value of RSSI in dBuV
0x16	snr	Signed RF SNR in dB
0x17	issi	Signed Image Strength Signal indicator
0x18	lassi	Signed Low side adjacent Channel Strength indicator
0x19	hassi	ditto fpr High side
0x20	mult	Multipath indicator
0x21	dev	Frequency deviation
0x24	assi	Adjacent channel SSI
0x25	usn	Ultrasonic noise indicator
0x26	pilotdev	Pilot deviation in units of 100 Hz
0x27	rdsdev	ditto for RDS
0x28	assidev	ditto for ASSI
0x29	strongdev	Frequency deviation
0x30	rdspi	RDS PI code

- /sys/kernel/debug/<device-name>/rsq\_primary This file contains information about parameters pertaining to RSQ(Received Signal Quality) for primary tuner only. Layout is as the one above.

## The Virtual Media Controller Driver (vimc)

The vimc driver emulates complex video hardware using the V4L2 API and the Media API. It has a capture device and three subdevices: sensor, debayer and scaler.

### Topology

The topology is hardcoded, although you could modify it in vimc-core and recompile the driver to achieve your own topology. This is the default topology:

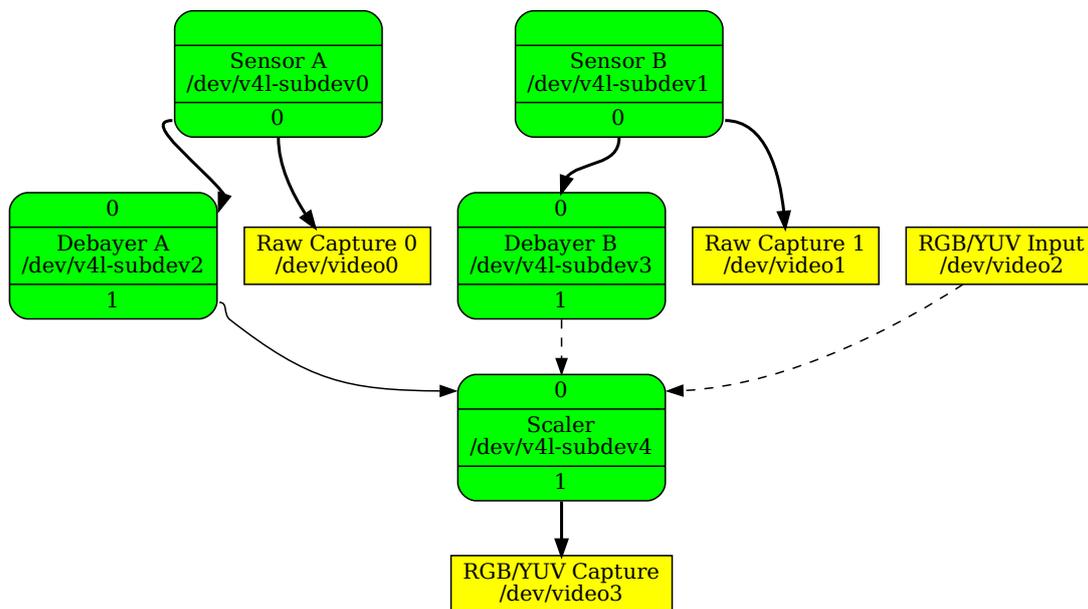


Fig. 7: Media pipeline graph on vimc

### Configuring the topology

Each subdevice will come with its default configuration (pixelformat, height, width, ...). One needs to configure the topology in order to match the configuration on each linked subdevice to stream frames through the pipeline. If the configuration doesn't match, the stream will fail. The v4l-utils package is a bundle of user-space applications, that comes with media-ctl and v4l2-ctl that can be used to configure the vimc configuration. This sequence of commands fits for the default topology:

```
media-ctl -d platform:vimc -V '"Sensor A":0[fmt:SBGGR8_1X8/640x480]'
media-ctl -d platform:vimc -V '"Debayer A":0[fmt:SBGGR8_1X8/640x480]'
media-ctl -d platform:vimc -V '"Sensor B":0[fmt:SBGGR8_1X8/640x480]'
media-ctl -d platform:vimc -V '"Debayer B":0[fmt:SBGGR8_1X8/640x480]'
v4l2-ctl -z platform:vimc -d "RGB/YUV Capture" -v width=1920,height=1440
```

(continues on next page)

(continued from previous page)

```
v4l2-ctl -z platform:vimc -d "Raw Capture 0" -v pixelformat=BA81
v4l2-ctl -z platform:vimc -d "Raw Capture 1" -v pixelformat=BA81
```

### Subdevices

Subdevices define the behavior of an entity in the topology. Depending on the subdevice, the entity can have multiple pads of type source or sink.

**vimc-sensor:** Generates images in several formats using video test pattern generator. Exposes:

- 1 Pad source

**vimc-debayer:** Transforms images in bayer format into a non-bayer format. Exposes:

- 1 Pad sink
- 1 Pad source

**vimc-scaler:** Scale up the image by a factor of 3. E.g.: a 640x480 image becomes a 1920x1440 image. (this value can be configured, see at Module options). Exposes:

- 1 Pad sink
- 1 Pad source

**vimc-capture:** Exposes node /dev/videoX to allow userspace to capture the stream. Exposes:

- 1 Pad sink
- 1 Pad source

### Module options

Vimc has a module parameter to configure the driver.

- `sca_mult=<unsigned int>`

Image size multiplier factor to be used to multiply both width and height, so the image size will be `sca_mult^2` bigger than the original one. Currently, only supports scaling up (the default value is 3).

## **The Virtual Video Test Driver (vivid)**

This driver emulates video4linux hardware of various types: video capture, video output, vbi capture and output, metadata capture and output, radio receivers and transmitters, touch capture and a software defined radio receiver. In addition a simple framebuffer device is available for testing capture and output overlays.

Up to 64 vivid instances can be created, each with up to 16 inputs and 16 outputs.

Each input can be a webcam, TV capture device, S-Video capture device or an HDMI capture device. Each output can be an S-Video output device or an HDMI output device.

These inputs and outputs act exactly as a real hardware device would behave. This allows you to use this driver as a test input for application development, since you can test the various features without requiring special hardware.

This document describes the features implemented by this driver:

- Support for read()/write(), MMAP, USERPTR and DMABUF streaming I/O.
- A large list of test patterns and variations thereof
- Working brightness, contrast, saturation and hue controls
- Support for the alpha color component
- Full colorspace support, including limited/full RGB range
- All possible control types are present
- Support for various pixel aspect ratios and video aspect ratios
- Error injection to test what happens if errors occur
- Supports crop/compose/scale in any combination for both input and output
- Can emulate up to 4K resolutions
- All Field settings are supported for testing interlaced capturing
- Supports all standard YUV and RGB formats, including two multiplanar YUV formats
- Raw and Sliced VBI capture and output support
- Radio receiver and transmitter support, including RDS support
- Software defined radio (SDR) support
- Capture and output overlay support
- Metadata capture and output support
- Touch capture support

These features will be described in more detail below.

### Configuring the driver

By default the driver will create a single instance that has a video capture device with webcam, TV, S-Video and HDMI inputs, a video output device with S-Video and HDMI outputs, one vbi capture device, one vbi output device, one radio receiver device, one radio transmitter device and one SDR device.

The number of instances, devices, video inputs and outputs and their types are all configurable using the following module options:

- `n_devs`:
  - number of driver instances to create. By default set to 1. Up to 64 instances can be created.
- `node_types`:
  - which devices should each driver instance create. An array of hexadecimal values, one for each instance. The default is 0x1d3d. Each value is a bitmask with the following meaning:
    - bit 0: Video Capture node
    - bit 2-3: VBI Capture node: 0 = none, 1 = raw vbi, 2 = sliced vbi, 3 = both
    - bit 4: Radio Receiver node
    - bit 5: Software Defined Radio Receiver node
    - bit 8: Video Output node
    - bit 10-11: VBI Output node: 0 = none, 1 = raw vbi, 2 = sliced vbi, 3 = both
    - bit 12: Radio Transmitter node
    - bit 16: Framebuffer for testing overlays
    - bit 17: Metadata Capture node
    - bit 18: Metadata Output node
    - bit 19: Touch Capture node

So to create four instances, the first two with just one video capture device, the second two with just one video output device you would pass these module options to `vivid`:

```
n_devs=4 node_types=0x1,0x1,0x100,0x100
```

- `num_inputs`:
  - the number of inputs, one for each instance. By default 4 inputs are created for each video capture device. At most 16 inputs can be created, and there must be at least one.
- `input_types`:
  - the input types for each instance, the default is 0xe4. This defines what the type of each input is when the inputs are created for each

driver instance. This is a hexadecimal value with up to 16 pairs of bits, each pair gives the type and bits 0-1 map to input 0, bits 2-3 map to input 1, 30-31 map to input 15. Each pair of bits has the following meaning:

- 00: this is a webcam input
- 01: this is a TV tuner input
- 10: this is an S-Video input
- 11: this is an HDMI input

So to create a video capture device with 8 inputs where input 0 is a TV tuner, inputs 1-3 are S-Video inputs and inputs 4-7 are HDMI inputs you would use the following module options:

```
num_inputs=8 input_types=0xffa9
```

- **num\_outputs:**

the number of outputs, one for each instance. By default 2 outputs are created for each video output device. At most 16 outputs can be created, and there must be at least one.

- **output\_types:**

the output types for each instance, the default is 0x02. This defines what the type of each output is when the outputs are created for each driver instance. This is a hexadecimal value with up to 16 bits, each bit gives the type and bit 0 maps to output 0, bit 1 maps to output 1, bit 15 maps to output 15. The meaning of each bit is as follows:

- 0: this is an S-Video output
- 1: this is an HDMI output

So to create a video output device with 8 outputs where outputs 0-3 are S-Video outputs and outputs 4-7 are HDMI outputs you would use the following module options:

```
num_outputs=8 output_types=0xf0
```

- **vid\_cap\_nr:**

give the desired videoX start number for each video capture device. The default is -1 which will just take the first free number. This allows you to map capture video nodes to specific videoX device nodes. Example:

```
n_devs=4 vid_cap_nr=2,4,6,8
```

This will attempt to assign /dev/video2 for the video capture device of the first vivid instance, video4 for the next up to video8 for the last instance. If it can't succeed, then it will just take the next free number.

- **vid\_out\_nr:**

give the desired videoX start number for each video output device. The default is -1 which will just take the first free number.

- `vbi_cap_nr`:

give the desired vbiX start number for each vbi capture device. The default is -1 which will just take the first free number.

- `vbi_out_nr`:

give the desired vbiX start number for each vbi output device. The default is -1 which will just take the first free number.

- `radio_rx_nr`:

give the desired radioX start number for each radio receiver device. The default is -1 which will just take the first free number.

- `radio_tx_nr`:

give the desired radioX start number for each radio transmitter device. The default is -1 which will just take the first free number.

- `sdr_cap_nr`:

give the desired swradioX start number for each SDR capture device. The default is -1 which will just take the first free number.

- `meta_cap_nr`:

give the desired videoX start number for each metadata capture device. The default is -1 which will just take the first free number.

- `meta_out_nr`:

give the desired videoX start number for each metadata output device. The default is -1 which will just take the first free number.

- `touch_cap_nr`:

give the desired v4l-touchX start number for each touch capture device. The default is -1 which will just take the first free number.

- `ccs_cap_mode`:

specify the allowed video capture crop/compose/scaling combination for each driver instance. Video capture devices can have any combination of cropping, composing and scaling capabilities and this will tell the vivid driver which of those it should emulate. By default the user can select this through controls.

The value is either -1 (controlled by the user) or a set of three bits, each enabling (1) or disabling (0) one of the features:

- bit 0:

Enable crop support. Cropping will take only part of the incoming picture.

- bit 1:

Enable compose support. Composing will copy the incoming picture into a larger buffer.

- bit 2:

Enable scaling support. Scaling can scale the incoming picture. The scaler of the vivid driver can enlarge up or down to four times the original size. The scaler is very simple and low-quality. Simplicity and speed were key, not quality.

Note that this value is ignored by webcam inputs: those enumerate discrete framesizes and that is incompatible with cropping, composing or scaling.

- `ccs_out_mode`:

specify the allowed video output crop/compose/scaling combination for each driver instance. Video output devices can have any combination of cropping, composing and scaling capabilities and this will tell the vivid driver which of those it should emulate. By default the user can select this through controls.

The value is either -1 (controlled by the user) or a set of three bits, each enabling (1) or disabling (0) one of the features:

- bit 0:

Enable crop support. Cropping will take only part of the outgoing buffer.

- bit 1:

Enable compose support. Composing will copy the incoming buffer into a larger picture frame.

- bit 2:

Enable scaling support. Scaling can scale the incoming buffer. The scaler of the vivid driver can enlarge up or down to four times the original size. The scaler is very simple and low-quality. Simplicity and speed were key, not quality.

- `multiplanar`:

select whether each device instance supports multi-planar formats, and thus the V4L2 multi-planar API. By default device instances are single-planar.

This module option can override that for each instance. Values are:

- 1: this is a single-planar instance.
- 2: this is a multi-planar instance.

- `vivid_debug`:

enable driver debugging info

- `no_error_inj`:

if set disable the error injecting controls. This option is needed in order to run a tool like v4l2-compliance. Tools like that exercise all

controls including a control like ‘Disconnect’ which emulates a USB disconnect, making the device inaccessible and so all tests that v4l2-compliance is doing will fail afterwards.

There may be other situations as well where you want to disable the error injection support of vivid. When this option is set, then the controls that select crop, compose and scale behavior are also removed. Unless overridden by `ccs_cap_mode` and/or `ccs_out_mode` the will default to enabling crop, compose and scaling.

- allocators:

memory allocator selection, default is 0. It specifies the way buffers will be allocated.

- 0: `vmalloc`
- 1: `dma-contig`

Taken together, all these module options allow you to precisely customize the driver behavior and test your application with all sorts of permutations. It is also very suitable to emulate hardware that is not yet available, e.g. when developing software for a new upcoming device.

## Video Capture

This is probably the most frequently used feature. The video capture device can be configured by using the module options `num_inputs`, `input_types` and `ccs_cap_mode` (see section 1 for more detailed information), but by default four inputs are configured: a webcam, a TV tuner, an S-Video and an HDMI input, one input for each input type. Those are described in more detail below.

Special attention has been given to the rate at which new frames become available. The jitter will be around 1 jiffie (that depends on the HZ configuration of your kernel, so usually 1/100, 1/250 or 1/1000 of a second), but the long-term behavior is exactly following the framerate. So a framerate of 59.94 Hz is really different from 60 Hz. If the framerate exceeds your kernel’s HZ value, then you will get dropped frames, but the frame/field sequence counting will keep track of that so the sequence count will skip whenever frames are dropped.

## Webcam Input

The webcam input supports three framesizes: 320x180, 640x360 and 1280x720. It supports frames per second settings of 10, 15, 25, 30, 50 and 60 fps. Which ones are available depends on the chosen framesize: the larger the framesize, the lower the maximum frames per second.

The initially selected colorspace when you switch to the webcam input will be sRGB.

## TV and S-Video Inputs

The only difference between the TV and S-Video input is that the TV has a tuner. Otherwise they behave identically.

These inputs support audio inputs as well: one TV and one Line-In. They both support all TV standards. If the standard is queried, then the Vivid controls 'Standard Signal Mode' and 'Standard' determine what the result will be.

These inputs support all combinations of the field setting. Special care has been taken to faithfully reproduce how fields are handled for the different TV standards. This is particularly noticeable when generating a horizontally moving image so the temporal effect of using interlaced formats becomes clearly visible. For 50 Hz standards the top field is the oldest and the bottom field is the newest in time. For 60 Hz standards that is reversed: the bottom field is the oldest and the top field is the newest in time.

When you start capturing in `V4L2_FIELD_ALTERNATE` mode the first buffer will contain the top field for 50 Hz standards and the bottom field for 60 Hz standards. This is what capture hardware does as well.

Finally, for PAL/SECAM standards the first half of the top line contains noise. This simulates the Wide Screen Signal that is commonly placed there.

The initially selected colorspace when you switch to the TV or S-Video input will be SMPTE-170M.

The pixel aspect ratio will depend on the TV standard. The video aspect ratio can be selected through the 'Standard Aspect Ratio' Vivid control. Choices are '4x3', '16x9' which will give letterboxed widescreen video and '16x9 Anamorphic' which will give full screen squashed anamorphic widescreen video that will need to be scaled accordingly.

The TV 'tuner' supports a frequency range of 44-958 MHz. Channels are available every 6 MHz, starting from 49.25 MHz. For each channel the generated image will be in color for the +/- 0.25 MHz around it, and in grayscale for +/- 1 MHz around the channel. Beyond that it is just noise. The `VIDIOC_G_TUNER` ioctl will return 100% signal strength for +/- 0.25 MHz and 50% for +/- 1 MHz. It will also return correct `afc` values to show whether the frequency is too low or too high.

The audio subchannels that are returned are MONO for the +/- 1 MHz range around a valid channel frequency. When the frequency is within +/- 0.25 MHz of the channel it will return either MONO, STEREO, either MONO | SAP (for NTSC) or LANG1 | LANG2 (for others), or STEREO | SAP.

Which one is returned depends on the chosen channel, each next valid channel will cycle through the possible audio subchannel combinations. This allows you to test the various combinations by just switching channels..

Finally, for these inputs the `v4l2_timecode` struct is filled in in the dequeued `v4l2_buffer` struct.

### HDMI Input

The HDMI inputs supports all CEA-861 and DMT timings, both progressive and interlaced, for pixelclock frequencies between 25 and 600 MHz. The field mode for interlaced formats is always `V4L2_FIELD_ALTERNATE`. For HDMI the field order is always top field first, and when you start capturing an interlaced format you will receive the top field first.

The initially selected colorspace when you switch to the HDMI input or select an HDMI timing is based on the format resolution: for resolutions less than or equal to 720x576 the colorspace is set to SMPTE-170M, for others it is set to REC-709 (CEA-861 timings) or sRGB (VESA DMT timings).

The pixel aspect ratio will depend on the HDMI timing: for 720x480 is it set as for the NTSC TV standard, for 720x576 it is set as for the PAL TV standard, and for all others a 1:1 pixel aspect ratio is returned.

The video aspect ratio can be selected through the ‘DV Timings Aspect Ratio’ Vivid control. Choices are ‘Source Width x Height’ (just use the same ratio as the chosen format), ‘4x3’ or ‘16x9’, either of which can result in pillarboxed or letterboxed video.

For HDMI inputs it is possible to set the EDID. By default a simple EDID is provided. You can only set the EDID for HDMI inputs. Internally, however, the EDID is shared between all HDMI inputs.

No interpretation is done of the EDID data with the exception of the physical address. See the CEC section for more details.

There is a maximum of 15 HDMI inputs (if there are more, then they will be reduced to 15) since that’s the limitation of the EDID physical address.

### Video Output

The video output device can be configured by using the module options `num_outputs`, `output_types` and `ccs_out_mode` (see section 1 for more detailed information), but by default two outputs are configured: an S-Video and an HDMI input, one output for each output type. Those are described in more detail below.

Like with video capture the framerate is also exact in the long term.

### S-Video Output

This output supports audio outputs as well: “Line-Out 1” and “Line-Out 2”. The S-Video output supports all TV standards.

This output supports all combinations of the field setting.

The initially selected colorspace when you switch to the TV or S-Video input will be SMPTE-170M.

## **HDMI Output**

The HDMI output supports all CEA-861 and DMT timings, both progressive and interlaced, for pixelclock frequencies between 25 and 600 MHz. The field mode for interlaced formats is always `V4L2_FIELD_ALTERNATE`.

The initially selected colorspace when you switch to the HDMI output or select an HDMI timing is based on the format resolution: for resolutions less than or equal to 720x576 the colorspace is set to SMPTE-170M, for others it is set to REC-709 (CEA-861 timings) or sRGB (VESA DMT timings).

The pixel aspect ratio will depend on the HDMI timing: for 720x480 is it set as for the NTSC TV standard, for 720x576 it is set as for the PAL TV standard, and for all others a 1:1 pixel aspect ratio is returned.

An HDMI output has a valid EDID which can be obtained through `VIDIOC_G_EDID`.

There is a maximum of 15 HDMI outputs (if there are more, then they will be reduced to 15) since that's the limitation of the EDID physical address. See also the CEC section for more details.

## **VBI Capture**

There are three types of VBI capture devices: those that only support raw (undecoded) VBI, those that only support sliced (decoded) VBI and those that support both. This is determined by the `node_types` module option. In all cases the driver will generate valid VBI data: for 60 Hz standards it will generate Closed Caption and XDS data. The closed caption stream will alternate between "Hello world!" and "Closed captions test" every second. The XDS stream will give the current time once a minute. For 50 Hz standards it will generate the Wide Screen Signal which is based on the actual Video Aspect Ratio control setting and teletext pages 100-159, one page per frame.

The VBI device will only work for the S-Video and TV inputs, it will give back an error if the current input is a webcam or HDMI.

## **VBI Output**

There are three types of VBI output devices: those that only support raw (undecoded) VBI, those that only support sliced (decoded) VBI and those that support both. This is determined by the `node_types` module option.

The sliced VBI output supports the Wide Screen Signal and the teletext signal for 50 Hz standards and Closed Captioning + XDS for 60 Hz standards.

The VBI device will only work for the S-Video output, it will give back an error if the current output is HDMI.

### Radio Receiver

The radio receiver emulates an FM/AM/SW receiver. The FM band also supports RDS. The frequency ranges are:

- FM: 64 MHz - 108 MHz
- AM: 520 kHz - 1710 kHz
- SW: 2300 kHz - 26.1 MHz

Valid channels are emulated every 1 MHz for FM and every 100 kHz for AM and SW. The signal strength decreases the further the frequency is from the valid frequency until it becomes 0% at +/- 50 kHz (FM) or 5 kHz (AM/SW) from the ideal frequency. The initial frequency when the driver is loaded is set to 95 MHz.

The FM receiver supports RDS as well, both using 'Block I/O' and 'Controls' modes. In the 'Controls' mode the RDS information is stored in read-only controls. These controls are updated every time the frequency is changed, or when the tuner status is requested. The Block I/O method uses the read() interface to pass the RDS blocks on to the application for decoding.

The RDS signal is 'detected' for +/- 12.5 kHz around the channel frequency, and the further the frequency is away from the valid frequency the more RDS errors are randomly introduced into the block I/O stream, up to 50% of all blocks if you are +/- 12.5 kHz from the channel frequency. All four errors can occur in equal proportions: blocks marked 'CORRECTED', blocks marked 'ERROR', blocks marked 'INVALID' and dropped blocks.

The generated RDS stream contains all the standard fields contained in a 0B group, and also radio text and the current time.

The receiver supports HW frequency seek, either in Bounded mode, Wrap Around mode or both, which is configurable with the "Radio HW Seek Mode" control.

### Radio Transmitter

The radio transmitter emulates an FM/AM/SW transmitter. The FM band also supports RDS. The frequency ranges are:

- FM: 64 MHz - 108 MHz
- AM: 520 kHz - 1710 kHz
- SW: 2300 kHz - 26.1 MHz

The initial frequency when the driver is loaded is 95.5 MHz.

The FM transmitter supports RDS as well, both using 'Block I/O' and 'Controls' modes. In the 'Controls' mode the transmitted RDS information is configured using controls, and in 'Block I/O' mode the blocks are passed to the driver using write().

## Software Defined Radio Receiver

The SDR receiver has three frequency bands for the ADC tuner:

- 300 kHz
- 900 kHz - 2800 kHz
- 3200 kHz

The RF tuner supports 50 MHz - 2000 MHz.

The generated data contains the In-phase and Quadrature components of a 1 kHz tone that has an amplitude of  $\sqrt{2}$ .

## Metadata Capture

The Metadata capture generates UVC format metadata. The PTS and SCR are transmitted based on the values set in vivid controls.

The Metadata device will only work for the Webcam input, it will give back an error for all other inputs.

## Metadata Output

The Metadata output can be used to set brightness, contrast, saturation and hue.

The Metadata device will only work for the Webcam output, it will give back an error for all other outputs.

## Touch Capture

The Touch capture generates touch patterns simulating single tap, double tap, triple tap, move from left to right, zoom in, zoom out, palm press (simulating a large area being pressed on a touchpad), and simulating 16 simultaneous touch points.

## Controls

Different devices support different controls. The sections below will describe each control and which devices support them.

### User Controls - Test Controls

The Button, Boolean, Integer 32 Bits, Integer 64 Bits, Menu, String, Bitmask and Integer Menu are controls that represent all possible control types. The Menu control and the Integer Menu control both have ‘holes’ in their menu list, meaning that one or more menu items return EINVAL when VIDIOC\_QUERYMENU is called. Both menu controls also have a non-zero minimum control value. These features allow you to check if your application can handle such things correctly. These controls are supported for every device type.

### User Controls - Video Capture

The following controls are specific to video capture.

The Brightness, Contrast, Saturation and Hue controls actually work and are standard. There is one special feature with the Brightness control: each video input has its own brightness value, so changing input will restore the brightness for that input. In addition, each video input uses a different brightness range (minimum and maximum control values). Switching inputs will cause a control event to be sent with the V4L2\_EVENT\_CTRL\_CH\_RANGE flag set. This allows you to test controls that can change their range.

The ‘Gain, Automatic’ and Gain controls can be used to test volatile controls: if ‘Gain, Automatic’ is set, then the Gain control is volatile and changes constantly. If ‘Gain, Automatic’ is cleared, then the Gain control is a normal control.

The ‘Horizontal Flip’ and ‘Vertical Flip’ controls can be used to flip the image. These combine with the ‘Sensor Flipped Horizontally/Vertically’ Vivid controls.

The ‘Alpha Component’ control can be used to set the alpha component for formats containing an alpha channel.

### User Controls - Audio

The following controls are specific to video capture and output and radio receivers and transmitters.

The ‘Volume’ and ‘Mute’ audio controls are typical for such devices to control the volume and mute the audio. They don’t actually do anything in the vivid driver.

### Vivid Controls

These vivid custom controls control the image generation, error injection, etc.

## Test Pattern Controls

The Test Pattern Controls are all specific to video capture.

- **Test Pattern:**

selects which test pattern to use. Use the CSC Colorbar for testing colorspace conversions: the colors used in that test pattern map to valid colors in all colorspace. The colorspace conversion is disabled for the other test patterns.
- **OSD Text Mode:**

selects whether the text superimposed on the test pattern should be shown, and if so, whether only counters should be displayed or the full text.
- **Horizontal Movement:**

selects whether the test pattern should move to the left or right and at what speed.
- **Vertical Movement:**

does the same for the vertical direction.
- **Show Border:**

show a two-pixel wide border at the edge of the actual image, excluding letter or pillarboxing.
- **Show Square:**

show a square in the middle of the image. If the image is displayed with the correct pixel and image aspect ratio corrections, then the width and height of the square on the monitor should be the same.
- **Insert SAV Code in Image:**

adds a SAV (Start of Active Video) code to the image. This can be used to check if such codes in the image are inadvertently interpreted instead of being ignored.
- **Insert EAV Code in Image:**

does the same for the EAV (End of Active Video) code.

## Capture Feature Selection Controls

These controls are all specific to video capture.

- **Sensor Flipped Horizontally:**

the image is flipped horizontally and the V4L2\_IN\_ST\_HFLIP input status flag is set. This emulates the case where a sensor is for example mounted upside down.
- **Sensor Flipped Vertically:**

the image is flipped vertically and the `V4L2_IN_ST_VFLIP` input status flag is set. This emulates the case where a sensor is for example mounted upside down.

- **Standard Aspect Ratio:**

selects if the image aspect ratio as used for the TV or S-Video input should be 4x3, 16x9 or anamorphic widescreen. This may introduce letterboxing.

- **DV Timings Aspect Ratio:**

selects if the image aspect ratio as used for the HDMI input should be the same as the source width and height ratio, or if it should be 4x3 or 16x9. This may introduce letter or pillarboxing.

- **Timestamp Source:**

selects when the timestamp for each buffer is taken.

- **Colorspace:**

selects which colorspace should be used when generating the image. This only applies if the CSC Colorbar test pattern is selected, otherwise the test pattern will go through unconverted. This behavior is also what you want, since a 75% Colorbar should really have 75% signal intensity and should not be affected by colorspace conversions.

Changing the colorspace will result in the `V4L2_EVENT_SOURCE_CHANGE` to be sent since it emulates a detected colorspace change.

- **Transfer Function:**

selects which colorspace transfer function should be used when generating an image. This only applies if the CSC Colorbar test pattern is selected, otherwise the test pattern will go through unconverted. This behavior is also what you want, since a 75% Colorbar should really have 75% signal intensity and should not be affected by colorspace conversions.

Changing the transfer function will result in the `V4L2_EVENT_SOURCE_CHANGE` to be sent since it emulates a detected colorspace change.

- **Y' CbCr Encoding:**

selects which Y' CbCr encoding should be used when generating a Y' CbCr image. This only applies if the format is set to a Y' CbCr format as opposed to an RGB format.

Changing the Y' CbCr encoding will result in the `V4L2_EVENT_SOURCE_CHANGE` to be sent since it emulates a detected colorspace change.

- **Quantization:**

selects which quantization should be used for the RGB or Y' CbCr encoding when generating the test pattern.

Changing the quantization will result in the `V4L2_EVENT_SOURCE_CHANGE` to be sent since it emulates a detected colorspace change.

- **Limited RGB Range (16-235):**

selects if the RGB range of the HDMI source should be limited or full range. This combines with the Digital Video ‘Rx RGB Quantization Range’ control and can be used to test what happens if a source provides you with the wrong quantization range information. See the description of that control for more details.

- **Apply Alpha To Red Only:**

apply the alpha channel as set by the ‘Alpha Component’ user control to the red color of the test pattern only.

- **Enable Capture Cropping:**

enables crop support. This control is only present if the `ccs_cap_mode` module option is set to the default value of -1 and if the `no_error_inj` module option is set to 0 (the default).

- **Enable Capture Composing:**

enables composing support. This control is only present if the `ccs_cap_mode` module option is set to the default value of -1 and if the `no_error_inj` module option is set to 0 (the default).

- **Enable Capture Scaler:**

enables support for a scaler (maximum 4 times upscaling and down-scaling). This control is only present if the `ccs_cap_mode` module option is set to the default value of -1 and if the `no_error_inj` module option is set to 0 (the default).

- **Maximum EDID Blocks:**

determines how many EDID blocks the driver supports. Note that the vivid driver does not actually interpret new EDID data, it just stores it. It allows for up to 256 EDID blocks which is the maximum supported by the standard.

- **Fill Percentage of Frame:**

can be used to draw only the top X percent of the image. Since each frame has to be drawn by the driver, this demands a lot of the CPU. For large resolutions this becomes problematic. By drawing only part of the image this CPU load can be reduced.

### Output Feature Selection Controls

These controls are all specific to video output.

- **Enable Output Cropping:**  
enables crop support. This control is only present if the `ccs_out_mode` module option is set to the default value of -1 and if the `no_error_inj` module option is set to 0 (the default).
- **Enable Output Composing:**  
enables composing support. This control is only present if the `ccs_out_mode` module option is set to the default value of -1 and if the `no_error_inj` module option is set to 0 (the default).
- **Enable Output Scaler:**  
enables support for a scaler (maximum 4 times upscaling and down-scaling). This control is only present if the `ccs_out_mode` module option is set to the default value of -1 and if the `no_error_inj` module option is set to 0 (the default).

### Error Injection Controls

The following two controls are only valid for video and vbi capture.

- **Standard Signal Mode:**  
selects the behavior of `VIDIOC_QUERYSTD`: what should it return?  
Changing this control will result in the `V4L2_EVENT_SOURCE_CHANGE` to be sent since it emulates a changed input condition (e.g. a cable was plugged in or out).
- **Standard:**  
selects the standard that `VIDIOC_QUERYSTD` should return if the previous control is set to “Selected Standard” .  
Changing this control will result in the `V4L2_EVENT_SOURCE_CHANGE` to be sent since it emulates a changed input standard.

The following two controls are only valid for video capture.

- **DV Timings Signal Mode:**  
selects the behavior of `VIDIOC_QUERY_DV_TIMINGS`: what should it return?  
Changing this control will result in the `V4L2_EVENT_SOURCE_CHANGE` to be sent since it emulates a changed input condition (e.g. a cable was plugged in or out).
- **DV Timings:**  
selects the timings the `VIDIOC_QUERY_DV_TIMINGS` should return if the previous control is set to “Selected DV Timings” .

Changing this control will result in the `V4L2_EVENT_SOURCE_CHANGE` to be sent since it emulates changed input timings.

The following controls are only present if the `no_error_inj` module option is set to 0 (the default). These controls are valid for video and vbi capture and output streams and for the SDR capture device except for the Disconnect control which is valid for all devices.

- **Wrap Sequence Number:**

test what happens when you wrap the sequence number in struct `v4l2_buffer` around.
- **Wrap Timestamp:**

test what happens when you wrap the timestamp in struct `v4l2_buffer` around.
- **Percentage of Dropped Buffers:**

sets the percentage of buffers that are never returned by the driver (i.e., they are dropped).
- **Disconnect:**

emulates a USB disconnect. The device will act as if it has been disconnected. Only after all open filehandles to the device node have been closed will the device become 'connected' again.
- **Inject V4L2\_BUF\_FLAG\_ERROR:**

when pressed, the next frame returned by the driver will have the error flag set (i.e. the frame is marked corrupt).
- **Inject VIDIOC\_REQBUFS Error:**

when pressed, the next `REQBUFS` or `CREATE_BUFS` `ioctl` call will fail with an error. To be precise: the `videobuf2 queue_setup()` op will return `-EINVAL`.
- **Inject VIDIOC\_QBUF Error:**

when pressed, the next `VIDIOC_QBUF` or `VIDIOC_PREPARE_BUFFER` `ioctl` call will fail with an error. To be precise: the `videobuf2 buf_prepare()` op will return `-EINVAL`.
- **Inject VIDIOC\_STREAMON Error:**

when pressed, the next `VIDIOC_STREAMON` `ioctl` call will fail with an error. To be precise: the `videobuf2 start_streaming()` op will return `-EINVAL`.
- **Inject Fatal Streaming Error:**

when pressed, the streaming core will be marked as having suffered a fatal error, the only way to recover from that is to stop streaming. To be precise: the `videobuf2 vb2_queue_error()` function is called.

### VBI Raw Capture Controls

- Interlaced VBI Format:  
if set, then the raw VBI data will be interlaced instead of providing it grouped by field.

### Digital Video Controls

- Rx RGB Quantization Range:  
sets the RGB quantization detection of the HDMI input. This combines with the Vivid ‘Limited RGB Range (16-235)’ control and can be used to test what happens if a source provides you with the wrong quantization range information. This can be tested by selecting an HDMI input, setting this control to Full or Limited range and selecting the opposite in the ‘Limited RGB Range (16-235)’ control. The effect is easy to see if the ‘Gray Ramp’ test pattern is selected.
- Tx RGB Quantization Range:  
sets the RGB quantization detection of the HDMI output. It is currently not used for anything in vivid, but most HDMI transmitters would typically have this control.
- Transmit Mode:  
sets the transmit mode of the HDMI output to HDMI or DVI-D. This affects the reported colorspace since DVI\_D outputs will always use sRGB.
- Display Present:  
sets the presence of a “display” on the HDMI output. This affects the tx\_edid\_present, tx\_hotplug and tx\_rxsense controls.

### FM Radio Receiver Controls

- RDS Reception:  
set if the RDS receiver should be enabled.
- RDS Program Type:
- RDS PS Name:
- RDS Radio Text:
- RDS Traffic Announcement:
- RDS Traffic Program:
- RDS Music:  
these are all read-only controls. If RDS Rx I/O Mode is set to “Block I/O” , then they are inactive as well. If RDS Rx I/O Mode is set to “Controls” , then these controls report the received RDS data.

**Note:** The vivid implementation of this is pretty basic: they are only updated when you set a new frequency or when you get the tuner status (VIDIOC\_G\_TUNER).

---

- Radio HW Seek Mode:

can be one of “Bounded”, “Wrap Around” or “Both”. This determines if VIDIOC\_S\_HW\_FREQ\_SEEK will be bounded by the frequency range or wrap-around or if it is selectable by the user.

- Radio Programmable HW Seek:

if set, then the user can provide the lower and upper bound of the HW Seek. Otherwise the frequency range boundaries will be used.

- Generate RBDS Instead of RDS:

if set, then generate RBDS (the US variant of RDS) data instead of RDS (European-style RDS). This affects only the PICODE and PTY codes.

- RDS Rx I/O Mode:

this can be “Block I/O” where the RDS blocks have to be read() by the application, or “Controls” where the RDS data is provided by the RDS controls mentioned above.

## **FM Radio Modulator Controls**

- RDS Program ID:
- RDS Program Type:
- RDS PS Name:
- RDS Radio Text:
- RDS Stereo:
- RDS Artificial Head:
- RDS Compressed:
- RDS Dynamic PTY:
- RDS Traffic Announcement:
- RDS Traffic Program:
- RDS Music:

these are all controls that set the RDS data that is transmitted by the FM modulator.

- RDS Tx I/O Mode:

this can be “Block I/O” where the application has to use write() to pass the RDS blocks to the driver, or “Controls” where the RDS data is Provided by the RDS controls mentioned above.

### Metadata Capture Controls

- Generate PTS  
if set, then the generated metadata stream contains Presentation timestamp.
- Generate SCR  
if set, then the generated metadata stream contains Source Clock information.

### Video, VBI and RDS Looping

The vivid driver supports looping of video output to video input, VBI output to VBI input and RDS output to RDS input. For video/VBI looping this emulates as if a cable was hooked up between the output and input connector. So video and VBI looping is only supported between S-Video and HDMI inputs and outputs. VBI is only valid for S-Video as it makes no sense for HDMI.

Since radio is wireless this looping always happens if the radio receiver frequency is close to the radio transmitter frequency. In that case the radio transmitter will 'override' the emulated radio stations.

Looping is currently supported only between devices created by the same vivid driver instance.

### Video and Sliced VBI looping

The way to enable video/VBI looping is currently fairly crude. A 'Loop Video' control is available in the "Vivid" control class of the video capture and VBI capture devices. When checked the video looping will be enabled. Once enabled any video S-Video or HDMI input will show a static test pattern until the video output has started. At that time the video output will be looped to the video input provided that:

- the input type matches the output type. So the HDMI input cannot receive video from the S-Video output.
- the video resolution of the video input must match that of the video output. So it is not possible to loop a 50 Hz (720x576) S-Video output to a 60 Hz (720x480) S-Video input, or a 720p60 HDMI output to a 1080p30 input.
- the pixel formats must be identical on both sides. Otherwise the driver would have to do pixel format conversion as well, and that's taking things too far.
- the field settings must be identical on both sides. Same reason as above: requiring the driver to convert from one field format to another complicated matters too much. This also prohibits capturing with 'Field Top' or 'Field Bottom' when the output video is set to 'Field Alternate'. This combination, while legal, became too complicated to support. Both sides have to be 'Field Alternate' for this to work. Also note that for this specific case the sequence and field counting in struct `v4l2_buffer` on the capture side may not be 100% accurate.

- field settings `V4L2_FIELD_SEQ_TB/BT` are not supported. While it is possible to implement this, it would mean a lot of work to get this right. Since these field values are rarely used the decision was made not to implement this for now.
- on the input side the “Standard Signal Mode” for the S-Video input or the “DV Timings Signal Mode” for the HDMI input should be configured so that a valid signal is passed to the video input.

The framerates do not have to match, although this might change in the future.

By default you will see the OSD text superimposed on top of the looped video. This can be turned off by changing the “OSD Text Mode” control of the video capture device.

For VBI looping to work all of the above must be valid and in addition the vbi output must be configured for sliced VBI. The VBI capture side can be configured for either raw or sliced VBI. Note that at the moment only CC/XDS (60 Hz formats) and WSS (50 Hz formats) VBI data is looped. Teletext VBI data is not looped.

### **Radio & RDS Looping**

As mentioned in section 6 the radio receiver emulates stations are regular frequency intervals. Depending on the frequency of the radio receiver a signal strength value is calculated (this is returned by `VIDIOC_G_TUNER`). However, it will also look at the frequency set by the radio transmitter and if that results in a higher signal strength than the settings of the radio transmitter will be used as if it was a valid station. This also includes the RDS data (if any) that the transmitter ‘transmits’. This is received faithfully on the receiver side. Note that when the driver is loaded the frequencies of the radio receiver and transmitter are not identical, so initially no looping takes place.

### **Cropping, Composing, Scaling**

This driver supports cropping, composing and scaling in any combination. Normally which features are supported can be selected through the Vivid controls, but it is also possible to hardcode it when the module is loaded through the `ccs_cap_mode` and `ccs_out_mode` module options. See section 1 on the details of these module options.

This allows you to test your application for all these variations.

Note that the webcam input never supports cropping, composing or scaling. That only applies to the TV/S-Video/HDMI inputs and outputs. The reason is that webcams, including this virtual implementation, normally use `VIDIOC_ENUM_FRAMESIZES` to list a set of discrete framesizes that it supports. And that does not combine with cropping, composing or scaling. This is primarily a limitation of the V4L2 API which is carefully reproduced here.

The minimum and maximum resolutions that the scaler can achieve are 16x16 and  $(4096 * 4) \times (2160 \times 4)$ , but it can only scale up or down by a factor of 4 or less. So for a source resolution of 1280x720 the minimum the scaler can do is 320x180

and the maximum is 5120x2880. You can play around with this using the `qv4l2` test tool and you will see these dependencies.

This driver also supports larger ‘bytesperline’ settings, something that `VIDIOC_S_FMT` allows but that few drivers implement.

The scaler is a simple scaler that uses the Coarse Bresenham algorithm. It’s designed for speed and simplicity, not quality.

If the combination of crop, compose and scaling allows it, then it is possible to change crop and compose rectangles on the fly.

### Formats

The driver supports all the regular packed and planar 4:4:4, 4:2:2 and 4:2:0 YUYV formats, 8, 16, 24 and 32 RGB packed formats and various multiplanar formats.

The alpha component can be set through the ‘Alpha Component’ User control for those formats that support it. If the ‘Apply Alpha To Red Only’ control is set, then the alpha component is only used for the color red and set to 0 otherwise.

The driver has to be configured to support the multiplanar formats. By default the driver instances are single-planar. This can be changed by setting the multiplanar module option, see section 1 for more details on that option.

If the driver instance is using the multiplanar formats/API, then the first single planar format (YUYV) and the multiplanar NV16M and NV61M formats will have a plane that has a non-zero `data_offset` of 128 bytes. It is rare for `data_offset` to be non-zero, so this is a useful feature for testing applications.

Video output will also honor any `data_offset` that the application set.

### Capture Overlay

Note: capture overlay support is implemented primarily to test the existing V4L2 capture overlay API. In practice few if any GPUs support such overlays anymore, and neither are they generally needed anymore since modern hardware is so much more capable. By setting flag `0x10000` in the `node_types` module option the vivid driver will create a simple framebuffer device that can be used for testing this API. Whether this API should be used for new drivers is questionable.

This driver has support for a destructive capture overlay with bitmap clipping and list clipping (up to 16 rectangles) capabilities. Overlays are not supported for multiplanar formats. It also honors the `struct v4l2_window` field setting: if it is set to `FIELD_TOP` or `FIELD_BOTTOM` and the capture setting is `FIELD_ALTERNATE`, then only the top or bottom fields will be copied to the overlay.

The overlay only works if you are also capturing at that same time. This is a vivid limitation since it copies from a buffer to the overlay instead of filling the overlay directly. And if you are not capturing, then no buffers are available to fill.

In addition, the `pixelformat` of the capture format and that of the framebuffer must be the same for the overlay to work. Otherwise `VIDIOC_OVERLAY` will return an error.

In order to really see what it going on you will need to create two vivid instances: the first with a framebuffer enabled. You configure the capture overlay of the second instance to use the framebuffer of the first, then you start capturing in the second instance. For the first instance you setup the output overlay for the video output, turn on video looping and capture to see the blended framebuffer overlay that' s being written to by the second instance. This setup would require the following commands:

```
$ sudo modprobe vivid n_devs=2 node_types=0x10101,0x1
$ v4l2-ctl -d1 --find-fb
/dev/fb1 is the framebuffer associated with base address 0x12800000
$ sudo v4l2-ctl -d2 --set-fbuf fb=1
$ v4l2-ctl -d1 --set-fbuf fb=1
$ v4l2-ctl -d0 --set-fmt-video=pixelformat='AR15'
$ v4l2-ctl -d1 --set-fmt-video-out=pixelformat='AR15'
$ v4l2-ctl -d2 --set-fmt-video=pixelformat='AR15'
$ v4l2-ctl -d0 -i2
$ v4l2-ctl -d2 -i2
$ v4l2-ctl -d2 -c horizontal_movement=4
$ v4l2-ctl -d1 --overlay=1
$ v4l2-ctl -d1 -c loop_video=1
$ v4l2-ctl -d2 --stream-mmap --overlay=1
```

And from another console:

```
$ v4l2-ctl -d1 --stream-out-mmap
```

And yet another console:

```
$ qv4l2
```

and start streaming.

As you can see, this is not for the faint of heart...

## Output Overlay

Note: output overlays are primarily implemented in order to test the existing V4L2 output overlay API. Whether this API should be used for new drivers is questionable.

This driver has support for an output overlay and is capable of:

- bitmap clipping,
- list clipping (up to 16 rectangles)
- chromakey
- source chromakey
- global alpha
- local alpha
- local inverse alpha

Output overlays are not supported for multiplanar formats. In addition, the pixel format of the capture format and that of the framebuffer must be the same for the overlay to work. Otherwise `VIDIOC_OVERLAY` will return an error.

Output overlays only work if the driver has been configured to create a framebuffer by setting flag `0x10000` in the `node_types` module option. The created framebuffer has a size of `720x576` and supports `ARGB 1:5:5:5` and `RGB 5:6:5`.

In order to see the effects of the various clipping, chromakeying or alpha processing capabilities you need to turn on video looping and see the results on the capture side. The use of the clipping, chromakeying or alpha processing capabilities will slow down the video loop considerably as a lot of checks have to be done per pixel.

### CEC (Consumer Electronics Control)

If there are HDMI inputs then a CEC adapter will be created that has the same number of input ports. This is the equivalent of e.g. a TV that has that number of inputs. Each HDMI output will also create a CEC adapter that is hooked up to the corresponding input port, or (if there are more outputs than inputs) is not hooked up at all. In other words, this is the equivalent of hooking up each output device to an input port of the TV. Any remaining output devices remain unconnected.

The EDID that each output reads reports a unique CEC physical address that is based on the physical address of the EDID of the input. So if the EDID of the receiver has physical address `A.B.0.0`, then each output will see an EDID containing physical address `A.B.C.0` where `C` is 1 to the number of inputs. If there are more outputs than inputs then the remaining outputs have a CEC adapter that is disabled and reports an invalid physical address.

### Some Future Improvements

Just as a reminder and in no particular order:

- Add a virtual alsa driver to test audio
- Add virtual sub-devices and media controller support
- Some support for testing compressed video
- Add support to loop raw VBI output to raw VBI input
- Add support to loop teletext sliced VBI output to VBI input
- Fix sequence/field numbering when looping of video with alternate fields
- Add support for `V4L2_CID_BG_COLOR` for video outputs
- Add `ARGB888` overlay support: better testing of the alpha channel
- Improve pixel aspect support in the tpg code by passing a real `v4l2_fract`
- Use per-queue locks and/or per-device locks to improve throughput
- Add support to loop from a specific output to a specific input across vivid instances

- The SDR radio should use the same ‘frequencies’ for stations as the normal radio receiver, and give back noise if the frequency doesn’t match up with a station frequency
- Make a thread for the RDS generation, that would help in particular for the “Controls” RDS Rx I/O Mode as the read-only RDS controls could be updated in real-time.
- Changing the EDID should cause hotplug detect emulation to happen.

### 53.1.7 Digital TV driver-specific documentation

#### Avermedia DVB-T on BT878 Release Notes

February 14th 2006

---

**Note:** Several other Avermedia devices are supported. For a more broader and updated content about that, please check:

<https://linuxtv.org/wiki/index.php/AVerMedia>

---

#### The Avermedia DVB-T

The Avermedia DVB-T is a budget PCI DVB card. It has 3 inputs:

- RF Tuner Input
- Composite Video Input (RCA Jack)
- SVIDEO Input (Mini-DIN)

The RF Tuner Input is the input to the tuner module of the card. The Tuner is otherwise known as the “Frontend” . The Frontend of the Avermedia DVB-T is a Microtune 7202D. A timely post to the linux-dvb mailing list ascertained that the Microtune 7202D is supported by the sp887x driver which is found in the dvb-hw CVS module.

The DVB-T card is based around the BT878 chip which is a very common multimedia bridge and often found on Analogue TV cards. There is no on-board MPEG2 decoder, which means that all MPEG2 decoding must be done in software, or if you have one, on an MPEG2 hardware decoding card or chipset.

#### Getting the card going

At this stage, it has not been able to ascertain the functionality of the remaining device nodes in respect of the Avermedia DVBT. However, full functionality in respect of tuning, receiving and supplying the MPEG2 data stream is possible with the currently available versions of the driver. It may be possible that additional functionality is available from the card (i.e. viewing the additional analogue inputs that the card presents), but this has not been tested yet. If I get around to this, I’ll update the document with whatever I find.

To power up the card, load the following modules in the following order:

- modprobe bttv (normally loaded automatically)
- modprobe dvb-bt8xx (or place dvb-bt8xx in /etc/modules)

Insertion of these modules into the running kernel will activate the appropriate DVB device nodes. It is then possible to start accessing the card with utilities such as scan, tzap, dvbstream etc.

The frontend module sp887x.o, requires an external firmware. Please use the command “get\_dvb\_firmware sp887x” to download it. Then copy it to /usr/lib/hotplug/firmware or /lib/firmware/ (depending on configuration of firmware hotplug).

### Known Limitations

At present I can say with confidence that the frontend tunes via /dev/dvb/adapter{x}/frontend0 and supplies an MPEG2 stream via /dev/dvb/adapter{x}/dvr0. I have not tested the functionality of any other part of the card yet. I will do so over time and update this document.

There are some limitations in the i2c layer due to a returned error message inconsistency. Although this generates errors in dmesg and the system logs, it does not appear to affect the ability of the frontend to function correctly.

### Further Update

dvbstream and VideoLAN Client on windows works a treat with DVB, in fact this is currently serving as my main way of viewing DVB-T at the moment. Additionally, VLC is happily decoding HDTV signals, although the PC is dropping the odd frame here and there - I assume due to processing capability - as all the decoding is being done under windows in software.

Many thanks to Nigel Pearson for the updates to this document since the recent revision of the driver.

### How to get the bt8xx cards working

**Authors:** Richard Walker, Jamie Honan, Michael Hunold, Manu Abraham, Uwe Bugla, Michael Krufky

## General information

This class of cards has a bt878a as the PCI interface, and require the bttv driver for accessing the i2c bus and the gpio pins of the bt8xx chipset.

Please see BTTV cards list for a complete list of Cards based on the Conexant Bt8xx PCI bridge supported by the Linux Kernel.

In order to be able to compile the kernel, some config options should be enabled:

```
./scripts/config -e PCI
./scripts/config -e INPUT
./scripts/config -m I2C
./scripts/config -m MEDIA_SUPPORT
./scripts/config -e MEDIA_PCI_SUPPORT
./scripts/config -e MEDIA_ANALOG_TV_SUPPORT
./scripts/config -e MEDIA_DIGITAL_TV_SUPPORT
./scripts/config -e MEDIA_RADIO_SUPPORT
./scripts/config -e RC_CORE
./scripts/config -m VIDEO_BT848
./scripts/config -m DVB_BT8XX
```

If you want to automatically support all possible variants of the Bt8xx cards, you should also do:

```
./scripts/config -e MEDIA_SUBDRV_AUTOSELECT
```

---

**Note:** Please use the following options with care as deselection of drivers which are in fact necessary may result in DVB devices that cannot be tuned due to lack of driver support.

---

If your goal is to just support an specific board, you may, instead, disable MEDIA\_SUBDRV\_AUTOSELECT and manually select the frontend drivers required by your board. With that, you can save some RAM.

You can do that by calling make xconfig/qconfig/menuconfig and look at the options on those menu options (only enabled if Autoselect ancillary drivers is disabled:

- 1) Device drivers => Multimedia support => Customize TV tuners
- 2) Device drivers => Multimedia support => Customize DVB frontends

Then, on each of the above menu, please select your card-specific frontend and tuner modules.

### Loading Modules

Regular case: If the bttv driver detects a bt8xx-based DVB card, all frontend and backend modules will be loaded automatically.

Exceptions are:

- Old TV cards without EEPROMs, sharing a common PCI subsystem ID;
- Old TwinHan DST cards or clones with or without CA slot and not containing an Eeprom.

In the following cases overriding the PCI type detection for bttv and for dvb-bt8xx drivers by passing modprobe parameters may be necessary.

### Running TwinHan and Clones

As shown at BTTV cards list, TwinHan and clones use card=113 modprobe parameter. So, in order to properly detect it for devices without EEPROM, you should use:

```
$ modprobe bttv card=113
$ modprobe dst
```

Useful parameters for verbosity level and debugging the dst module:

```
verbose=0:      messages are disabled
  1:            only error messages are displayed
  2:            notifications are displayed
  3:            other useful messages are displayed
  4:            debug setting
dst_addons=0:   card is a free to air (FTA) card only
  0x20:        card has a conditional access slot for scrambled channels
dst_algo=0:     (default) Software tuning algorithm
  1:            Hardware tuning algorithm
```

The autodetected values are determined by the cards' "response string" .

In your logs see f. ex.: dst\_get\_device\_id: Recognize [DSTMCI].

For bug reports please send in a complete log with verbose=4 activated. Please also see Digital TV Conditional Access Interface.

### Running multiple cards

See BTTV cards list for a complete list of Card ID. Some examples:

Brand name	ID
Pinnacle PCTV Sat	94
Nebula Electronics Digi TV	104
pcHDTV HD-2000 TV	112
Twinhan DST and clones	113
Avermedia AverTV DVB-T 77:	123
Avermedia AverTV DVB-T 761	124
DViCO FusionHDTV DVB-T Lite	128
DViCO FusionHDTV 5 Lite	135

---

**Note:** When you have multiple cards, the order of the card ID should match the order where they're detected by the system. Please notice that removing/inserting other PCI cards may change the detection order.

---

Example:

```
$ modprobe bttv card=113 card=135
```

In case of further problems please subscribe and send questions to the mailing list: [linux-media@vger.kernel.org](mailto:linux-media@vger.kernel.org).

### Probing the cards with broken PCI subsystem ID

There are some TwinHan cards whose EEPROM has become corrupted for some reason. The cards do not have a correct PCI subsystem ID. Still, it is possible to force probing the cards with:

```
$ echo 109e 0878 $subvendor $subdevice > \
    /sys/bus/pci/drivers/bt878/new_id
```

The two numbers there are:

```
109e: PCI_VENDOR_ID_BROOKTREE
0878: PCI_DEVICE_ID_BROOKTREE_878
```

### Firmware files for lmedm04 cards

To extract firmware for the DM04/QQBOX you need to copy the following file(s) to this directory.

### For DM04+/QQBOX LME2510C (Sharp 7395 Tuner)

The Sharp 7395 driver can be found in windows/system32/drivers

US2A0D.sys (dated 17 Mar 2009)

and run:

```
scripts/get_dvb_firmware lme2510c_s7395
```

will produce dvb-usb-lme2510c-s7395.fw

An alternative but older firmware can be found on the driver disk DVB-S\_EN\_3.5A in BDADriver/driver

LMEBDA\_DVBS7395C.sys (dated 18 Jan 2008)

and run:

```
./get_dvb_firmware lme2510c_s7395_old
```

will produce dvb-usb-lme2510c-s7395.fw

The LG firmware can be found on the driver disk DM04+\_5.1A[LG] in BDADriver/driver

### For DM04 LME2510 (LG Tuner)

LMEBDA\_DVBS.sys (dated 13 Nov 2007)

and run:

```
./get_dvb_firmware lme2510_lg
```

will produce dvb-usb-lme2510-lg.fw

Other LG firmware can be extracted manually from US280D.sys only found in windows/system32/drivers

```
dd if=US280D.sys ibs=1 skip=42360 count=3924 of=dvb-usb-lme2510-lg.fw
```

### For DM04 LME2510C (LG Tuner)

```
dd if=US280D.sys ibs=1 skip=35200 count=3850 of=dvb-usb-lme2510c-lg.fw
```

The Sharp 0194 tuner driver can be found in windows/system32/drivers

US290D.sys (dated 09 Apr 2009)

### For LME2510

```
dd if=US290D.sys ibs=1 skip=36856 count=3976 of=dvb-usb-lme2510-s0194.fw
```

### For LME2510C

```
dd if=US290D.sys ibs=1 skip=33152 count=3697 of=dvb-usb-lme2510c-s0194.fw
```

The m88rs2000 tuner driver can be found in windows/system32/drivers

US2B0D.sys (dated 29 Jun 2010)

```
dd if=US2B0D.sys ibs=1 skip=34432 count=3871 of=dvb-usb-lme2510c-rs2000.fw
```

We need to modify id of rs2000 firmware or it will warm boot id 3344:1120.

```
echo -ne \\xF0\\x22 | dd conv=notrunc bs=1 count=2 seek=266 of=dvb-usb-  
→lme2510c-rs2000.fw
```

Copy the firmware file(s) to /lib/firmware

### Opera firmware

Author: Marco Gittler <[g.marco@freenet.de](mailto:g.marco@freenet.de)>

To extract the firmware for the Opera DVB-S1 USB-Box you need to copy the files:  
2830SCap2.sys 2830SLoad2.sys

from the windriver disk into this directory.

Then run:

```
scripts/get_dvb_firmware opera1
```

and after that you have 2 files:

```
dvb-usb-opera-01.fw dvb-usb-opera1-fpga-01.fw
```

in here.

Copy them into /lib/firmware/ .

After that the driver can load the firmware (if you have enabled firmware loading in kernel config and have hotplug running).

### How to set up the Technisat/B2C2 Flexcop devices

---

**Note:** This documentation is outdated.

---

Author: Uwe Bugla <[uwe.bugla@gmx.de](mailto:uwe.bugla@gmx.de)> August 2009

#### Find out what device you have

Important Notice: The driver does NOT support Technisat USB 2 devices!

First start your linux box with a shipped kernel:

```
lspci -vvv for a PCI device (lsusb -vvv for an USB device) will show you
↳for example:
02:0b.0 Network controller: Techsan Electronics Co Ltd B2C2 FlexCopII DVB
↳chip /
Technisat SkyStar2 DVB card (rev 02)

dmesg |grep frontend may show you for example:
DVB: registering frontend 0 (Conexant CX24123/CX24109)...
```

#### Kernel compilation:

If the Flexcop / Technisat is the only DVB / TV / Radio device in your box get rid of unnecessary modules and check this one:

Multimedia support => Customise analog and hybrid tuner modules to build

In this directory uncheck every driver which is activated there (except Simple tuner support for ATSC 3rd generation only -> see case 9 please).

Then please activate:

- Main module part:

Multimedia support => DVB/ATSC adapters => Technisat/B2C2 FlexcopII(b) and FlexCopIII adapters

1) => Technisat/B2C2 Air/Sky/Cable2PC PCI (PCI card) or

2) => Technisat/B2C2 Air/Sky/Cable2PC USB (USB 1.1 adapter) and for troubleshooting purposes:

3) => Enable debug for the B2C2 FlexCop drivers

- Frontend / Tuner / Demodulator module part:

**Multimedia support => DVB/ATSC adapters =>** Customise the frontend modules to build Customise DVB frontends =>

- SkyStar DVB-S Revision 2.3:

1) => Zarlink VP310/MT312/ZL10313 based

2) => Generic I2C PLL based tuners

- SkyStar DVB-S Revision 2.6:
  - 1) => ST STV0299 based
  - 2) => Generic I2C PLL based tuners
- SkyStar DVB-S Revision 2.7:
  - 1) => Samsung S5H1420 based
  - 2) => Integrant ITD1000 Zero IF tuner for DVB-S/DSS
  - 3) => ISL6421 SEC controller
- SkyStar DVB-S Revision 2.8:
  - 1) => Conexant CX24123 based
  - 2) => Conexant CX24113/CX24128 tuner for DVB-S/DSS
  - 3) => ISL6421 SEC controller
- AirStar DVB-T card:
  - 1) => Zarlink MT352 based
  - 2) => Generic I2C PLL based tuners
- CableStar DVB-C card:
  - 1) => ST STV0297 based
  - 2) => Generic I2C PLL based tuners
- AirStar ATSC card 1st generation:
  - 1) => Broadcom BCM3510
- AirStar ATSC card 2nd generation:
  - 1) => NxtWave Communications NXT2002/NXT2004 based
  - 2) => Generic I2C PLL based tuners
- AirStar ATSC card 3rd generation:
  - 1) => LG Electronics LGDT3302/LGDT3303 based
  - 2) Multimedia support => Customise analog and hybrid tuner modules to build => Simple tuner support

## TechnoTrend/Hauppage DEC USB Driver

### Driver Status

Supported:

- DEC2000-t
- DEC2450-t
- DEC3000-s
- Video Streaming

- Audio Streaming
- Section Filters
- Channel Zapping
- Hotplug firmware loader

To Do:

- Tuner status information
- DVB network interface
- Streaming video PC->DEC
- Conax support for 2450-t

### Getting the Firmware

To download the firmware, use the following commands:

```
scripts/get_dvb_firmware dec2000t
scripts/get_dvb_firmware dec2540t
scripts/get_dvb_firmware dec3000s
```

### Hotplug Firmware Loading

Since 2.6 kernels, the firmware is loaded at the point that the driver module is loaded.

Copy the three files downloaded above into the `/usr/lib/hotplug/firmware` or `/lib/firmware` directory (depending on configuration of firmware hotplug).

### Zoran 364xx based USB webcam module

site: <http://royale.zerezo.com/zr364xx/>

mail: [royale@zerezo.com](mailto:royale@zerezo.com)

### Introduction

This brings support under Linux for the Aiptek PocketDV 3300 and similar devices in webcam mode. If you just want to get on your PC the pictures and movies on the camera, you should use the usb-storage module instead.

The driver works with several other cameras in webcam mode (see the list below).

Possible chipsets are : ZR36430 (ZR36430BGC) and maybe ZR36431, ZR36440, ZR36442...

You can try the experience changing the vendor/product ID values (look at the source code).

You can get these values by looking at `/var/log/messages` when you plug your camera, or by typing : `cat /sys/kernel/debug/usb/devices`.

## Install

In order to use this driver, you must compile it with your kernel, with the following config options:

```
./scripts/config -e USB
./scripts/config -m MEDIA_SUPPORT
./scripts/config -e MEDIA_USB_SUPPORT
./scripts/config -e MEDIA_CAMERA_SUPPORT
./scripts/config -m USB_ZR364XX
```

## Usage

`modprobe zr364xx debug=X mode=Y`

- `debug` : set to 1 to enable verbose debug messages
- `mode` : 0 = 320x240, 1 = 160x120, 2 = 640x480

You can then use the camera with V4L2 compatible applications, for example Ekiga.

To capture a single image, try this: `dd if=/dev/video0 of=test.jpg bs=1M count=1`

## links

<http://mxhaard.free.fr/> (support for many others cams including some Aiptek PocketDV) <http://www.harmwal.nl/pccam880/> (this project also supports cameras based on this chipset)

## Supported devices

	Vendor	Product	Distributor	Model
0x08ca	0x0109	Aiptek		PocketDV 3300
0x08ca	0x0109	Maxell		Maxcam PRO DV3
0x041e	0x4024	Creative		PC-CAM 880
0x0d64	0x0108	Aiptek		Fidelity 3200
0x0d64	0x0108	Praktica		DCZ 1.3 S
0x0d64	0x0108	Genius		Digital Camera (?)
0x0d64	0x0108	DXG Technology		Fashion Cam
0x0546	0x3187	Polaroid		iON 230
0x0d64	0x3108	Praktica		Exakta DC 2200
0x0d64	0x3108	Genius		G-Shot D211
0x0595	0x4343	Concord		Eye-Q Duo 1300
0x0595	0x4343	Concord		Eye-Q Duo 2000

Continued on next page

Table 21 – continued from previous page

	Vendor	Product	Distributor	Model
0x0595	0x4343	Fujifilm		EX-10
0x0595	0x4343	Ricoh		RDC-6000
0x0595	0x4343	Digitrex		DSC 1300
0x0595	0x4343	Firstline		FDC 2000
0x0bb0	0x500d	Concord		EyeQ Go Wireless
0x0feb	0x2004	CRS Electronic		3.3 Digital Camera
0x0feb	0x2004	Packard Bell		DSC-300
0x055f	0xb500	Mustek		MDC 3000
0x08ca	0x2062	Aiptek		PocketDV 5700
0x052b	0x1a18	Chiphead		Megapix V12
0x04c8	0x0729	Konica		Revio 2
0x04f2	0xa208	Creative		PC-CAM 850
0x0784	0x0040	Traveler		Slimline X5
0x06d6	0x0034	Trust		Powerc@m 750
0x0a17	0x0062	Pentax		Optio 50L
0x06d6	0x003b	Trust		Powerc@m 970Z
0x0a17	0x004e	Pentax		Optio 50
0x041e	0x405d	Creative		DiVi CAM 516
0x08ca	0x2102	Aiptek		DV T300
0x06d6	0x003d	Trust		Powerc@m 910Z

### 53.1.8 CEC driver-specific documentation

#### Pulse-Eight CEC Adapter driver

The pulse8-cec driver implements the following module option:

#### `persistent_config`

By default this is off, but when set to 1 the driver will store the current settings to the device' s internal eeprom and restore it the next time the device is connected to the USB port.

**Copyright** © 1999-2020 : LinuxTV Developers

This documentation is free software; you can redistribute it and/or modify [it](#) under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) [any](#) later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

For more details see the file COPYING in the source distribution of Linux.

## **MEMORY MANAGEMENT**

Linux memory management subsystem is responsible, as the name implies, for managing the memory in the system. This includes implementation of virtual memory and demand paging, memory allocation both for kernel internal structures and user space programs, mapping of files into processes address space and many other cool things.

Linux memory management is a complex system with many configurable settings. Most of these settings are available via `/proc` filesystem and can be queried and adjusted using `sysctl`. These APIs are described in `Documentation/admin-guide/sysctl/vm.rst` and in `man 5 proc`.

Linux memory management has its own jargon and if you are not yet familiar with it, consider reading `Documentation/admin-guide/mm/concepts.rst`.

Here we document in detail how to interact with various mechanisms in the Linux memory management.

### **54.1 Concepts overview**

The memory management in Linux is a complex system that evolved over the years and included more and more functionality to support a variety of systems from MMU-less microcontrollers to supercomputers. The memory management for systems without an MMU is called `nommu` and it definitely deserves a dedicated document, which hopefully will be eventually written. Yet, although some of the concepts are the same, here we assume that an MMU is available and a CPU can translate a virtual address to a physical address.

- Virtual Memory Primer
- Huge Pages
- Zones
- Nodes
- Page cache
- Anonymous Memory
- Reclaim

- Compaction
- OOM killer

### 54.1.1 Virtual Memory Primer

The physical memory in a computer system is a limited resource and even for systems that support memory hotplug there is a hard limit on the amount of memory that can be installed. The physical memory is not necessarily contiguous; it might be accessible as a set of distinct address ranges. Besides, different CPU architectures, and even different implementations of the same architecture have different views of how these address ranges are defined.

All this makes dealing directly with physical memory quite complex and to avoid this complexity a concept of virtual memory was developed.

The virtual memory abstracts the details of physical memory from the application software, allows to keep only needed information in the physical memory (demand paging) and provides a mechanism for the protection and controlled sharing of data between processes.

With virtual memory, each and every memory access uses a virtual address. When the CPU decodes the an instruction that reads (or writes) from (or to) the system memory, it translates the virtual address encoded in that instruction to a physical address that the memory controller can understand.

The physical system memory is divided into page frames, or pages. The size of each page is architecture specific. Some architectures allow selection of the page size from several supported values; this selection is performed at the kernel build time by setting an appropriate kernel configuration option.

Each physical memory page can be mapped as one or more virtual pages. These mappings are described by page tables that allow translation from a virtual address used by programs to the physical memory address. The page tables are organized hierarchically.

The tables at the lowest level of the hierarchy contain physical addresses of actual pages used by the software. The tables at higher levels contain physical addresses of the pages belonging to the lower levels. The pointer to the top level page table resides in a register. When the CPU performs the address translation, it uses this register to access the top level page table. The high bits of the virtual address are used to index an entry in the top level page table. That entry is then used to access the next level in the hierarchy with the next bits of the virtual address as the index to that level page table. The lowest bits in the virtual address define the offset inside the actual page.

### 54.1.2 Huge Pages

The address translation requires several memory accesses and memory accesses are slow relatively to CPU speed. To avoid spending precious processor cycles on the address translation, CPUs maintain a cache of such translations called Translation Lookaside Buffer (or TLB). Usually TLB is pretty scarce resource and applications with large memory working set will experience performance hit because of TLB misses.

Many modern CPU architectures allow mapping of the memory pages directly by the higher levels in the page table. For instance, on x86, it is possible to map 2M and even 1G pages using entries in the second and the third level page tables. In Linux such pages are called huge. Usage of huge pages significantly reduces pressure on TLB, improves TLB hit-rate and thus improves overall system performance.

There are two mechanisms in Linux that enable mapping of the physical memory with the huge pages. The first one is HugeTLB filesystem, or hugetlbfs. It is a pseudo filesystem that uses RAM as its backing store. For the files created in this filesystem the data resides in the memory and mapped using huge pages. The hugetlbfs is described at [Documentation/admin-guide/mm/hugetlbpage.rst](#).

Another, more recent, mechanism that enables use of the huge pages is called Transparent HugePages, or THP. Unlike the hugetlbfs that requires users and/or system administrators to configure what parts of the system memory should and can be mapped by the huge pages, THP manages such mappings transparently to the user and hence the name. See [Documentation/admin-guide/mm/transhuge.rst](#) for more details about THP.

### 54.1.3 Zones

Often hardware poses restrictions on how different physical memory ranges can be accessed. In some cases, devices cannot perform DMA to all the addressable memory. In other cases, the size of the physical memory exceeds the maximal addressable size of virtual memory and special actions are required to access portions of the memory. Linux groups memory pages into zones according to their possible usage. For example, `ZONE_DMA` will contain memory that can be used by devices for DMA, `ZONE_HIGHMEM` will contain memory that is not permanently mapped into kernel's address space and `ZONE_NORMAL` will contain normally addressed pages.

The actual layout of the memory zones is hardware dependent as not all architectures define all zones, and requirements for DMA are different for different platforms.

### 54.1.4 Nodes

Many multi-processor machines are NUMA - Non-Uniform Memory Access - systems. In such systems the memory is arranged into banks that have different access latency depending on the “distance” from the processor. Each bank is referred to as a node and for each node Linux constructs an independent memory management subsystem. A node has its own set of zones, lists of free and used pages and various statistics counters. You can find more details about NUMA in Documentation/vm/numa.rst and in Documentation/admin-guide/mm/numa\_memory\_policy.rst.

### 54.1.5 Page cache

The physical memory is volatile and the common case for getting data into the memory is to read it from files. Whenever a file is read, the data is put into the page cache to avoid expensive disk access on the subsequent reads. Similarly, when one writes to a file, the data is placed in the page cache and eventually gets into the backing storage device. The written pages are marked as dirty and when Linux decides to reuse them for other purposes, it makes sure to synchronize the file contents on the device with the updated data.

### 54.1.6 Anonymous Memory

The anonymous memory or anonymous mappings represent memory that is not backed by a filesystem. Such mappings are implicitly created for program’s stack and heap or by explicit calls to `mmap(2)` system call. Usually, the anonymous mappings only define virtual memory areas that the program is allowed to access. The read accesses will result in creation of a page table entry that references a special physical page filled with zeroes. When the program performs a write, a regular physical page will be allocated to hold the written data. The page will be marked dirty and if the kernel decides to repurpose it, the dirty page will be swapped out.

### 54.1.7 Reclaim

Throughout the system lifetime, a physical page can be used for storing different types of data. It can be kernel internal data structures, DMA’ able buffers for device drivers use, data read from a filesystem, memory allocated by user space processes etc.

Depending on the page usage it is treated differently by the Linux memory management. The pages that can be freed at any time, either because they cache the data available elsewhere, for instance, on a hard disk, or because they can be swapped out, again, to the hard disk, are called reclaimable. The most notable categories of the reclaimable pages are page cache and anonymous memory.

In most cases, the pages holding internal kernel data and used as DMA buffers cannot be repurposed, and they remain pinned until freed by their user. Such pages are called unreclaimable. However, in certain circumstances, even pages occupied with kernel data structures can be reclaimed. For instance, in-memory

caches of filesystem metadata can be re-read from the storage device and therefore it is possible to discard them from the main memory when system is under memory pressure.

The process of freeing the reclaimable physical memory pages and repurposing them is called (surprise!) reclaim. Linux can reclaim pages either asynchronously or synchronously, depending on the state of the system. When the system is not loaded, most of the memory is free and allocation requests will be satisfied immediately from the free pages supply. As the load increases, the amount of the free pages goes down and when it reaches a certain threshold (high watermark), an allocation request will awaken the kswapd daemon. It will asynchronously scan memory pages and either just free them if the data they contain is available elsewhere, or evict to the backing storage device (remember those dirty pages?). As memory usage increases even more and reaches another threshold - min watermark - an allocation will trigger direct reclaim. In this case allocation is stalled until enough memory pages are reclaimed to satisfy the request.

### **54.1.8 Compaction**

As the system runs, tasks allocate and free the memory and it becomes fragmented. Although with virtual memory it is possible to present scattered physical pages as virtually contiguous range, sometimes it is necessary to allocate large physically contiguous memory areas. Such need may arise, for instance, when a device driver requires a large buffer for DMA, or when THP allocates a huge page. Memory compaction addresses the fragmentation issue. This mechanism moves occupied pages from the lower part of a memory zone to free pages in the upper part of the zone. When a compaction scan is finished free pages are grouped together at the beginning of the zone and allocations of large physically contiguous areas become possible.

Like reclaim, the compaction may happen asynchronously in the kcompactd daemon or synchronously as a result of a memory allocation request.

### **54.1.9 OOM killer**

It is possible that on a loaded machine memory will be exhausted and the kernel will be unable to reclaim enough memory to continue to operate. In order to save the rest of the system, it invokes the OOM killer.

The OOM killer selects a task to sacrifice for the sake of the overall system health. The selected task is killed in a hope that after it exits enough memory will be freed to continue normal operation.

## 54.2 CMA Debugfs Interface

The CMA debugfs interface is useful to retrieve basic information out of the different CMA areas and to test allocation/release in each of the areas.

Each CMA zone represents a directory under `<debugfs>/cma/`, indexed by the kernel's CMA index. So the first CMA zone would be:

```
<debugfs>/cma/cma-0
```

The structure of the files created under that directory is as follows:

- [RO] `base_pfn`: The base PFN (Page Frame Number) of the zone.
- [RO] `count`: Amount of memory in the CMA area.
- [RO] `order_per_bit`: Order of pages represented by one bit.
- [RO] `bitmap`: The bitmap of page states in the zone.
- [WO] `alloc`: Allocate N pages from that CMA area. For example:

```
echo 5 > <debugfs>/cma/cma-2/alloc
```

would try to allocate 5 pages from the `cma-2` area.

- [WO] `free`: Free N pages from that CMA area, similar to the above.

## 54.3 HugeTLB Pages

### 54.3.1 Overview

The intent of this file is to give a brief summary of `hugetlbpage` support in the Linux kernel. This support is built on top of multiple page size support that is provided by most modern architectures. For example, x86 CPUs normally support 4K and 2M (1G if architecturally supported) page sizes, ia64 architecture supports multiple page sizes 4K, 8K, 64K, 256K, 1M, 4M, 16M, 256M and ppc64 supports 4K and 16M. A TLB is a cache of virtual-to-physical translations. Typically this is a very scarce resource on processor. Operating systems try to make best use of limited number of TLB resources. This optimization is more critical now as bigger and bigger physical memories (several GBs) are more readily available.

Users can use the huge page support in Linux kernel by either using the `mmap` system call or standard SYSV shared memory system calls (`shmget`, `shmat`).

First the Linux kernel needs to be built with the `CONFIG_HUGETLBFS` (present under "File systems") and `CONFIG_HUGETLB_PAGE` (selected automatically when `CONFIG_HUGETLBFS` is selected) configuration options.

The `/proc/meminfo` file provides information about the total number of persistent `hugetlb` pages in the kernel's huge page pool. It also displays default huge page size and information about the number of free, reserved and surplus huge pages in the pool of huge pages of default size. The huge page size is needed for generating the proper alignment and size of the arguments to system calls that map huge page regions.

The output of `cat /proc/meminfo` will include lines like:

```
HugePages_Total: uuu
HugePages_Free:  vvv
HugePages_Rsvd:  www
HugePages_Surp:  xxx
Hugepagesize:    yyy kB
Hugetlb:         zzz kB
```

where:

**HugePages\_Total** is the size of the pool of huge pages.

**HugePages\_Free** is the number of huge pages in the pool that are not yet allocated.

**HugePages\_Rsvd** is short for “reserved,” and is the number of huge pages for which a commitment to allocate from the pool has been made, but no allocation has yet been made. Reserved huge pages guarantee that an application will be able to allocate a huge page from the pool of huge pages at fault time.

**HugePages\_Surp** is short for “surplus,” and is the number of huge pages in the pool above the value in `/proc/sys/vm/nr_hugepages`. The maximum number of surplus huge pages is controlled by `/proc/sys/vm/nr_overcommit_hugepages`.

**Hugepagesize** is the default hugepage size (in Kb).

**Hugetlb** is the total amount of memory (in kB), consumed by huge pages of all sizes. If huge pages of different sizes are in use, this number will exceed `HugePages_Total * Hugepagesize`. To get more detailed information, please, refer to `/sys/kernel/mm/hugepages` (described below).

`/proc/filesystems` should also show a filesystem of type “`hugetlbfs`” configured in the kernel.

`/proc/sys/vm/nr_hugepages` indicates the current number of “persistent” huge pages in the kernel’s huge page pool. “Persistent” huge pages will be returned to the huge page pool when freed by a task. A user with root privileges can dynamically allocate more or free some persistent huge pages by increasing or decreasing the value of `nr_hugepages`.

Pages that are used as huge pages are reserved inside the kernel and cannot be used for other purposes. Huge pages cannot be swapped out under memory pressure.

Once a number of huge pages have been pre-allocated to the kernel huge page pool, a user with appropriate privilege can use either the `mmap` system call or shared memory system calls to use the huge pages. See the discussion of Using Huge Pages, below.

The administrator can allocate persistent huge pages on the kernel boot command line by specifying the “`hugepages=N`” parameter, where ‘N’ = the number of huge pages requested. This is the most reliable method of allocating huge pages as memory has not yet become fragmented.

Some platforms support multiple huge page sizes. To allocate huge pages of a specific size, one must precede the huge pages boot command parameters with

a huge page size selection parameter “`hugepagesz=<size>`” . `<size>` must be specified in bytes with optional scale suffix [kKmMgG]. The default huge page size may be selected with the “`default_hugepagesz=<size>`” boot parameter.

Hugetlb boot command line parameter semantics `hugepagesz` - Specify a huge page size. Used in conjunction with `hugepages`

parameter to preallocate a number of huge pages of the specified size. Hence, `hugepagesz` and `hugepages` are typically specified in pairs such as:

```
hugepagesz=2M hugepages=512
```

`hugepagesz` can only be specified once on the command line for a specific huge page size. Valid huge page sizes are architecture dependent.

**hugepages - Specify the number of huge pages to preallocate. This typically follows a valid `hugepagesz` or `default_hugepagesz` parameter.** However, if `hugepages` is the first or only hugetlb command line parameter it implicitly specifies the number of huge pages of default size to allocate. If the number of huge pages of default size is implicitly specified, it can not be overwritten by a `hugepagesz,hugepages` parameter pair for the default size. For example, on an architecture with 2M default huge page size:

```
hugepages=256 hugepagesz=2M hugepages=512
```

will result in 256 2M huge pages being allocated and a warning message indicating that the `hugepages=512` parameter is ignored. If a `hugepages` parameter is preceded by an invalid `hugepagesz` parameter, it will be ignored.

**default\_hugepagesz - Specify the default huge page size. This parameter can only be specified once on the command line.** `default_hugepagesz` can optionally be followed by the `hugepages` parameter to preallocate a specific number of huge pages of default size. The number of default sized huge pages to preallocate can also be implicitly specified as mentioned in the `hugepages` section above. Therefore, on an architecture with 2M default huge page size:

```
hugepages=256      default_hugepagesz=2M      hugepages=256
hugepages=256 default_hugepagesz=2M
```

will all result in 256 2M huge pages being allocated. Valid default huge page size is architecture dependent.

When multiple huge page sizes are supported, `/proc/sys/vm/nr_hugepages` indicates the current number of pre-allocated huge pages of the default size. Thus, one can use the following command to dynamically allocate/deallocate default sized persistent huge pages:

```
echo 20 > /proc/sys/vm/nr_hugepages
```

This command will try to adjust the number of default sized huge pages in the huge page pool to 20, allocating or freeing huge pages, as required.

On a NUMA platform, the kernel will attempt to distribute the huge page pool over all the set of allowed nodes specified by the NUMA memory policy of the task that modifies `nr_hugepages`. The default for the allowed nodes-when the task has default memory policy-is all on-line nodes with memory. Allowed nodes with

insufficient available, contiguous memory for a huge page will be silently skipped when allocating persistent huge pages. See the discussion below of the interaction of task memory policy, cpusets and per node attributes with the allocation and freeing of persistent huge pages.

The success or failure of huge page allocation depends on the amount of physically contiguous memory that is present in system at the time of the allocation attempt. If the kernel is unable to allocate huge pages from some nodes in a NUMA system, it will attempt to make up the difference by allocating extra pages on other nodes with sufficient available contiguous memory, if any.

System administrators may want to put this command in one of the local rc init files. This will enable the kernel to allocate huge pages early in the boot process when the possibility of getting physical contiguous pages is still very high. Administrators can verify the number of huge pages actually allocated by checking the `sysctl` or `meminfo`. To check the per node distribution of huge pages in a NUMA system, use:

```
cat /sys/devices/system/node/node*/meminfo | fgrep Huge
```

`/proc/sys/vm/nr_overcommit_hugepages` specifies how large the pool of huge pages can grow, if more huge pages than `/proc/sys/vm/nr_hugepages` are requested by applications. Writing any non-zero value into this file indicates that the `hugetlb` subsystem is allowed to try to obtain that number of “surplus” huge pages from the kernel’s normal page pool, when the persistent huge page pool is exhausted. As these surplus huge pages become unused, they are freed back to the kernel’s normal page pool.

When increasing the huge page pool size via `nr_hugepages`, any existing surplus pages will first be promoted to persistent huge pages. Then, additional huge pages will be allocated, if necessary and if possible, to fulfill the new persistent huge page pool size.

The administrator may shrink the pool of persistent huge pages for the default huge page size by setting the `nr_hugepages` `sysctl` to a smaller value. The kernel will attempt to balance the freeing of huge pages across all nodes in the memory policy of the task modifying `nr_hugepages`. Any free huge pages on the selected nodes will be freed back to the kernel’s normal page pool.

Caveat: Shrinking the persistent huge page pool via `nr_hugepages` such that it becomes less than the number of huge pages in use will convert the balance of the in-use huge pages to surplus huge pages. This will occur even if the number of surplus pages would exceed the `overcommit` value. As long as this condition holds—that is, until `nr_hugepages+nr_overcommit_hugepages` is increased sufficiently, or the surplus huge pages go out of use and are freed—no more surplus huge pages will be allowed to be allocated.

With support for multiple huge page pools at run-time available, much of the huge page userspace interface in `/proc/sys/vm` has been duplicated in `sysfs`. The `/proc` interfaces discussed above have been retained for backwards compatibility. The root huge page control directory in `sysfs` is:

```
/sys/kernel/mm/hugepages
```

For each huge page size supported by the running kernel, a subdirectory will exist,

of the form:

```
hugepages- $\{size\}$ kB
```

Inside each of these directories, the same set of files will exist:

```
nr_hugepages
nr_hugepages_mempolicy
nr_overcommit_hugepages
free_hugepages
resv_hugepages
surplus_hugepages
```

which function as described above for the default huge page-sized case.

### 54.3.2 Interaction of Task Memory Policy with Huge Page Allocation/Freeing

Whether huge pages are allocated and freed via the `/proc` interface or the `/sys/fs` interface using the `nr_hugepages_mempolicy` attribute, the NUMA nodes from which huge pages are allocated or freed are controlled by the NUMA memory policy of the task that modifies the `nr_hugepages_mempolicy` `sysctl` or attribute. When the `nr_hugepages` attribute is used, `mempolicy` is ignored.

The recommended method to allocate or free huge pages to/from the kernel huge page pool, using the `nr_hugepages` example above, is:

```
numactl --interleave <node-list> echo 20 \  
>/proc/sys/vm/nr_hugepages_mempolicy
```

or, more succinctly:

```
numactl -m <node-list> echo 20 >/proc/sys/vm/nr_hugepages_mempolicy
```

This will allocate or free `abs(20 - nr_hugepages)` to or from the nodes specified in `<node-list>`, depending on whether number of persistent huge pages is initially less than or greater than 20, respectively. No huge pages will be allocated nor freed on any node not included in the specified `<node-list>`.

When adjusting the persistent hugepage count via `nr_hugepages_mempolicy`, any memory policy mode—`bind`, preferred, local or `interleave`—may be used. The resulting effect on persistent huge page allocation is as follows:

1. Regardless of `mempolicy` mode [see `Documentation/admin-guide/mm/numa_memory_policy.rst`], persistent huge pages will be distributed across the node or nodes specified in the `mempolicy` as if “interleave” had been specified. However, if a node in the policy does not contain sufficient contiguous memory for a huge page, the allocation will not “fallback” to the nearest neighbor node with sufficient contiguous memory. To do this would cause undesirable imbalance in the distribution of the huge page pool, or possibly, allocation of persistent huge pages on nodes not allowed by the task’s memory policy.
2. One or more nodes may be specified with the `bind` or `interleave` policy. If more than one node is specified with the preferred policy, only the lowest

numeric id will be used. Local policy will select the node where the task is running at the time the `nodes_allowed` mask is constructed. For local policy to be deterministic, the task must be bound to a `cpu` or `cpus` in a single node. Otherwise, the task could be migrated to some other node at any time after launch and the resulting node will be indeterminate. Thus, local policy is not very useful for this purpose. Any of the other mempolicy modes may be used to specify a single node.

3. The nodes allowed mask will be derived from any non-default task mempolicy, whether this policy was set explicitly by the task itself or one of its ancestors, such as `numactl`. This means that if the task is invoked from a shell with non-default policy, that policy will be used. One can specify a node list of “all” with `numactl -interleave` or `-mbind [-m]` to achieve interleaving over all nodes in the system or `cpuset`.
4. Any task mempolicy specified—e.g., using `numactl`—will be constrained by the resource limits of any `cpuset` in which the task runs. Thus, there will be no way for a task with non-default policy running in a `cpuset` with a subset of the system nodes to allocate huge pages outside the `cpuset` without first moving to a `cpuset` that contains all of the desired nodes.
5. Boot-time huge page allocation attempts to distribute the requested number of huge pages over all on-lines nodes with memory.

### 54.3.3 Per Node Hugepages Attributes

A subset of the contents of the root huge page control directory in `sysfs`, described above, will be replicated under each the system device of each NUMA node with memory in:

```
/sys/devices/system/node/node[0-9]*/hugepages/
```

Under this directory, the subdirectory for each supported huge page size contains the following attribute files:

```
nr_hugepages
free_hugepages
surplus_hugepages
```

The `free_` and `surplus_` attribute files are read-only. They return the number of free and surplus [overcommitted] huge pages, respectively, on the parent node.

The `nr_hugepages` attribute returns the total number of huge pages on the specified node. When this attribute is written, the number of persistent huge pages on the parent node will be adjusted to the specified value, if sufficient resources exist, regardless of the task’s mempolicy or `cpuset` constraints.

Note that the number of overcommit and reserve pages remain global quantities, as we don’t know until fault time, when the faulting task’s mempolicy is applied, from which node the huge page allocation will be attempted.

### 54.3.4 Using Huge Pages

If the user applications are going to request huge pages using `mmap` system call, then it is required that system administrator mount a file system of type `hugetlbfs`:

```
mount -t hugetlbfs \
-o uid=<value>,gid=<value>,mode=<value>,pagesize=<value>,size=<value>
→,\
  min_size=<value>,nr_inodes=<value> none /mnt/huge
```

This command mounts a (pseudo) filesystem of type `hugetlbfs` on the directory `/mnt/huge`. Any file created on `/mnt/huge` uses huge pages.

The `uid` and `gid` options sets the owner and group of the root of the file system. By default the `uid` and `gid` of the current process are taken.

The `mode` option sets the mode of root of file system to value `& 01777`. This value is given in octal. By default the value `0755` is picked.

If the platform supports multiple huge page sizes, the `pagesize` option can be used to specify the huge page size and associated pool. `pagesize` is specified in bytes. If `pagesize` is not specified the platform's default huge page size and associated pool will be used.

The `size` option sets the maximum value of memory (huge pages) allowed for that filesystem (`/mnt/huge`). The `size` option can be specified in bytes, or as a percentage of the specified huge page pool (`nr_hugepages`). The size is rounded down to `HPAGE_SIZE` boundary.

The `min_size` option sets the minimum value of memory (huge pages) allowed for the filesystem. `min_size` can be specified in the same way as `size`, either bytes or a percentage of the huge page pool. At mount time, the number of huge pages specified by `min_size` are reserved for use by the filesystem. If there are not enough free huge pages available, the mount will fail. As huge pages are allocated to the filesystem and freed, the reserve count is adjusted so that the sum of allocated and reserved huge pages is always at least `min_size`.

The option `nr_inodes` sets the maximum number of inodes that `/mnt/huge` can use.

If the `size`, `min_size` or `nr_inodes` option is not provided on command line then no limits are set.

For `pagesize`, `size`, `min_size` and `nr_inodes` options, you can use `[G|g]/[M|m]/[K|k]` to represent giga/mega/kilo. For example, `size=2K` has the same meaning as `size=2048`.

While read system calls are supported on files that reside on `hugetlb` file systems, write system calls are not.

Regular `chown`, `chgrp`, and `chmod` commands (with right permissions) could be used to change the file attributes on `hugetlbfs`.

Also, it is important to note that no such mount command is required if applications are going to use only `shmat/shmget` system calls or `mmap` with `MAP_HUGETLB`. For an example of how to use `mmap` with `MAP_HUGETLB` see `map_hugetlb` below.

Users who wish to use hugetlb memory via shared memory segment should be members of a supplementary group and system admin needs to configure that gid into `/proc/sys/vm/hugetlb_shm_group`. It is possible for same or different applications to use any combination of `mmaps` and `shm*` calls, though the mount of filesystem will be required for using `mmap` calls without `MAP_HUGETLB`.

Syscalls that operate on memory backed by hugetlb pages only have their lengths aligned to the native page size of the processor; they will normally fail with `errno` set to `EINVAL` or exclude hugetlb pages that extend beyond the length if not hugepage aligned. For example, `munmap(2)` will fail if memory is backed by a hugetlb page and the length is smaller than the hugepage size.

### 54.3.5 Examples

**map\_hugetlb** see `tools/testing/selftests/vm/map_hugetlb.c`

**hugepage-shm** see `tools/testing/selftests/vm/hugepage-shm.c`

**hugepage-mmap** see `tools/testing/selftests/vm/hugepage-mmap.c`

The `libhugetlbf`s library provides a wide range of userspace tools to help with huge page usability, environment setup, and control.

## 54.4 Idle Page Tracking

### 54.4.1 Motivation

The idle page tracking feature allows to track which memory pages are being accessed by a workload and which are idle. This information can be useful for estimating the workload's working set size, which, in turn, can be taken into account when configuring the workload parameters, setting memory cgroup limits, or deciding where to place the workload within a compute cluster.

It is enabled by `CONFIG_IDLE_PAGE_TRACKING=y`.

### 54.4.2 User API

The idle page tracking API is located at `/sys/kernel/mm/page_idle`. Currently, it consists of the only read-write file, `/sys/kernel/mm/page_idle/bitmap`.

The file implements a bitmap where each bit corresponds to a memory page. The bitmap is represented by an array of 8-byte integers, and the page at PFN `#i` is mapped to bit `#i%64` of array element `#i/64`, byte order is native. When a bit is set, the corresponding page is idle.

A page is considered idle if it has not been accessed since it was marked idle (for more details on what “accessed” actually means see the Implementation Details section). To mark a page idle one has to set the bit corresponding to the page by writing to the file. A value written to the file is OR-ed with the current bitmap value.

Only accesses to user memory pages are tracked. These are pages mapped to a process address space, page cache and buffer pages, swap cache pages. For other page types (e.g. SLAB pages) an attempt to mark a page idle is silently ignored, and hence such pages are never reported idle.

For huge pages the idle flag is set only on the head page, so one has to read `/proc/kpageflags` in order to correctly count idle huge pages.

Reading from or writing to `/sys/kernel/mm/page_idle/bitmap` will return `-EINVAL` if you are not starting the read/write on an 8-byte boundary, or if the size of the read/write is not a multiple of 8 bytes. Writing to this file beyond `max PFN` will return `-ENXIO`.

That said, in order to estimate the amount of pages that are not used by a workload one should:

1. Mark all the workload's pages as idle by setting corresponding bits in `/sys/kernel/mm/page_idle/bitmap`. The pages can be found by reading `/proc/pid/pagemap` if the workload is represented by a process, or by filtering out alien pages using `/proc/kpagecgroup` in case the workload is placed in a memory cgroup.
2. Wait until the workload accesses its working set.
3. Read `/sys/kernel/mm/page_idle/bitmap` and count the number of bits set. If one wants to ignore certain types of pages, e.g. mlocked pages since they are not reclaimable, he or she can filter them out using `/proc/kpageflags`.

The `page-types` tool in the `tools/vm` directory can be used to assist in this. If the tool is run initially with the appropriate option, it will mark all the queried pages as idle. Subsequent runs of the tool can then show which pages have their idle flag cleared in the interim.

See `Documentation/admin-guide/mm/pagemap.rst` for more information about `/proc/pid/pagemap`, `/proc/kpageflags`, and `/proc/kpagecgroup`.

### 54.4.3 Implementation Details

The kernel internally keeps track of accesses to user memory pages in order to reclaim unreferenced pages first on memory shortage conditions. A page is considered referenced if it has been recently accessed via a process address space, in which case one or more PTEs it is mapped to will have the Accessed bit set, or marked accessed explicitly by the kernel (see `mark_page_accessed()`). The latter happens when:

- a userspace process reads or writes a page using a system call (e.g. `read(2)` or `write(2)`)
- a page that is used for storing filesystem buffers is read or written, because a process needs filesystem metadata stored in it (e.g. lists a directory tree)
- a page is accessed by a device driver using `get_user_pages()`

When a dirty page is written to swap or disk as a result of memory reclaim or exceeding the dirty memory limit, it is not marked referenced.

The idle memory tracking feature adds a new page flag, the Idle flag. This flag is set manually, by writing to `/sys/kernel/mm/page_idle/bitmap` (see the User API section), and cleared automatically whenever a page is referenced as defined above.

When a page is marked idle, the Accessed bit must be cleared in all PTEs it is mapped to, otherwise we will not be able to detect accesses to the page coming from a process address space. To avoid interference with the reclaimer, which, as noted above, uses the Accessed bit to promote actively referenced pages, one more page flag is introduced, the Young flag. When the PTE Accessed bit is cleared as a result of setting or updating a page's Idle flag, the Young flag is set on the page. The reclaimer treats the Young flag as an extra PTE Accessed bit and therefore will consider such a page as referenced.

Since the idle memory tracking feature is based on the memory reclaimer logic, it only works with pages that are on an LRU list, other pages are silently ignored. That means it will ignore a user memory page if it is isolated, but since there are usually not many of them, it should not affect the overall result noticeably. In order not to stall scanning of the idle page bitmap, locked pages may be skipped too.

## **54.5 Kernel Samepage Merging**

### **54.5.1 Overview**

KSM is a memory-saving de-duplication feature, enabled by `CONFIG_KSM=y`, added to the Linux kernel in 2.6.32. See `mm/ksm.c` for its implementation, and <http://lwn.net/Articles/306704/> and <http://lwn.net/Articles/330589/>

KSM was originally developed for use with KVM (where it was known as Kernel Shared Memory), to fit more virtual machines into physical memory, by sharing the data common between them. But it can be useful to any application which generates many instances of the same data.

The KSM daemon `ksmd` periodically scans those areas of user memory which have been registered with it, looking for pages of identical content which can be replaced by a single write-protected page (which is automatically copied if a process later wants to update its content). The amount of pages that KSM daemon scans in a single pass and the time between the passes are configured using `sysfs` interface

KSM only merges anonymous (private) pages, never pagecache (file) pages. KSM's merged pages were originally locked into kernel memory, but can now be swapped out just like other user pages (but sharing is broken when they are swapped back in: `ksmd` must rediscover their identity and merge again).

### 54.5.2 Controlling KSM with madvise

KSM only operates on those areas of address space which an application has advised to be likely candidates for merging, by using the `madvise(2)` system call:

```
int madvise(addr, length, MADV_MERGEABLE)
```

The app may call

```
int madvise(addr, length, MADV_UNMERGEABLE)
```

to cancel that advice and restore unshared pages: whereupon KSM unmerges whatever it merged in that range. Note: this unmerging call may suddenly require more memory than is available - possibly failing with `EAGAIN`, but more probably arousing the Out-Of-Memory killer.

If KSM is not configured into the running kernel, `madvise MADV_MERGEABLE` and `MADV_UNMERGEABLE` simply fail with `EINVAL`. If the running kernel was built with `CONFIG_KSM=y`, those calls will normally succeed: even if the the KSM daemon is not currently running, `MADV_MERGEABLE` still registers the range for whenever the KSM daemon is started; even if the range cannot contain any pages which KSM could actually merge; even if `MADV_UNMERGEABLE` is applied to a range which was never `MADV_MERGEABLE`.

If a region of memory must be split into at least one new `MADV_MERGEABLE` or `MADV_UNMERGEABLE` region, the `madvise` may return `ENOMEM` if the process will exceed `vm.max_map_count` (see [Documentation/admin-guide/sysctl/vm.rst](#)).

Like other `madvise` calls, they are intended for use on mapped areas of the user address space: they will report `ENOMEM` if the specified range includes unmapped gaps (though working on the intervening mapped areas), and might fail with `EAGAIN` if not enough memory for internal structures.

Applications should be considerate in their use of `MADV_MERGEABLE`, restricting its use to areas likely to benefit. KSM's scans may use a lot of processing power: some installations will disable KSM for that reason.

### 54.5.3 KSM daemon sysfs interface

The KSM daemon is controlled by sysfs files in `/sys/kernel/mm/ksm/`, readable by all but writable only by root:

**pages\_to\_scan** how many pages to scan before `ksmd` goes to sleep e.g. `echo 100 > /sys/kernel/mm/ksm/pages_to_scan`.

Default: 100 (chosen for demonstration purposes)

**sleep\_millisecs** how many milliseconds `ksmd` should sleep before next scan e.g. `echo 20 > /sys/kernel/mm/ksm/sleep_millisecs`

Default: 20 (chosen for demonstration purposes)

**merge\_across\_nodes** specifies if pages from different NUMA nodes can be merged. When set to 0, `ksm` merges only pages which physically reside in

the memory area of same NUMA node. That brings lower latency to access of shared pages. Systems with more nodes, at significant NUMA distances, are likely to benefit from the lower latency of setting 0. Smaller systems, which need to minimize memory usage, are likely to benefit from the greater sharing of setting 1 (default). You may wish to compare how your system performs under each setting, before deciding on which to use. `merge_across_nodes` setting can be changed only when there are no ksm shared pages in the system: set `run` to 2 to unmerge pages first, then to 1 after changing `merge_across_nodes`, to remerge according to the new setting.

Default: 1 (merging across nodes as in earlier releases)

### **run**

- set to 0 to stop `ksmd` from running but keep merged pages,
- set to 1 to run `ksmd` e.g. `echo 1 > /sys/kernel/mm/ksm/run`,
- set to 2 to stop `ksmd` and unmerge all pages currently merged, but leave mergeable areas registered for next run.

Default: 0 (must be changed to 1 to activate KSM, except if `CONFIG_SYSFS` is disabled)

**use\_zero\_pages** specifies whether empty pages (i.e. allocated pages that only contain zeroes) should be treated specially. When set to 1, empty pages are merged with the kernel zero page(s) instead of with each other as it would happen normally. This can improve the performance on architectures with coloured zero pages, depending on the workload. Care should be taken when enabling this setting, as it can potentially degrade the performance of KSM for some workloads, for example if the checksums of pages candidate for merging match the checksum of an empty page. This setting can be changed at any time, it is only effective for pages merged after the change.

Default: 0 (normal KSM behaviour as in earlier releases)

**max\_page\_sharing** Maximum sharing allowed for each KSM page. This enforces a deduplication limit to avoid high latency for virtual memory operations that involve traversal of the virtual mappings that share the KSM page. The minimum value is 2 as a newly created KSM page will have at least two sharers. The higher this value the faster KSM will merge the memory and the higher the deduplication factor will be, but the slower the worst case virtual mappings traversal could be for any given KSM page. Slowing down this traversal means there will be higher latency for certain virtual memory operations happening during swapping, compaction, NUMA balancing and page migration, in turn decreasing responsiveness for the caller of those virtual memory operations. The scheduler latency of other tasks not involved with the VM operations doing the virtual mappings traversal is not affected by this parameter as these traversals are always schedule friendly themselves.

**stable\_node\_chains\_prune\_millisecs** specifies how frequently KSM checks the metadata of the pages that hit the deduplication limit for stale information. Smaller millisecs values will free up the KSM metadata with lower latency, but they will make `ksmd` use more CPU during the scan. It's a noop if not a single KSM page hit the `max_page_sharing` yet.

The effectiveness of KSM and MADV\_MERGEABLE is shown in `/sys/kernel/mm/ksm/`:

**pages\_shared** how many shared pages are being used

**pages\_sharing** how many more sites are sharing them i.e. how much saved

**pages\_unshared** how many pages unique but repeatedly checked for merging

**pages\_volatile** how many pages changing too fast to be placed in a tree

**full\_scans** how many times all mergeable areas have been scanned

**stable\_node\_chains** the number of KSM pages that hit the `max_page_sharing` limit

**stable\_node\_dups** number of duplicated KSM pages

A high ratio of `pages_sharing` to `pages_shared` indicates good sharing, but a high ratio of `pages_unshared` to `pages_sharing` indicates wasted effort. `pages_volatile` embraces several different kinds of activity, but a high proportion there would also indicate poor use of `madvise MADV_MERGEABLE`.

The maximum possible `pages_sharing/pages_shared` ratio is limited by the `max_page_sharing` tunable. To increase the ratio `max_page_sharing` must be increased accordingly.

- Izik Eidus, Hugh Dickins, 17 Nov 2009

## 54.6 Memory Hotplug

**Created** Jul 28 2007

**Updated** Add some details about locking internals: Aug 20 2018

This document is about memory hotplug including how-to-use and current status. Because Memory Hotplug is still under development, contents of this text will be changed often.

- Introduction
  - Purpose of memory hotplug
  - Phases of memory hotplug
  - Unit of Memory online/offline operation
- Kernel Configuration
- sysfs files for memory hotplug
- Physical memory hot-add phase
  - Hardware(Firmware) Support
  - Notify memory hot-add event by hand
- Logical Memory hot-add phase

- State of memory
- How to online memory
- Logical memory remove
  - Memory offline and ZONE\_MOVABLE
  - How to offline memory
- Physical memory remove
- Locking Internals
- Future Work

---

**Note:**

- (1) x86\_64' s has special implementation for memory hotplug. This text does not describe it.
  - (2) This text assumes that sysfs is mounted at /sys.
- 

### **54.6.1 Introduction**

#### **Purpose of memory hotplug**

Memory Hotplug allows users to increase/decrease the amount of memory. Generally, there are two purposes.

- (A) For changing the amount of memory. This is to allow a feature like capacity on demand.
- (B) For installing/removing DIMMs or NUMA-nodes physically. This is to exchange DIMMs/NUMA-nodes, reduce power consumption, etc.

(A) is required by highly virtualized environments and (B) is required by hardware which supports memory power management.

Linux memory hotplug is designed for both purpose.

#### **Phases of memory hotplug**

There are 2 phases in Memory Hotplug:

- 1) Physical Memory Hotplug phase
- 2) Logical Memory Hotplug phase.

The First phase is to communicate hardware/firmware and make/erase environment for hotplugged memory. Basically, this phase is necessary for the purpose (B), but this is good phase for communication between highly virtualized environments too.

When memory is hotplugged, the kernel recognizes new memory, makes new memory management tables, and makes sysfs files for new memory' s operation.

If firmware supports notification of connection of new memory to OS, this phase is triggered automatically. ACPI can notify this event. If not, “probe” operation by system administration is used instead. (see Physical memory hot-add phase).

Logical Memory Hotplug phase is to change memory state into available/unavailable for users. Amount of memory from user’s view is changed by this phase. The kernel makes all memory in it as free pages when a memory range is available.

In this document, this phase is described as online/offline.

Logical Memory Hotplug phase is triggered by write of sysfs file by system administrator. For the hot-add case, it must be executed after Physical Hotplug phase by hand. (However, if you writes udev’s hotplug scripts for memory hotplug, these phases can be execute in seamless way.)

### Unit of Memory online/offline operation

Memory hotplug uses SPARSEMEM memory model which allows memory to be divided into chunks of the same size. These chunks are called “sections” . The size of a memory section is architecture dependent. For example, power uses 16MiB, ia64 uses 1GiB.

Memory sections are combined into chunks referred to as “memory blocks” . The size of a memory block is architecture dependent and represents the logical unit upon which memory online/offline operations are to be performed. The default size of a memory block is the same as memory section size unless an architecture specifies otherwise. (see sysfs files for memory hotplug.)

To determine the size (in bytes) of a memory block please read this file:

```
/sys/devices/system/memory/block_size_bytes
```

### 54.6.2 Kernel Configuration

To use memory hotplug feature, kernel must be compiled with following config options.

- **For all memory hotplug:**
  - Memory model -> Sparse Memory (CONFIG\_SPARSEMEM)
  - Allow for memory hot-add (CONFIG\_MEMORY\_HOTPLUG)
- **To enable memory removal, the following are also necessary:**
  - Allow for memory hot remove (CONFIG\_MEMORY\_HOTREMOVE)
  - Page Migration (CONFIG\_MIGRATION)
- **For ACPI memory hotplug, the following are also necessary:**
  - Memory hotplug (under ACPI Support menu) (CONFIG\_ACPI\_HOTPLUG\_MEMORY)
  - This option can be kernel module.

- As a related configuration, if your box has a feature of NUMA-node hotplug via ACPI, then this option is necessary too.

- ACPI0004,PNP0A05 and PNP0A06 Container Driver (under ACPI Support menu) (CONFIG\_ACPI\_CONTAINER).

This option can be kernel module too.

### 54.6.3 sysfs files for memory hotplug

All memory blocks have their device information in sysfs. Each memory block is described under `/sys/devices/system/memory` as:

```
/sys/devices/system/memory/memoryXXX
```

where XXX is the memory block id.

For the memory block covered by the sysfs directory. It is expected that all memory sections in this range are present and no memory holes exist in the range. Currently there is no way to determine if there is a memory hole, but the existence of one should not affect the hotplug capabilities of the memory block.

For example, assume 1GiB memory block size. A device for a memory starting at 0x100000000 is `/sys/device/system/memory/memory4`:

```
(0x100000000 / 1Gib = 4)
```

This device covers address range [0x100000000 ..0x140000000)

Under each memory block, you can see 5 files:

- `/sys/devices/system/memory/memoryXXX/phys_index`
- `/sys/devices/system/memory/memoryXXX/phys_device`
- `/sys/devices/system/memory/memoryXXX/state`
- `/sys/devices/system/memory/memoryXXX/removable`
- `/sys/devices/system/memory/memoryXXX/valid_zones`

phys_index	read-only and contains memory block id, same as XXX.
state	read-write <ul style="list-style-type: none"> <li>• at read: contains online/offline state of memory.</li> <li>• at write: user can specify “online_kernel” , “online_movable” , “online” , “offline” command which will be performed on all sections in the block.</li> </ul>
phys_device	read-only: designed to show the name of physical memory device. This is not well implemented now.
removable	read-only: contains an integer value indicating whether the memory block is removable or not removable. A value of 1 indicates that the memory block is removable and a value of 0 indicates that it is not removable. A memory block is removable only if every section in the block is removable.
valid_zones	read-only: designed to show which zones this memory block can be on-lined to. The first column shows it`s default zone. “memory6/valid_zones: Normal Movable” shows this memoryblock can be onlined to ZONE_NORMAL by default and to ZONE_MOVABLE by online_movable. “memory7/valid_zones: Movable Normal” shows this memoryblock can be onlined to ZONE_MOVABLE by default and to ZONE_NORMAL by online_kernel.

---

**Note:** These directories/files appear after physical memory hotplug phase.

---

If CONFIG\_NUMA is enabled the memoryXXX/ directories can also be accessed via symbolic links located in the /sys/devices/system/node/node\* directories.

For example:

```
/sys/devices/system/node/node0/memory9 -> ../../memory/memory9
```

A backlink will also be created:

```
/sys/devices/system/memory/memory9/node0 -> ../../node/node0
```

## 54.6.4 Physical memory hot-add phase

### Hardware(Firmware) Support

On x86\_64/ia64 platform, memory hotplug by ACPI is supported.

In general, the firmware (ACPI) which supports memory hotplug defines memory class object of \_HID “PNP0C80” . When a notify is asserted to PNP0C80, Linux’ s ACPI handler does hot-add memory to the system and calls a hotplug udev script. This will be done automatically.

But scripts for memory hotplug are not contained in generic udev package(now). You may have to write it by yourself or online/offline memory by hand. Please see How to online memory and How to offline memory.

If firmware supports NUMA-node hotplug, and defines an object \_HID “ACPI0004” , “PNP0A05” , or “PNP0A06” , notification is asserted to it, and ACPI handler calls hotplug code for all of objects which are defined in it. If memory device is found, memory hotplug code will be called.

### Notify memory hot-add event by hand

On some architectures, the firmware may not notify the kernel of a memory hotplug event. Therefore, the memory “probe” interface is supported to explicitly notify the kernel. This interface depends on CONFIG\_ARCH\_MEMORY\_PROBE and can be configured on powerpc, sh, and x86 if hotplug is supported, although for x86 this should be handled by ACPI notification.

Probe interface is located at:

```
/sys/devices/system/memory/probe
```

You can tell the physical address of new memory to the kernel by:

```
% echo start_address_of_new_memory > /sys/devices/system/memory/probe
```

Then, [start\_address\_of\_new\_memory, start\_address\_of\_new\_memory + memory\_block\_size] memory range is hot-added. In this case, hotplug script is not called (in current implementation). You’ ll have to online memory by yourself. Please see How to online memory.

## 54.6.5 Logical Memory hot-add phase

### State of memory

To see (online/offline) state of a memory block, read ‘state’ file:

```
% cat /sys/device/system/memory/memoryXXX/state
```

- If the memory block is online, you’ ll read “online” .
- If the memory block is offline, you’ ll read “offline” .

### How to online memory

When the memory is hot-added, the kernel decides whether or not to “online” it according to the policy which can be read from “auto\_online\_blocks” file:

```
% cat /sys/devices/system/memory/auto_online_blocks
```

The default depends on the CONFIG\_MEMORY\_HOTPLUG\_DEFAULT\_ONLINE kernel config option. If it is disabled the default is “offline” which means the newly added memory is not in a ready-to-use state and you have to “online” the newly added memory blocks manually. Automatic onlining can be requested by writing “online” to “auto\_online\_blocks” file:

```
% echo online > /sys/devices/system/memory/auto_online_blocks
```

This sets a global policy and impacts all memory blocks that will subsequently be hotplugged. Currently offline blocks keep their state. It is possible, under certain circumstances, that some memory blocks will be added but will fail to online. User space tools can check their “state” files (/sys/devices/system/memory/memoryXXX/state) and try to online them manually.

If the automatic onlining wasn't requested, failed, or some memory block was offlined it is possible to change the individual block's state by writing to the “state” file:

```
% echo online > /sys/devices/system/memory/memoryXXX/state
```

This onlining will not change the ZONE type of the target memory block, If the memory block doesn't belong to any zone an appropriate kernel zone (usually ZONE\_NORMAL) will be used unless movable\_node kernel command line option is specified when ZONE\_MOVABLE will be used.

You can explicitly request to associate it with ZONE\_MOVABLE by:

```
% echo online_movable > /sys/devices/system/memory/memoryXXX/state
```

---

**Note:** current limit: this memory block must be adjacent to ZONE\_MOVABLE

---

Or you can explicitly request a kernel zone (usually ZONE\_NORMAL) by:

```
% echo online_kernel > /sys/devices/system/memory/memoryXXX/state
```

---

**Note:** current limit: this memory block must be adjacent to ZONE\_NORMAL

---

An explicit zone onlining can fail (e.g. when the range is already within and existing and incompatible zone already).

After this, memory block XXX' s state will be ‘online’ and the amount of available memory will be increased.

This may be changed in future.

## 54.6.6 Logical memory remove

### Memory offline and ZONE\_MOVABLE

Memory offlining is more complicated than memory online. Because memory offline has to make the whole memory block be unused, memory offline can fail if the memory block includes memory which cannot be freed.

In general, memory offline can use 2 techniques.

- (1) reclaim and free all memory in the memory block.
- (2) migrate all pages in the memory block.

In the current implementation, Linux's memory offline uses method (2), freeing all pages in the memory block by page migration. But not all pages are migratable. Under current Linux, migratable pages are anonymous pages and page caches. For offlining a memory block by migration, the kernel has to guarantee that the memory block contains only migratable pages.

Now, a boot option for making a memory block which consists of migratable pages is supported. By specifying "kernelcore=" or "movablecore=" boot option, you can create ZONE\_MOVABLE...a zone which is just used for movable pages. (See also Documentation/admin-guide/kernel-parameters.rst)

Assume the system has "TOTAL" amount of memory at boot time, this boot option creates ZONE\_MOVABLE as following.

- 1) When kernelcore=YYYY boot option is used, Size of memory not for movable pages (not for offline) is YYYY. Size of memory for movable pages (for offline) is TOTAL-YYYY.
- 2) When movablecore=ZZZZ boot option is used, Size of memory not for movable pages (not for offline) is TOTAL - ZZZZ. Size of memory for movable pages (for offline) is ZZZZ.

---

**Note:** Unfortunately, there is no information to show which memory block belongs to ZONE\_MOVABLE. This is TBD.

---

### How to offline memory

You can offline a memory block by using the same sysfs interface that was used in memory onlining:

```
% echo offline > /sys/devices/system/memory/memoryXXX/state
```

If offline succeeds, the state of the memory block is changed to be "offline". If it fails, some error code (like -EBUSY) will be returned by the kernel. Even if a memory block does not belong to ZONE\_MOVABLE, you can try to offline it. If it doesn't contain 'unmovable' memory, you'll get success.

A memory block under ZONE\_MOVABLE is considered to be able to be offlined easily. But under some busy state, it may return -EBUSY. Even if a memory block cannot be offlined due to -EBUSY, you can retry offlining it and may be able to

offline it (or not). (For example, a page is referred to by some kernel internal call and released soon.)

**Consideration:** Memory hotplug's design direction is to make the possibility of memory offlining higher and to guarantee unplugging memory under any situation. But it needs more work. Returning `-EBUSY` under some situation may be good because the user can decide to retry more or not by himself. Currently, memory offlining code does some amount of retry with 120 seconds timeout.

### 54.6.7 Physical memory remove

**Need more implementation yet...**

- Notification completion of remove works by OS to firmware.
- Guard from remove if not yet.

### 54.6.8 Locking Internals

When adding/removing memory that uses memory block devices (i.e. ordinary RAM), the `device_hotplug_lock` should be held to:

- synchronize against online/offline requests (e.g. via `sysfs`). This way, memory block devices can only be accessed (`.online/.state` attributes) by user space once memory has been fully added. And when removing memory, we know nobody is in critical sections.
- synchronize against CPU hotplug and similar (e.g. relevant for ACPI and PPC)

Especially, there is a possible lock inversion that is avoided using `device_hotplug_lock` when adding memory and user space tries to online that memory faster than expected:

- `device_online()` will first take the `device_lock()`, followed by `mem_hotplug_lock`
- `add_memory_resource()` will first take the `mem_hotplug_lock`, followed by the `device_lock()` (while creating the devices, during `bus_add_device()`).

As the device is visible to user space before taking the `device_lock()`, this can result in a lock inversion.

onlining/offlining of memory should be done via `device_online()/device_offline()` - to make sure it is properly synchronized to actions via `sysfs`. Holding `device_hotplug_lock` is advised (to e.g. protect `online_type`)

When adding/removing/onlining/offlining memory or adding/removing heterogeneous/device memory, we should always hold the `mem_hotplug_lock` in write mode to serialise memory hotplug (e.g. access to global/zone variables).

In addition, `mem_hotplug_lock` (in contrast to `device_hotplug_lock`) in read mode allows for a quite efficient `get_online_mems/put_online_mems` implementation, so code accessing memory can protect from that memory vanishing.

### 54.6.9 Future Work

- allowing memory hot-add to ZONE\_MOVABLE. maybe we need some switch like sysctl or new control file.
- showing memory block and physical device relationship.
- test and make it better memory offlining.
- support HugeTLB page migration and offlining.
- mmap removing at memory offline.
- physical remove memory.

## 54.7 NUMA Memory Policy

### 54.7.1 What is NUMA Memory Policy?

In the Linux kernel, “memory policy” determines from which node the kernel will allocate memory in a NUMA system or in an emulated NUMA system. Linux has supported platforms with Non-Uniform Memory Access architectures since 2.4.?. The current memory policy support was added to Linux 2.6 around May 2004. This document attempts to describe the concepts and APIs of the 2.6 memory policy support.

Memory policies should not be confused with cgroups (Documentation/admin-guide/cgroup-v1/cgroups.rst) which is an administrative mechanism for restricting the nodes from which memory may be allocated by a set of processes. Memory policies are a programming interface that a NUMA-aware application can take advantage of. When both cgroups and policies are applied to a task, the restrictions of the cgroup takes priority. See Memory Policies and cgroups below for more details.

### 54.7.2 Memory Policy Concepts

#### Scope of Memory Policies

The Linux kernel supports `_scopes_` of memory policy, described here from most general to most specific:

**System Default Policy** this policy is “hard coded” into the kernel. It is the policy that governs all page allocations that aren’t controlled by one of the more specific policy scopes discussed below. When the system is “up and running”, the system default policy will use “local allocation” described below. However, during boot up, the system default policy will be set to interleave allocations across all nodes with “sufficient” memory, so as not to overload the initial boot node with boot-time allocations.

**Task/Process Policy** this is an optional, per-task policy. When defined for a specific task, this policy controls all page allocations made by or on behalf of the task that aren’t controlled by a more specific scope. If a task does not define

a task policy, then all page allocations that would have been controlled by the task policy “fall back” to the System Default Policy.

The task policy applies to the entire address space of a task. Thus, it is inheritable, and indeed is inherited, across both `fork()` [`clone()` w/o the `CLONE_VM` flag] and `exec*()`. This allows a parent task to establish the task policy for a child task `exec()`'d from an executable image that has no awareness of memory policy. See the Memory Policy APIs section, below, for an overview of the system call that a task may use to set/change its task/process policy.

In a multi-threaded task, task policies apply only to the thread [Linux kernel task] that installs the policy and any threads subsequently created by that thread. Any sibling threads existing at the time a new task policy is installed retain their current policy.

A task policy applies only to pages allocated after the policy is installed. Any pages already faulted in by the task when the task changes its task policy remain where they were allocated based on the policy at the time they were allocated.

**VMA Policy** A “VMA” or “Virtual Memory Area” refers to a range of a task’s virtual address space. A task may define a specific policy for a range of its virtual address space. See the Memory Policy APIs section, below, for an overview of the `mbind()` system call used to set a VMA policy.

A VMA policy will govern the allocation of pages that back this region of the address space. Any regions of the task’s address space that don’t have an explicit VMA policy will fall back to the task policy, which may itself fall back to the System Default Policy.

VMA policies have a few complicating details:

- VMA policy applies ONLY to anonymous pages. These include pages allocated for anonymous segments, such as the task stack and heap, and any regions of the address space `mmap()`ed with the `MAP_ANONYMOUS` flag. If a VMA policy is applied to a file mapping, it will be ignored if the mapping used the `MAP_SHARED` flag. If the file mapping used the `MAP_PRIVATE` flag, the VMA policy will only be applied when an anonymous page is allocated on an attempt to write to the mapping- i.e., at Copy-On-Write.
- VMA policies are shared between all tasks that share a virtual address space-a.k.a. threads-independent of when the policy is installed; and they are inherited across `fork()`. However, because VMA policies refer to a specific region of a task’s address space, and because the address space is discarded and recreated on `exec*()`, VMA policies are NOT inheritable across `exec()`. Thus, only NUMA-aware applications may use VMA policies.
- A task may install a new VMA policy on a sub-range of a previously `mmap()`ed region. When this happens, Linux splits the existing virtual memory area into 2 or 3 VMAs, each with its own policy.
- By default, VMA policy applies only to pages allocated after the policy is installed. Any pages already faulted into the VMA range remain where they were allocated based on the policy at the time they were allocated.

However, since 2.6.16, Linux supports page migration via the `mbind()` system call, so that page contents can be moved to match a newly installed policy.

**Shared Policy** Conceptually, shared policies apply to “memory objects” mapped shared into one or more tasks’ distinct address spaces. An application installs shared policies the same way as VMA policies—using the `mbind()` system call specifying a range of virtual addresses that map the shared object. However, unlike VMA policies, which can be considered to be an attribute of a range of a task’ s address space, shared policies apply directly to the shared object. Thus, all tasks that attach to the object share the policy, and all pages allocated for the shared object, by any task, will obey the shared policy.

As of 2.6.22, only shared memory segments, created by `shmget()` or `mmap(MAP_ANONYMOUS|MAP_SHARED)`, support shared policy. When shared policy support was added to Linux, the associated data structures were added to hugetlbfs shmem segments. At the time, hugetlbfs did not support allocation at fault time—a.k.a lazy allocation—so hugetlbfs shmem segments were never “hooked up” to the shared policy support. Although hugetlbfs segments now support lazy allocation, their support for shared policy has not been completed.

As mentioned above in VMA policies section, allocations of page cache pages for regular files `mmap()`ed with `MAP_SHARED` ignore any VMA policy installed on the virtual address range backed by the shared file mapping. Rather, shared page cache pages, including pages backing private mappings that have not yet been written by the task, follow task policy, if any, else System Default Policy.

The shared policy infrastructure supports different policies on subset ranges of the shared object. However, Linux still splits the VMA of the task that installs the policy for each range of distinct policy. Thus, different tasks that attach to a shared memory segment can have different VMA configurations mapping that one shared object. This can be seen by examining the `/proc/<pid>/numa_maps` of tasks sharing a shared memory region, when one task has installed shared policy on one or more ranges of the region.

## Components of Memory Policies

A NUMA memory policy consists of a “mode”, optional mode flags, and an optional set of nodes. The mode determines the behavior of the policy, the optional mode flags determine the behavior of the mode, and the optional set of nodes can be viewed as the arguments to the policy behavior.

Internally, memory policies are implemented by a reference counted structure, `struct mempolicy`. Details of this structure will be discussed in context, below, as required to explain the behavior.

NUMA memory policy supports the following 4 behavioral modes:

**Default Mode-MPOL\_DEFAULT** This mode is only used in the memory policy APIs. Internally, `MPOL_DEFAULT` is converted to the `NULL` memory policy in all policy scopes. Any existing non-default policy will simply be removed

when `MPOL_DEFAULT` is specified. As a result, `MPOL_DEFAULT` means “fall back to the next most specific policy scope.”

For example, a `NULL` or default task policy will fall back to the system default policy. A `NULL` or default vma policy will fall back to the task policy.

When specified in one of the memory policy APIs, the Default mode does not use the optional set of nodes.

It is an error for the set of nodes specified for this policy to be non-empty.

**MPOL\_BIND** This mode specifies that memory must come from the set of nodes specified by the policy. Memory will be allocated from the node in the set with sufficient free memory that is closest to the node where the allocation takes place.

**MPOL\_PREFERRED** This mode specifies that the allocation should be attempted from the single node specified in the policy. If that allocation fails, the kernel will search other nodes, in order of increasing distance from the preferred node based on information provided by the platform firmware.

Internally, the Preferred policy uses a single node—the `preferred_node` member of `struct mempolicy`. When the internal mode flag `MPOL_F_LOCAL` is set, the `preferred_node` is ignored and the policy is interpreted as local allocation. “Local” allocation policy can be viewed as a Preferred policy that starts at the node containing the `cpu` where the allocation takes place.

It is possible for the user to specify that local allocation is always preferred by passing an empty `nodemask` with this mode. If an empty `nodemask` is passed, the policy cannot use the `MPOL_F_STATIC_NODES` or `MPOL_F_RELATIVE_NODES` flags described below.

**MPOL\_INTERLEAVED** This mode specifies that page allocations be interleaved, on a page granularity, across the nodes specified in the policy. This mode also behaves slightly differently, based on the context where it is used:

For allocation of anonymous pages and shared memory pages, Interleave mode indexes the set of nodes specified by the policy using the page offset of the faulting address into the segment [VMA] containing the address modulo the number of nodes specified by the policy. It then attempts to allocate a page, starting at the selected node, as if the node had been specified by a Preferred policy or had been selected by a local allocation. That is, allocation will follow the per node `zonelist`.

For allocation of page cache pages, Interleave mode indexes the set of nodes specified by the policy using a node counter maintained per task. This counter wraps around to the lowest specified node after it reaches the highest specified node. This will tend to spread the pages out over the nodes specified by the policy based on the order in which they are allocated, rather than based on any page offset into an address range or file. During system boot up, the temporary interleaved system default policy works in this mode.

NUMA memory policy supports the following optional mode flags:

**MPOL\_F\_STATIC\_NODES** This flag specifies that the `nodemask` passed by the user should not be remapped if the task or VMA’s set of allowed nodes changes after the memory policy has been defined.

Without this flag, any time a mempolicy is rebound because of a change in the set of allowed nodes, the node (Preferred) or nodemask (Bind, Interleave) is remapped to the new set of allowed nodes. This may result in nodes being used that were previously undesired.

With this flag, if the user-specified nodes overlap with the nodes allowed by the task's cpuset, then the memory policy is applied to their intersection. If the two sets of nodes do not overlap, the Default policy is used.

For example, consider a task that is attached to a cpuset with mems 1-3 that sets an Interleave policy over the same set. If the cpuset's mems change to 3-5, the Interleave will now occur over nodes 3, 4, and 5. With this flag, however, since only node 3 is allowed from the user's nodemask, the "interleave" only occurs over that node. If no nodes from the user's nodemask are now allowed, the Default behavior is used.

`MPOL_F_STATIC_NODES` cannot be combined with the `MPOL_F_RELATIVE_NODES` flag. It also cannot be used for `MPOL_PREFERRED` policies that were created with an empty nodemask (local allocation).

**MPOL\_F\_RELATIVE\_NODES** This flag specifies that the nodemask passed by the user will be mapped relative to the set of the task or VMA's set of allowed nodes. The kernel stores the user-passed nodemask, and if the allowed nodes changes, then that original nodemask will be remapped relative to the new set of allowed nodes.

Without this flag (and without `MPOL_F_STATIC_NODES`), anytime a mempolicy is rebound because of a change in the set of allowed nodes, the node (Preferred) or nodemask (Bind, Interleave) is remapped to the new set of allowed nodes. That remap may not preserve the relative nature of the user's passed nodemask to its set of allowed nodes upon successive rebinds: a nodemask of 1,3,5 may be remapped to 7-9 and then to 1-3 if the set of allowed nodes is restored to its original state.

With this flag, the remap is done so that the node numbers from the user's passed nodemask are relative to the set of allowed nodes. In other words, if nodes 0, 2, and 4 are set in the user's nodemask, the policy will be effected over the first (and in the Bind or Interleave case, the third and fifth) nodes in the set of allowed nodes. The nodemask passed by the user represents nodes relative to task or VMA's set of allowed nodes.

If the user's nodemask includes nodes that are outside the range of the new set of allowed nodes (for example, node 5 is set in the user's nodemask when the set of allowed nodes is only 0-3), then the remap wraps around to the beginning of the nodemask and, if not already set, sets the node in the mempolicy nodemask.

For example, consider a task that is attached to a cpuset with mems 2-5 that sets an Interleave policy over the same set with `MPOL_F_RELATIVE_NODES`. If the cpuset's mems change to 3-7, the interleave now occurs over nodes 3,5-7. If the cpuset's mems then change to 0,2-3,5, then the interleave occurs over nodes 0,2-3,5.

Thanks to the consistent remapping, applications preparing nodemasks to specify memory policies using this flag should disregard their current, actual

cpuset imposed memory placement and prepare the nodemask as if they were always located on memory nodes 0 to N-1, where N is the number of memory nodes the policy is intended to manage. Let the kernel then remap to the set of memory nodes allowed by the task's cpuset, as that may change over time.

`MPOL_F_RELATIVE_NODES` cannot be combined with the `MPOL_F_STATIC_NODES` flag. It also cannot be used for `MPOL_PREFERRED` policies that were created with an empty nodemask (local allocation).

### 54.7.3 Memory Policy Reference Counting

To resolve use/free races, struct mempolicy contains an atomic reference count field. Internal interfaces, `mpol_get()/mpol_put()` increment and decrement this reference count, respectively. `mpol_put()` will only free the structure back to the mempolicy kmem cache when the reference count goes to zero.

When a new memory policy is allocated, its reference count is initialized to '1', representing the reference held by the task that is installing the new policy. When a pointer to a memory policy structure is stored in another structure, another reference is added, as the task's reference will be dropped on completion of the policy installation.

During run-time "usage" of the policy, we attempt to minimize atomic operations on the reference count, as this can lead to cache lines bouncing between cpus and NUMA nodes. "Usage" here means one of the following:

- 1) querying of the policy, either by the task itself [using the `get_mempolicy()` API discussed below] or by another task using the `/proc/<pid>/numa_maps` interface.
- 2) examination of the policy to determine the policy mode and associated node or node lists, if any, for page allocation. This is considered a "hot path". Note that for `MPOL_BIND`, the "usage" extends across the entire allocation process, which may sleep during page reclamation, because the `BIND` policy nodemask is used, by reference, to filter ineligible nodes.

We can avoid taking an extra reference during the usages listed above as follows:

- 1) we never need to get/free the system default policy as this is never changed nor freed, once the system is up and running.
- 2) for querying the policy, we do not need to take an extra reference on the target task's task policy nor vma policies because we always acquire the task's mm's `mmap_lock` for read during the query. The `set_mempolicy()` and `mbind()` APIs [see below] always acquire the `mmap_lock` for write when installing or replacing task or vma policies. Thus, there is no possibility of a task or thread freeing a policy while another task or thread is querying it.
- 3) Page allocation usage of task or vma policy occurs in the fault path where we hold them `mmap_lock` for read. Again, because replacing the task or vma policy requires that the `mmap_lock` be held for write, the policy can't be freed out from under us while we're using it for page allocation.
- 4) Shared policies require special consideration. One task can replace a shared memory policy while another task, with a distinct `mmap_lock`, is querying or allocating a page based on the policy. To resolve this potential race, the

shared policy infrastructure adds an extra reference to the shared policy during lookup while holding a spin lock on the shared policy management structure. This requires that we drop this extra reference when we're finished "using" the policy. We must drop the extra reference on shared policies in the same query/allocation paths used for non-shared policies. For this reason, shared policies are marked as such, and the extra reference is dropped "conditionally" -i.e., only for shared policies.

Because of this extra reference counting, and because we must lookup shared policies in a tree structure under spinlock, shared policies are more expensive to use in the page allocation path. This is especially true for shared policies on shared memory regions shared by tasks running on different NUMA nodes. This extra overhead can be avoided by always falling back to task or system default policy for shared memory regions, or by prefaulting the entire shared memory region into memory and locking it down. However, this might not be appropriate for all applications.

#### 54.7.4 Memory Policy APIs

Linux supports 3 system calls for controlling memory policy. These APIs always affect only the calling task, the calling task's address space, or some shared object mapped into the calling task's address space.

---

**Note:** the headers that define these APIs and the parameter data types for user space applications reside in a package that is not part of the Linux kernel. The kernel system call interfaces, with the 'sys\_' prefix, are defined in <linux/syscalls.h>; the mode and flag definitions are defined in <linux/mempolicy.h>.

---

Set [Task] Memory Policy:

```
long set_mempolicy(int mode, const unsigned long *nmask,
                  unsigned long maxnode);
```

Set's the calling task's "task/process memory policy" to mode specified by the 'mode' argument and the set of nodes defined by 'nmask'. 'nmask' points to a bit mask of node ids containing at least 'maxnode' ids. Optional mode flags may be passed by combining the 'mode' argument with the flag (for example: MPOL\_INTERLEAVE | MPOL\_F\_STATIC\_NODES).

See the set\_mempolicy(2) man page for more details

Get [Task] Memory Policy or Related Information:

```
long get_mempolicy(int *mode,
                  const unsigned long *nmask, unsigned long maxnode,
                  void *addr, int flags);
```

Queries the "task/process memory policy" of the calling task, or the policy or location of a specified virtual address, depending on the 'flags' argument.

See the get\_mempolicy(2) man page for more details

Install VMA/Shared Policy for a Range of Task's Address Space:

```
long mbind(void *start, unsigned long len, int mode,
           const unsigned long *nmask, unsigned long maxnode,
           unsigned flags);
```

`mbind()` installs the policy specified by (`mode`, `nmask`, `maxnodes`) as a VMA policy for the range of the calling task's address space specified by the 'start' and 'len' arguments. Additional actions may be requested via the 'flags' argument.

See the `mbind(2)` man page for more details.

### 54.7.5 Memory Policy Command Line Interface

Although not strictly part of the Linux implementation of memory policy, a command line tool, `numactl(8)`, exists that allows one to:

- set the task policy for a specified program via `set_mempolicy(2)`, `fork(2)` and `exec(2)`
- set the shared policy for a shared memory segment via `mbind(2)`

The `numactl(8)` tool is packaged with the run-time version of the library containing the memory policy system call wrappers. Some distributions package the headers and compile-time libraries in a separate development package.

### 54.7.6 Memory Policies and cpusets

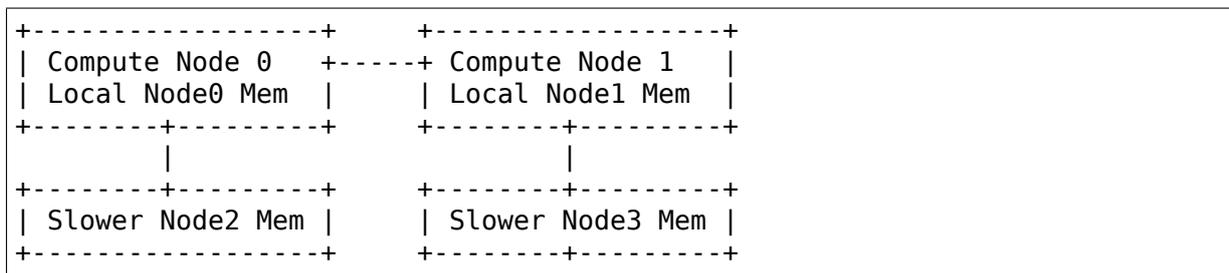
Memory policies work within cpusets as described above. For memory policies that require a node or set of nodes, the nodes are restricted to the set of nodes whose memories are allowed by the cpuset constraints. If the node-mask specified for the policy contains nodes that are not allowed by the cpuset and `MPOL_F_RELATIVE_NODES` is not used, the intersection of the set of nodes specified for the policy and the set of nodes with memory is used. If the result is the empty set, the policy is considered invalid and cannot be installed. If `MPOL_F_RELATIVE_NODES` is used, the policy's nodes are mapped onto and folded into the task's set of allowed nodes as previously described.

The interaction of memory policies and cpusets can be problematic when tasks in two cpusets share access to a memory region, such as shared memory segments created by `shmget()` or `mmap()` with the `MAP_ANONYMOUS` and `MAP_SHARED` flags, and any of the tasks install shared policy on the region, only nodes whose memories are allowed in both cpusets may be used in the policies. Obtaining this information requires "stepping outside" the memory policy APIs to use the cpuset information and requires that one know in what cpusets other task might be attaching to the shared region. Furthermore, if the cpusets' allowed memory sets are disjoint, "local" allocation is the only valid policy.

## 54.8 NUMA Locality

Some platforms may have multiple types of memory attached to a compute node. These disparate memory ranges may share some characteristics, such as CPU cache coherence, but may have different performance. For example, different media types and buses affect bandwidth and latency.

A system supports such heterogeneous memory by grouping each memory type under different domains, or “nodes”, based on locality and performance characteristics. Some memory may share the same node as a CPU, and others are provided as memory only nodes. While memory only nodes do not provide CPUs, they may still be local to one or more compute nodes relative to other nodes. The following diagram shows one such example of two compute nodes with local memory and a memory only node for each of compute node:



A “memory initiator” is a node containing one or more devices such as CPUs or separate memory I/O devices that can initiate memory requests. A “memory target” is a node containing one or more physical address ranges accessible from one or more memory initiators.

When multiple memory initiators exist, they may not all have the same performance when accessing a given memory target. Each initiator-target pair may be organized into different ranked access classes to represent this relationship. The highest performing initiator to a given target is considered to be one of that target’s local initiators, and given the highest access class, 0. Any given target may have one or more local initiators, and any given initiator may have multiple local memory targets.

To aid applications matching memory targets with their initiators, the kernel provides symlinks to each other. The following example lists the relationship for the access class “0” memory initiators and targets:

```

# symlinks -v /sys/devices/system/node/nodeX/access0/targets/
relative: /sys/devices/system/node/nodeX/access0/targets/nodeY -> ../../
↳ nodeY

# symlinks -v /sys/devices/system/node/nodeY/access0/initiators/
relative: /sys/devices/system/node/nodeY/access0/initiators/nodeX -> ../../
↳ nodeX

```

A memory initiator may have multiple memory targets in the same access class. The target memory’s initiators in a given class indicate the nodes’ access characteristics share the same performance relative to other linked initiator nodes. Each target within an initiator’s access class, though, do not necessarily perform the same as each other.

## 54.9 NUMA Performance

Applications may wish to consider which node they want their memory to be allocated from based on the node's performance characteristics. If the system provides these attributes, the kernel exports them under the node sysfs hierarchy by appending the attributes directory under the memory node's access class 0 initiators as follows:

```
/sys/devices/system/node/nodeY/access0/initiators/
```

These attributes apply only when accessed from nodes that have the are linked under the this access's initiators.

The performance characteristics the kernel provides for the local initiators are exported are as follows:

```
# tree -P "read*|write*" /sys/devices/system/node/nodeY/access0/initiators/
/sys/devices/system/node/nodeY/access0/initiators/
|-- read_bandwidth
|-- read_latency
|-- write_bandwidth
`-- write_latency
```

The bandwidth attributes are provided in MiB/second.

The latency attributes are provided in nanoseconds.

The values reported here correspond to the rated latency and bandwidth for the platform.

## 54.10 NUMA Cache

System memory may be constructed in a hierarchy of elements with various performance characteristics in order to provide large address space of slower performing memory cached by a smaller higher performing memory. The system physical addresses memory initiators are aware of are provided by the last memory level in the hierarchy. The system meanwhile uses higher performing memory to transparently cache access to progressively slower levels.

The term "far memory" is used to denote the last level memory in the hierarchy. Each increasing cache level provides higher performing initiator access, and the term "near memory" represents the fastest cache provided by the system.

This numbering is different than CPU caches where the cache level (ex: L1, L2, L3) uses the CPU-side view where each increased level is lower performing. In contrast, the memory cache level is centric to the last level memory, so the higher numbered cache level corresponds to memory nearer to the CPU, and further from far memory.

The memory-side caches are not directly addressable by software. When software accesses a system address, the system will return it from the near memory cache if it is present. If it is not present, the system accesses the next level of memory until there is either a hit in that cache level, or it reaches far memory.

An application does not need to know about caching attributes in order to use the system. Software may optionally query the memory cache attributes in order to maximize the performance out of such a setup. If the system provides a way for the kernel to discover this information, for example with ACPI HMAT (Heterogeneous Memory Attribute Table), the kernel will append these attributes to the NUMA node memory target.

When the kernel first registers a memory cache with a node, the kernel will create the following directory:

```
/sys/devices/system/node/nodeX/memory_side_cache/
```

If that directory is not present, the system either does not provide a memory-side cache, or that information is not accessible to the kernel.

The attributes for each level of cache is provided under its cache level index:

```
/sys/devices/system/node/nodeX/memory_side_cache/indexA/  
/sys/devices/system/node/nodeX/memory_side_cache/indexB/  
/sys/devices/system/node/nodeX/memory_side_cache/indexC/
```

Each cache level's directory provides its attributes. For example, the following shows a single cache level and the attributes available for software to query:

```
# tree sys/devices/system/node/node0/memory_side_cache/  
/sys/devices/system/node/node0/memory_side_cache/  
|-- index1  
|   |-- indexing  
|   |-- line_size  
|   |-- size  
|   `-- write_policy
```

The “indexing” will be 0 if it is a direct-mapped cache, and non-zero for any other indexed based, multi-way associativity.

The “line\_size” is the number of bytes accessed from the next cache level on a miss.

The “size” is the number of bytes provided by this cache level.

The “write\_policy” will be 0 for write-back, and non-zero for write-through caching.

## 54.11 See Also

[1] [https://www.uefi.org/sites/default/files/resources/ACPI\\_6\\_2.pdf](https://www.uefi.org/sites/default/files/resources/ACPI_6_2.pdf) - Section 5.2.27

## 54.12 Examining Process Page Tables

`pagemap` is a new (as of 2.6.25) set of interfaces in the kernel that allow userspace programs to examine the page tables and related information by reading files in `/proc`.

There are four components to `pagemap`:

- `/proc/pid/pagemap`. This file lets a userspace process find out which physical frame each virtual page is mapped to. It contains one 64-bit value for each virtual page, containing the following data (from `fs/proc/task_mmu.c`, above `pagemap_read`):
  - Bits 0-54 page frame number (PFN) if present
  - Bits 0-4 swap type if swapped
  - Bits 5-54 swap offset if swapped
  - Bit 55 pte is soft-dirty (see [Documentation/admin-guide/mm/soft-dirty.rst](#))
  - Bit 56 page exclusively mapped (since 4.2)
  - Bits 57-60 zero
  - Bit 61 page is file-page or shared-anon (since 3.5)
  - Bit 62 page swapped
  - Bit 63 page present

Since Linux 4.0 only users with the `CAP_SYS_ADMIN` capability can get PFNs. In 4.0 and 4.1 opens by unprivileged fail with `-EPERM`. Starting from 4.2 the PFN field is zeroed if the user does not have `CAP_SYS_ADMIN`. Reason: information about PFNs helps in exploiting Rowhammer vulnerability.

If the page is not present but in swap, then the PFN contains an encoding of the swap file number and the page's offset into the swap. Unmapped pages return a null PFN. This allows determining precisely which pages are mapped (or in swap) and comparing mapped pages between processes.

Efficient users of this interface will use `/proc/pid/maps` to determine which areas of memory are actually mapped and `llseek` to skip over unmapped regions.

- `/proc/kpagecount`. This file contains a 64-bit count of the number of times each page is mapped, indexed by PFN.

The `page-types` tool in the `tools/vm` directory can be used to query the number of times a page is mapped.

- `/proc/kpageflags`. This file contains a 64-bit set of flags for each page, indexed by PFN.

The flags are (from `fs/proc/page.c`, above `kpageflags_read`):

0. LOCKED
1. ERROR

2. REFERENCED
  3. UPTODATE
  4. DIRTY
  5. LRU
  6. ACTIVE
  7. SLAB
  8. WRITEBACK
  9. RECLAIM
  10. BUDDY
  11. MMAP
  12. ANON
  13. SWAPCACHE
  14. SWAPBACKED
  15. COMPOUND\_HEAD
  16. COMPOUND\_TAIL
  17. HUGE
  18. UNEVICTABLE
  19. HWPOISON
  20. NOPAGE
  21. KSM
  22. THP
  23. OFFLINE
  24. ZERO\_PAGE
  25. IDLE
  26. PGTABLE
- `/proc/kpagecgroup`. This file contains a 64-bit inode number of the memory cgroup each page is charged to, indexed by PFN. Only available when `CONFIG_MEMCG` is set.

### 54.12.1 Short descriptions to the page flags

- 0 - LOCKED** page is being locked for exclusive access, e.g. by undergoing read/write IO
- 7 - SLAB** page is managed by the SLAB/SLOB/SLUB/SLQB kernel memory allocator. When compound page is used, SLUB/SLQB will only set this flag on the head page; SLOB will not flag it at all.
- 10 - BUDDY** a free memory block managed by the buddy system allocator. The buddy system organizes free memory in blocks of various orders. An order N block has  $2^N$  physically contiguous pages, with the BUDDY flag set for and `_only_` for the first page.
- 15 - COMPOUND\_HEAD** A compound page with order N consists of  $2^N$  physically contiguous pages. A compound page with order 2 takes the form of “HTTT”, where H donates its head page and T donates its tail page(s). The major consumers of compound pages are hugeTLB pages (Documentation/admin-guide/mm/hugetlbpage.rst), the SLUB etc. memory allocators and various device drivers. However in this interface, only huge/giga pages are made visible to end users.
- 16 - COMPOUND\_TAIL** A compound page tail (see description above).
- 17 - HUGE** this is an integral part of a HugeTLB page
- 19 - HWPOISON** hardware detected memory corruption on this page: don't touch the data!
- 20 - NOPAGE** no page frame exists at the requested address
- 21 - KSM** identical memory pages dynamically shared between one or more processes
- 22 - THP** contiguous pages which construct transparent hugepages
- 23 - OFFLINE** page is logically offline
- 24 - ZERO\_PAGE** zero page for `pfm_zero` or `huge_zero` page
- 25 - IDLE** page has not been accessed since it was marked idle (see Documentation/admin-guide/mm/idle\_page\_tracking.rst). Note that this flag may be stale in case the page was accessed via a PTE. To make sure the flag is up-to-date one has to read `/sys/kernel/mm/page_idle/bitmap` first.
- 26 - PGTABLE** page is in use as a page table

### IO related page flags

- 1 - ERROR** IO error occurred
- 3 - UPTODATE** page has up-to-date data ie. for file backed page: (in-memory data revision  $\geq$  on-disk one)
- 4 - DIRTY** page has been written to, hence contains new data i.e. for file backed page: (in-memory data revision  $>$  on-disk one)
- 8 - WRITEBACK** page is being synced to disk

## LRU related page flags

**5 - LRU** page is in one of the LRU lists

**6 - ACTIVE** page is in the active LRU list

**18 - UNEVICTABLE** page is in the unevictable (non-)LRU list It is somehow pinned and not a candidate for LRU page reclaims, e.g. ramfs pages, shmctl(SHM\_LOCK) and mlock() memory segments

**2 - REFERENCED** page has been referenced since last LRU list enqueue/requeue

**9 - RECLAIM** page will be reclaimed soon after its pageout IO completed

**11 - MMAP** a memory mapped page

**12 - ANON** a memory mapped page that is not part of a file

**13 - SWAPCACHE** page is mapped to swap space, i.e. has an associated swap entry

**14 - SWAPBACKED** page is backed by swap/RAM

The page-types tool in the tools/vm directory can be used to query the above flags.

### 54.12.2 Using pagemap to do something useful

The general procedure for using pagemap to find out about a process' memory usage goes like this:

1. Read /proc/pid/maps to determine which parts of the memory space are mapped to what.
2. Select the maps you are interested in - all of them, or a particular library, or the stack or the heap, etc.
3. Open /proc/pid/pagemap and seek to the pages you would like to examine.
4. Read a u64 for each page from pagemap.
5. Open /proc/kpagecount and/or /proc/kpageflags. For each PFN you just read, seek to that entry in the file, and read the data you want.

For example, to find the “unique set size” (USS), which is the amount of memory that a process is using that is not shared with any other process, you can go through every map in the process, find the PFNs, look those up in kpagecount, and tally up the number of pages that are only referenced once.

### 54.12.3 Other notes

Reading from any of the files will return `-EINVAL` if you are not starting the read on an 8-byte boundary (e.g., if you sought an odd number of bytes into the file), or if the size of the read is not a multiple of 8 bytes.

Before Linux 3.11 pagemap bits 55-60 were used for “page-shift” (which is always 12 at most architectures). Since Linux 3.11 their meaning changes after first clear of soft-dirty bits. Since Linux 4.2 they are used for flags unconditionally.

### 54.13 Soft-Dirty PTEs

The soft-dirty is a bit on a PTE which helps to track which pages a task writes to. In order to do this tracking one should

1. Clear soft-dirty bits from the task's PTEs.

This is done by writing “4” into the `/proc/PID/clear_refs` file of the task in question.

2. Wait some time.

3. Read soft-dirty bits from the PTEs.

This is done by reading from the `/proc/PID/pagemap`. The bit 55 of the 64-bit `qword` is the soft-dirty one. If set, the respective PTE was written to since step 1.

Internally, to do this tracking, the writable bit is cleared from PTEs when the soft-dirty bit is cleared. So, after this, when the task tries to modify a page at some virtual address the `#PF` occurs and the kernel sets the soft-dirty bit on the respective PTE.

Note, that although all the task's address space is marked as r/o after the soft-dirty bits clear, the `#PF`s that occur after that are processed fast. This is so, since the pages are still mapped to physical memory, and thus all the kernel does is find this fact out and puts both writable and soft-dirty bits on the PTE.

While in most cases tracking memory changes by `#PF`s is more than enough there is still a scenario when we can lose soft dirty bits – a task unmaps a previously mapped memory region and then maps a new one at exactly the same place. When `unmap` is called, the kernel internally clears PTE values including soft dirty bits. To notify user space application about such memory region renewal the kernel always marks new memory regions (and expanded regions) as soft dirty.

This feature is actively used by the checkpoint-restore project. You can find more details about it on <http://criu.org>

– Pavel Emelyanov, Apr 9, 2013

## 54.14 Transparent Hugepage Support

### 54.14.1 Objective

Performance critical computing applications dealing with large memory working sets are already running on top of libhugetlbfs and in turn hugetlbfs. Transparent HugePage Support (THP) is an alternative mean of using huge pages for the backing of virtual memory with huge pages that supports the automatic promotion and demotion of page sizes and without the shortcomings of hugetlbfs.

Currently THP only works for anonymous memory mappings and tmpfs/shmem. But in the future it can expand to other filesystems.

---

**Note:** in the examples below we presume that the basic page size is 4K and the huge page size is 2M, although the actual numbers may vary depending on the CPU architecture.

---

The reason applications are running faster is because of two factors. The first factor is almost completely irrelevant and it's not of significant interest because it'll also have the downside of requiring larger clear-page copy-page in page faults which is a potentially negative effect. The first factor consists in taking a single page fault for each 2M virtual region touched by userland (so reducing the enter/exit kernel frequency by a 512 times factor). This only matters the first time the memory is accessed for the lifetime of a memory mapping. The second long lasting and much more important factor will affect all subsequent accesses to the memory for the whole runtime of the application. The second factor consist of two components:

- 1) the TLB miss will run faster (especially with virtualization using nested pagetables but almost always also on bare metal without virtualization)
- 2) a single TLB entry will be mapping a much larger amount of virtual memory in turn reducing the number of TLB misses. With virtualization and nested pagetables the TLB can be mapped of larger size only if both KVM and the Linux guest are using hugepages but a significant speedup already happens if only one of the two is using hugepages just because of the fact the TLB miss is going to run faster.

THP can be enabled system wide or restricted to certain tasks or even memory ranges inside task's address space. Unless THP is completely disabled, there is khugepaged daemon that scans memory and collapses sequences of basic pages into huge pages.

The THP behaviour is controlled via sysfs interface and using `madvise(2)` and `prctl(2)` system calls.

Transparent Hugepage Support maximizes the usefulness of free memory if compared to the reservation approach of hugetlbfs by allowing all unused memory to be used as cache or other movable (or even unmovable entities). It doesn't require reservation to prevent hugepage allocation failures to be noticeable from userland. It allows paging and all other advanced VM features to be available on the hugepages. It requires no modifications for applications to take advantage of it.

Applications however can be further optimized to take advantage of this feature, like for example they've been optimized before to avoid a flood of `mmap` system calls for every `malloc(4k)`. Optimizing userland is by far not mandatory and `khugepaged` already can take care of long lived page allocations even for hugepage unaware applications that deals with large amounts of memory.

In certain cases when hugepages are enabled system wide, application may end up allocating more memory resources. An application may `mmap` a large region but only touch 1 byte of it, in that case a 2M page might be allocated instead of a 4k page for no good. This is why it's possible to disable hugepages system-wide and to only have them inside `MADV_HUGEPAGE` `madvise` regions.

Embedded systems should enable hugepages only inside `madvise` regions to eliminate any risk of wasting any precious byte of memory and to only run faster.

Applications that gets a lot of benefit from hugepages and that don't risk to lose memory by using hugepages, should use `madvise(MADV_HUGEPAGE)` on their critical `mmap`d regions.

### 54.14.2 sysfs

#### Global THP controls

Transparent Hugepage Support for anonymous memory can be entirely disabled (mostly for debugging purposes) or only enabled inside `MADV_HUGEPAGE` regions (to avoid the risk of consuming more memory resources) or enabled system wide. This can be achieved with one of:

```
echo always >/sys/kernel/mm/transparent_hugepage/enabled
echo madvise >/sys/kernel/mm/transparent_hugepage/enabled
echo never >/sys/kernel/mm/transparent_hugepage/enabled
```

It's also possible to limit defrag efforts in the VM to generate anonymous hugepages in case they're not immediately free to `madvise` regions or to never try to defrag memory and simply fallback to regular pages unless hugepages are immediately available. Clearly if we spend CPU time to defrag memory, we would expect to gain even more by the fact we use hugepages later instead of regular pages. This isn't always guaranteed, but it may be more likely in case the allocation is for a `MADV_HUGEPAGE` region.

```
echo always >/sys/kernel/mm/transparent_hugepage/defrag
echo defer >/sys/kernel/mm/transparent_hugepage/defrag
echo defer+madvise >/sys/kernel/mm/transparent_hugepage/defrag
echo madvise >/sys/kernel/mm/transparent_hugepage/defrag
echo never >/sys/kernel/mm/transparent_hugepage/defrag
```

**always** means that an application requesting THP will stall on allocation failure and directly reclaim pages and compact memory in an effort to allocate a THP immediately. This may be desirable for virtual machines that benefit heavily from THP use and are willing to delay the VM start to utilise them.

**defer** means that an application will wake `kswapd` in the background to reclaim pages and wake `kcompactd` to compact memory so that THP is available in

the near future. It's the responsibility of khugepaged to then install the THP pages later.

**defer+madvise** will enter direct reclaim and compaction like always, but only for regions that have used `madvise(MADV_HUGEPAGE)`; all other regions will wake `kswapd` in the background to reclaim pages and wake `kcompactd` to compact memory so that THP is available in the near future.

**madvise** will enter direct reclaim like always but only for regions that are have used `madvise(MADV_HUGEPAGE)`. This is the default behaviour.

**never** should be self-explanatory.

By default kernel tries to use huge zero page on read page fault to anonymous mapping. It's possible to disable huge zero page by writing 0 or enable it back by writing 1:

```
echo 0 >/sys/kernel/mm/transparent_hugepage/use_zero_page
echo 1 >/sys/kernel/mm/transparent_hugepage/use_zero_page
```

Some userspace (such as a test program, or an optimized memory allocation library) may want to know the size (in bytes) of a transparent hugepage:

```
cat /sys/kernel/mm/transparent_hugepage/hpage_pmd_size
```

khugepaged will be automatically started when `transparent_hugepage/enabled` is set to "always" or "madvise, and it'll be automatically shutdown if it's set to "never" .

## Khugepaged controls

khugepaged runs usually at low frequency so while one may not want to invoke defrag algorithms synchronously during the page faults, it should be worth invoking defrag at least in khugepaged. However it's also possible to disable defrag in khugepaged by writing 0 or enable defrag in khugepaged by writing 1:

```
echo 0 >/sys/kernel/mm/transparent_hugepage/khugepaged/defrag
echo 1 >/sys/kernel/mm/transparent_hugepage/khugepaged/defrag
```

You can also control how many pages khugepaged should scan at each pass:

```
/sys/kernel/mm/transparent_hugepage/khugepaged/pages_to_scan
```

and how many milliseconds to wait in khugepaged between each pass (you can set this to 0 to run khugepaged at 100% utilization of one core):

```
/sys/kernel/mm/transparent_hugepage/khugepaged/scan_sleep_millisecs
```

and how many milliseconds to wait in khugepaged if there's an hugepage allocation failure to throttle the next allocation attempt:

```
/sys/kernel/mm/transparent_hugepage/khugepaged/alloc_sleep_millisecs
```

The khugepaged progress can be seen in the number of pages collapsed:

```
/sys/kernel/mm/transparent_hugepage/khugepaged/pages_collapsed
```

for each pass:

```
/sys/kernel/mm/transparent_hugepage/khugepaged/full_scans
```

`max_ptes_none` specifies how many extra small pages (that are not already mapped) can be allocated when collapsing a group of small pages into one large page:

```
/sys/kernel/mm/transparent_hugepage/khugepaged/max_ptes_none
```

A higher value leads to use additional memory for programs. A lower value leads to gain less thp performance. Value of `max_ptes_none` can waste cpu time very little, you can ignore it.

`max_ptes_swap` specifies how many pages can be brought in from swap when collapsing a group of pages into a transparent huge page:

```
/sys/kernel/mm/transparent_hugepage/khugepaged/max_ptes_swap
```

A higher value can cause excessive swap IO and waste memory. A lower value can prevent THPs from being collapsed, resulting fewer pages being collapsed into THPs, and lower memory access performance.

`max_ptes_shared` specifies how many pages can be shared across multiple processes. Exceeding the number would block the collapse:

```
/sys/kernel/mm/transparent_hugepage/khugepaged/max_ptes_shared
```

A higher value may increase memory footprint for some workloads.

### 54.14.3 Boot parameter

You can change the sysfs boot time defaults of Transparent Hugepage Support by passing the parameter `transparent_hugepage=always` or `transparent_hugepage=madvise` or `transparent_hugepage=never` to the kernel command line.

### 54.14.4 Hugepages in tmpfs/shmem

You can control hugepage allocation policy in tmpfs with mount option `huge=`. It can have following values:

**always** Attempt to allocate huge pages every time we need a new page;

**never** Do not allocate huge pages;

**within\_size** Only allocate huge page if it will be fully within `i_size`. Also respect `fadvise()/madvise()` hints;

**advise** Only allocate huge pages if requested with `fadvise()/madvise()`;

The default policy is never.

`mount -o remount,huge= /mountpoint` works fine after `mount: remounting huge=never` will not attempt to break up huge pages at all, just stop more from being allocated.

There's also sysfs knob to control hugepage allocation policy for internal shmem mount: `/sys/kernel/mm/transparent_hugepage/shmem_enabled`. The mount is used for SysV SHM, memfds, shared anonymous mmmaps (of `/dev/zero` or `MAP_ANONYMOUS`), GPU drivers' DRM objects, Ashmem.

In addition to policies listed above, `shmem_enabled` allows two further values:

**deny** For use in emergencies, to force the huge option off from all mounts;

**force** Force the huge option on for all - very useful for testing;

#### 54.14.5 Need of application restart

The `transparent_hugepage/enabled` values and `tmpfs` mount option only affect future behavior. So to make them effective you need to restart any application that could have been using hugepages. This also applies to the regions registered in `khugepaged`.

#### 54.14.6 Monitoring usage

The number of anonymous transparent huge pages currently used by the system is available by reading the `AnonHugePages` field in `/proc/meminfo`. To identify what applications are using anonymous transparent huge pages, it is necessary to read `/proc/PID/smmaps` and count the `AnonHugePages` fields for each mapping.

The number of file transparent huge pages mapped to userspace is available by reading `ShmemPmdMapped` and `ShmemHugePages` fields in `/proc/meminfo`. To identify what applications are mapping file transparent huge pages, it is necessary to read `/proc/PID/smmaps` and count the `FileHugeMapped` fields for each mapping.

Note that reading the `smmaps` file is expensive and reading it frequently will incur overhead.

There are a number of counters in `/proc/vmstat` that may be used to monitor how successfully the system is providing huge pages for use.

**thp\_fault\_alloc** is incremented every time a huge page is successfully allocated to handle a page fault.

**thp\_collapse\_alloc** is incremented by `khugepaged` when it has found a range of pages to collapse into one huge page and has successfully allocated a new huge page to store the data.

**thp\_fault\_fallback** is incremented if a page fault fails to allocate a huge page and instead falls back to using small pages.

**thp\_fault\_fallback\_charge** is incremented if a page fault fails to charge a huge page and instead falls back to using small pages even though the allocation was successful.

**thp\_collapse\_alloc\_failed** is incremented if khugepaged found a range of pages that should be collapsed into one huge page but failed the allocation.

**thp\_file\_alloc** is incremented every time a file huge page is successfully allocated.

**thp\_file\_fallback** is incremented if a file huge page is attempted to be allocated but fails and instead falls back to using small pages.

**thp\_file\_fallback\_charge** is incremented if a file huge page cannot be charged and instead falls back to using small pages even though the allocation was successful.

**thp\_file\_mapped** is incremented every time a file huge page is mapped into user address space.

**thp\_split\_page** is incremented every time a huge page is split into base pages. This can happen for a variety of reasons but a common reason is that a huge page is old and is being reclaimed. This action implies splitting all PMD the page mapped with.

**thp\_split\_page\_failed** is incremented if kernel fails to split huge page. This can happen if the page was pinned by somebody.

**thp\_deferred\_split\_page** is incremented when a huge page is put onto split queue. This happens when a huge page is partially unmapped and splitting it would free up some memory. Pages on split queue are going to be split under memory pressure.

**thp\_split\_pmd** is incremented every time a PMD split into table of PTEs. This can happen, for instance, when application calls `mprotect()` or `munmap()` on part of huge page. It doesn't split huge page, only page table entry.

**thp\_zero\_page\_alloc** is incremented every time a huge zero page is successfully allocated. It includes allocations which were dropped due race with other allocation. Note, it doesn't count every map of the huge zero page, only its allocation.

**thp\_zero\_page\_alloc\_failed** is incremented if kernel fails to allocate huge zero page and falls back to using small pages.

**thp\_swpage** is incremented every time a huge page is swapped in one piece without splitting.

**thp\_swpage\_fallback** is incremented if a huge page has to be split before swapout. Usually because failed to allocate some continuous swap space for the huge page.

As the system ages, allocating huge pages may be expensive as the system uses memory compaction to copy data around memory to free a huge page for use. There are some counters in `/proc/vmstat` to help monitor this overhead.

**compact\_stall** is incremented every time a process stalls to run memory compaction so that a huge page is free for use.

**compact\_success** is incremented if the system compacted memory and freed a huge page for use.

**compact\_fail** is incremented if the system tries to compact memory but failed.

**compact\_pages\_moved** is incremented each time a page is moved. If this value is increasing rapidly, it implies that the system is copying a lot of data to satisfy the huge page allocation. It is possible that the cost of copying exceeds any savings from reduced TLB misses.

**compact\_pagemigrate\_failed** is incremented when the underlying mechanism for moving a page failed.

**compact\_blocks\_moved** is incremented each time memory compaction examines a huge page aligned range of pages.

It is possible to establish how long the stalls were using the function tracer to record how long was spent in `__alloc_pages_nodemask` and using the `mm_page_alloc` tracepoint to identify which allocations were for huge pages.

### 54.14.7 Optimizing the applications

To be guaranteed that the kernel will map a 2M page immediately in any memory region, the `mmap` region has to be hugepage naturally aligned. `posix_memalign()` can provide that guarantee.

### 54.14.8 Hugetlbfs

You can use hugetlbfs on a kernel that has transparent hugepage support enabled just fine as always. No difference can be noted in hugetlbfs other than there will be less overall fragmentation. All usual features belonging to hugetlbfs are preserved and unaffected. `libhugetlbfs` will also work fine as usual.

## 54.15 Userfaultfd

### 54.15.1 Objective

Userfaults allow the implementation of on-demand paging from userland and more generally they allow userland to take control of various memory page faults, something otherwise only the kernel code could do.

For example userfaults allows a proper and more optimal implementation of the `PROT_NONE+SIGSEGV` trick.

### 54.15.2 Design

Userfaults are delivered and resolved through the `userfaultfd` syscall.

The `userfaultfd` (aside from registering and unregistering virtual memory ranges) provides two primary functionalities:

- 1) `read/POLLIN` protocol to notify a userland thread of the faults happening
- 2) various `UFFDIO_*` ioctls that can manage the virtual memory regions registered in the `userfaultfd` that allows userland to efficiently resolve the userfaults it receives via 1) or to manage the virtual memory in the background

The real advantage of userfaults if compared to regular virtual memory management of `mremap/mprotect` is that the userfaults in all their operations never involve heavyweight structures like `vmas` (in fact the `userfaultfd` runtime load never takes the `mmap_lock` for writing).

`Vmas` are not suitable for page- (or hugepage) granular fault tracking when dealing with virtual address spaces that could span Terabytes. Too many `vmas` would be needed for that.

The `userfaultfd` once opened by invoking the syscall, can also be passed using unix domain sockets to a manager process, so the same manager process could handle the userfaults of a multitude of different processes without them being aware about what is going on (well of course unless they later try to use the `userfaultfd` themselves on the same region the manager is already tracking, which is a corner case that would currently return `-EBUSY`).

### 54.15.3 API

When first opened the `userfaultfd` must be enabled invoking the `UFFDIO_API` ioctl specifying a `uffdio_api.api` value set to `UFFD_API` (or a later API version) which will specify the `read/POLLIN` protocol userland intends to speak on the UFFD and the `uffdio_api.features` userland requires. The `UFFDIO_API` ioctl if successful (i.e. if the requested `uffdio_api.api` is spoken also by the running kernel and the requested features are going to be enabled) will return into `uffdio_api.features` and `uffdio_api.ioctls` two 64bit bitmasks of respectively all the available features of the `read(2)` protocol and the generic ioctl available.

The `uffdio_api.features` bitmask returned by the `UFFDIO_API` ioctl defines what memory types are supported by the `userfaultfd` and what events, except page fault notifications, may be generated.

If the kernel supports registering `userfaultfd` ranges on `hugetlbfs` virtual memory areas, `UFFD_FEATURE_MISSING_HUGETLBFS` will be set in `uffdio_api.features`. Similarly, `UFFD_FEATURE_MISSING_SHMEM` will be set if the kernel supports registering `userfaultfd` ranges on shared memory (covering all `shmem` APIs, i.e. `tmpfs`, `IPCshm`, `/dev/zero`, `MAP_SHARED`, `memfd_create`, etc).

The userland application that wants to use `userfaultfd` with `hugetlbfs` or shared memory need to set the corresponding flag in `uffdio_api.features` to enable those features.

If the userland desires to receive notifications for events other than page faults, it has to verify that `uffdio_api.features` has appropriate `UFFD_FEATURE_EVENT_*` bits set. These events are described in more detail below in `Non-cooperative userfaultfd` section.

Once the `userfaultfd` has been enabled the `UFFDIO_REGISTER` ioctl should be invoked (if present in the returned `uffdio_api.ioctls` bitmask) to register a memory range in the `userfaultfd` by setting the `uffdio_register` structure accordingly. The `uffdio_register.mode` bitmask will specify to the kernel which kind of faults to track for the range (`UFFDIO_REGISTER_MODE_MISSING` would track missing pages). The `UFFDIO_REGISTER` ioctl will return the `uffdio_register.ioctls` bitmask of ioctls that are suitable to resolve userfaults on the range registered. Not

all ioctls will necessarily be supported for all memory types depending on the underlying virtual memory backend (anonymous memory vs tmpfs vs real filebacked mappings).

Userland can use the `uffdio_register.ioctls` to manage the virtual address space in the background (to add or potentially also remove memory from the `userfaultfd` registered range). This means a userfault could be triggering just before userland maps in the background the user-faulted page.

The primary ioctl to resolve userfaults is `UFFDIO_COPY`. That atomically copies a page into the userfault registered range and wakes up the blocked userfaults (unless `uffdio_copy.mode` & `UFFDIO_COPY_MODE_DONTWAKE` is set). Other ioctls work similarly to `UFFDIO_COPY`. They're atomic as in guaranteeing that nothing can see a half copied page since it'll keep userfaulting until the copy has finished.

Notes:

- If you requested `UFFDIO_REGISTER_MODE_MISSING` when registering then you must provide some kind of page in your thread after reading from the `uffd`. You must provide either `UFFDIO_COPY` or `UFFDIO_ZEROPAGE`. The normal behavior of the OS automatically providing a zero page on an anonymous mmaping is not in place.
- None of the page-delivering ioctls default to the range that you registered with. You must fill in all fields for the appropriate ioctl struct including the range.
- You get the address of the access that triggered the missing page event out of a struct `uffd_msg` that you read in the thread from the `uffd`. You can supply as many pages as you want with `UFFDIO_COPY` or `UFFDIO_ZEROPAGE`. Keep in mind that unless you used `DONTWAKE` then the first of any of those IOCTLS wakes up the faulting thread.
- Be sure to test for all errors including (`pollfd[0].revents` & `POLLERR`). This can happen, e.g. when ranges supplied were incorrect.

## Write Protect Notifications

This is equivalent to (but faster than) using `mprotect` and a `SIGSEGV` signal handler.

Firstly you need to register a range with `UFFDIO_REGISTER_MODE_WP`. Instead of using `mprotect(2)` you use `ioctl(uffd, UFFDIO_WRITEPROTECT, struct *uffdio_writeprotect)` while `mode = UFFDIO_WRITEPROTECT_MODE_WP` in the struct passed in. The range does not default to and does not have to be identical to the range you registered with. You can write protect as many ranges as you like (inside the registered range). Then, in the thread reading from `uffd` the struct will have `msg.arg.pagefault.flags` & `UFFD_PAGEFAULT_FLAG_WP` set. Now you send `ioctl(uffd, UFFDIO_WRITEPROTECT, struct *uffdio_writeprotect)` again while `pagefault.mode` does not have `UFFDIO_WRITEPROTECT_MODE_WP` set. This wakes up the thread which will continue to run with writes. This allows you to do the bookkeeping about the write in the `uffd` reading thread before the ioctl.

If you registered with both `UFFDIO_REGISTER_MODE_MISSING` and `UFFDIO_REGISTER_MODE_WP` then you need to think about the sequence in

which you supply a page and undo write protect. Note that there is a difference between writes into a WP area and into a !WP area. The former will have `UFFD_PAGEFAULT_FLAG_WP` set, the latter `UFFD_PAGEFAULT_FLAG_WRITE`. The latter did not fail on protection but you still need to supply a page when `UFFDIO_REGISTER_MODE_MISSING` was used.

### 54.15.4 QEMU/KVM

QEMU/KVM is using the `userfaultfd` syscall to implement postcopy live migration. Postcopy live migration is one form of memory externalization consisting of a virtual machine running with part or all of its memory residing on a different node in the cloud. The `userfaultfd` abstraction is generic enough that not a single line of KVM kernel code had to be modified in order to add postcopy live migration to QEMU.

Guest async page faults, `FOLL_NOWAIT` and all other GUP\* features work just fine in combination with userfaults. Userfaults trigger async page faults in the guest scheduler so those guest processes that aren't waiting for userfaults (i.e. network bound) can keep running in the guest vcpu.

It is generally beneficial to run one pass of precopy live migration just before starting postcopy live migration, in order to avoid generating userfaults for readonly guest regions.

The implementation of postcopy live migration currently uses one single bidirectional socket but in the future two different sockets will be used (to reduce the latency of the userfaults to the minimum possible without having to decrease /proc/sys/net/ipv4/tcp\_wmem).

The QEMU in the source node writes all pages that it knows are missing in the destination node, into the socket, and the migration thread of the QEMU running in the destination node runs `UFFDIO_COPY|ZEROPAGE` ioctls on the `userfaultfd` in order to map the received pages into the guest (`UFFDIO_ZEROCOPY` is used if the source page was a zero page).

A different postcopy thread in the destination node listens with `poll()` to the `userfaultfd` in parallel. When a `POLLIN` event is generated after a userfault triggers, the postcopy thread `read()` from the `userfaultfd` and receives the fault address (or `-EAGAIN` in case the userfault was already resolved and waken by a `UFFDIO_COPY|ZEROPAGE` run by the parallel QEMU migration thread).

After the QEMU postcopy thread (running in the destination node) gets the userfault address it writes the information about the missing page into the socket. The QEMU source node receives the information and roughly “seeks” to that page address and continues sending all remaining missing pages from that new page offset. Soon after that (just the time to flush the `tcp_wmem` queue through the network) the migration thread in the QEMU running in the destination node will receive the page that triggered the userfault and it'll map it as usual with the `UFFDIO_COPY|ZEROPAGE` (without actually knowing if it was spontaneously sent by the source or if it was an urgent page requested through a userfault).

By the time the userfaults start, the QEMU in the destination node doesn't need to keep any per-page state bitmap relative to the live migration around and a single per-page bitmap has to be maintained in the QEMU running in the source node

to know which pages are still missing in the destination node. The bitmap in the source node is checked to find which missing pages to send in round robin and we seek over it when receiving incoming userfaults. After sending each page of course the bitmap is updated accordingly. It's also useful to avoid sending the same page twice (in case the userfault is read by the postcopy thread just before UFFDIO\_COPY|ZEROPAGE runs in the migration thread).

### 54.15.5 Non-cooperative userfaultfd

When the `userfaultfd` is monitored by an external manager, the manager must be able to track changes in the process virtual memory layout. `Userfaultfd` can notify the manager about such changes using the same `read(2)` protocol as for the page fault notifications. The manager has to explicitly enable these events by setting appropriate bits in `uffdio_api.features` passed to `UFFDIO_API` `ioctl`:

**UFFD\_FEATURE\_EVENT\_FORK** enable `userfaultfd` hooks for `fork()`. When this feature is enabled, the `userfaultfd` context of the parent process is duplicated into the newly created process. The manager receives `UFFD_EVENT_FORK` with file descriptor of the new `userfaultfd` context in the `uffd_msg.fork`.

**UFFD\_FEATURE\_EVENT\_REMAP** enable notifications about `mremap()` calls. When the non-cooperative process moves a virtual memory area to a different location, the manager will receive `UFFD_EVENT_REMAP`. The `uffd_msg.remap` will contain the old and new addresses of the area and its original length.

**UFFD\_FEATURE\_EVENT\_REMOVE** enable notifications about `madvise(MADV_REMOVE)` and `madvise(MADV_DONTNEED)` calls. The event `UFFD_EVENT_REMOVE` will be generated upon these calls to `madvise()`. The `uffd_msg.remove` will contain start and end addresses of the removed area.

**UFFD\_FEATURE\_EVENT\_UNMAP** enable notifications about memory unmapping. The manager will get `UFFD_EVENT_UNMAP` with `uffd_msg.remove` containing start and end addresses of the unmapped area.

Although the `UFFD_FEATURE_EVENT_REMOVE` and `UFFD_FEATURE_EVENT_UNMAP` are pretty similar, they quite differ in the action expected from the `userfaultfd` manager. In the former case, the virtual memory is removed, but the area is not, the area remains monitored by the `userfaultfd`, and if a page fault occurs in that area it will be delivered to the manager. The proper resolution for such page fault is to `zeromap` the faulting address. However, in the latter case, when an area is unmapped, either explicitly (with `munmap()` system call), or implicitly (e.g. during `mremap()`), the area is removed and in turn the `userfaultfd` context for such area disappears too and the manager will not get further userland page faults from the removed area. Still, the notification is required in order to prevent manager from using `UFFDIO_COPY` on the unmapped area.

Unlike userland page faults which have to be synchronous and require explicit or implicit wakeup, all the events are delivered asynchronously and the non-cooperative process resumes execution as soon as manager executes `read()`. The `userfaultfd` manager should carefully synchronize calls to `UFFDIO_COPY` with the events processing. To aid the synchronization, the `UFFDIO_COPY` `ioctl` will return `-ENOSPC` when the monitored process exits at the time of `UFFDIO_COPY`, and `-ENOENT`, when the non-cooperative process has changed its virtual memory layout simultaneously with outstanding `UFFDIO_COPY` operation.

The current asynchronous model of the event delivery is optimal for single threaded non-cooperative `userfaultfd` manager implementations. A synchronous event delivery model can be added later as a new `userfaultfd` feature to facilitate multithreading enhancements of the non cooperative manager, for example to allow `UFFDIO_COPY` ioctls to run in parallel to the event reception. Single threaded implementations should continue to use the current async event delivery model instead.

## **KERNEL MODULE SIGNING FACILITY**

### **55.1 Overview**

The kernel module signing facility cryptographically signs modules during installation and then checks the signature upon loading the module. This allows increased kernel security by disallowing the loading of unsigned modules or modules signed with an invalid key. Module signing increases security by making it harder to load a malicious module into the kernel. The module signature checking is done by the kernel so that it is not necessary to have trusted userspace bits.

This facility uses X.509 ITU-T standard certificates to encode the public keys involved. The signatures are not themselves encoded in any industrial standard type. The facility currently only supports the RSA public key encryption standard (though it is pluggable and permits others to be used). The possible hash algorithms that can be used are SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512 (the algorithm is selected by data in the signature).

### **55.2 Configuring module signing**

The module signing facility is enabled by going to the Enable Loadable Module Support section of the kernel configuration and turning on:

<code>CONFIG_MODULE_SIG</code>	"Module signature verification"
--------------------------------	---------------------------------

This has a number of options available:

- (1) Require modules to be validly signed (`CONFIG_MODULE_SIG_FORCE`)

This specifies how the kernel should deal with a module that has a signature for which the key is not known or a module that is unsigned.

If this is off (ie. “permissive”), then modules for which the key is not available and modules that are unsigned are permitted, but the kernel will be marked as being tainted, and the concerned modules will be marked as tainted, shown with the character ‘E’ .

If this is on (ie. “restrictive” ), only modules that have a valid signature that can be verified by a public key in the kernel’s possession will be loaded. All other modules will generate an error.

Irrespective of the setting here, if the module has a signature block that cannot be parsed, it will be rejected out of hand.

(2) Automatically sign all modules (CONFIG\_MODULE\_SIG\_ALL)

If this is on then modules will be automatically signed during the `modules_install` phase of a build. If this is off, then the modules must be signed manually using:

```
scripts/sign-file
```

(3) Which hash algorithm should modules be signed with?

This presents a choice of which hash algorithm the installation phase will sign the modules with:

CONFIG_MODULE_SIG_SHA1	Sign modules with SHA-1
CONFIG_MODULE_SIG_SHA224	Sign modules with SHA-224
CONFIG_MODULE_SIG_SHA256	Sign modules with SHA-256
CONFIG_MODULE_SIG_SHA384	Sign modules with SHA-384
CONFIG_MODULE_SIG_SHA512	Sign modules with SHA-512

The algorithm selected here will also be built into the kernel (rather than being a module) so that modules signed with that algorithm can have their signatures checked without causing a dependency loop.

(4) File name or PKCS#11 URI of module signing key (CONFIG\_MODULE\_SIG\_KEY)

Setting this option to something other than its default of `certs/signing_key.pem` will disable the autogeneration of signing keys and allow the kernel modules to be signed with a key of your choosing. The string provided should identify a file containing both a private key and its corresponding X.509 certificate in PEM form, or —on systems where the OpenSSL `ENGINE_pkcs11` is functional —a PKCS#11 URI as defined by RFC7512. In the latter case, the PKCS#11 URI should reference both a certificate and a private key.

If the PEM file containing the private key is encrypted, or if the PKCS#11 token requires a PIN, this can be provided at build time by means of the `KBUILD_SIGN_PIN` variable.

(5) Additional X.509 keys for default system keyring (CONFIG\_SYSTEM\_TRUSTED\_KEYS)

This option can be set to the filename of a PEM-encoded file containing additional certificates which will be included in the system keyring by default.

Note that enabling module signing adds a dependency on the OpenSSL devel packages to the kernel build processes for the tool that does the signing.

## 55.3 Generating signing keys

Cryptographic keypairs are required to generate and check signatures. A private key is used to generate a signature and the corresponding public key is used to check it. The private key is only needed during the build, after which it can be deleted or stored securely. The public key gets built into the kernel so that it can be used to check the signatures as the modules are loaded.

Under normal conditions, when `CONFIG_MODULE_SIG_KEY` is unchanged from its default, the kernel build will automatically generate a new keypair using `openssl` if one does not exist in the file:

```
certs/signing_key.pem
```

during the building of `vmlinux` (the public part of the key needs to be built into `vmlinux`) using parameters in the:

```
certs/x509.genkey
```

file (which is also generated if it does not already exist).

It is strongly recommended that you provide your own `x509.genkey` file.

Most notably, in the `x509.genkey` file, the `req_distinguished_name` section should be altered from the default:

```
[ req_distinguished_name ]
#0 = Unspecified company
CN = Build time autogenerated kernel key
#emailAddress = unspecified.user@unspecified.company
```

The generated RSA key size can also be set with:

```
[ req ]
default_bits = 4096
```

It is also possible to manually generate the key private/public files using the `x509.genkey` key generation configuration file in the root node of the Linux kernel sources tree and the `openssl` command. The following is an example to generate the public/private key files:

```
openssl req -new -nodes -utf8 -sha256 -days 36500 -batch -x509 \
  -config x509.genkey -outform PEM -out kernel_key.pem \
  -keyout kernel_key.pem
```

The full pathname for the resulting `kernel_key.pem` file can then be specified in the `CONFIG_MODULE_SIG_KEY` option, and the certificate and key therein will be used instead of an autogenerated keypair.

## 55.4 Public keys in the kernel

The kernel contains a ring of public keys that can be viewed by root. They're in a keyring called ".builtin\_trusted\_keys" that can be seen by:

```
[root@deneb ~]# cat /proc/keys
...
223c7853 I-----      1 perm 1f030000      0      0 keyring  .builtin_
↳trusted_keys: 1
302d2d52 I-----      1 perm 1f010000      0      0 asymmetri Fedora kernel
↳signing key: d69a84e6bce3d216b979e9505b3e3ef9a7118079: X509.RSA a7118079
↳[]
...
```

Beyond the public key generated specifically for module signing, additional trusted certificates can be provided in a PEM-encoded file referenced by the CONFIG\_SYSTEM\_TRUSTED\_KEYS configuration option.

Further, the architecture code may take public keys from a hardware store and add those in also (e.g. from the UEFI key database).

Finally, it is possible to add additional public keys by doing:

```
keyctl padd asymmetric "" [.builtin_trusted_keys-ID] <[key-file]
```

e.g.:

```
keyctl padd asymmetric "" 0x223c7853 <my_public_key.x509
```

Note, however, that the kernel will only permit keys to be added to .builtin\_trusted\_keys **if** the new key's X.509 wrapper is validly signed by a key that is already resident in the .builtin\_trusted\_keys at the time the key was added.

## 55.5 Manually signing modules

To manually sign a module, use the scripts/sign-file tool available in the Linux kernel source tree. The script requires 4 arguments:

1. The hash algorithm (e.g., sha256)
2. The private key filename or PKCS#11 URI
3. The public key filename
4. The kernel module to be signed

The following is an example to sign a kernel module:

```
scripts/sign-file sha512 kernel-signkey.priv \
kernel-signkey.x509 module.ko
```

The hash algorithm used does not have to match the one configured, but if it doesn't, you should make sure that hash algorithm is either built into the kernel or can be loaded without requiring itself.

If the private key requires a passphrase or PIN, it can be provided in the `$KBUILD_SIGN_PIN` environment variable.

## 55.6 Signed modules and stripping

A signed module has a digital signature simply appended at the end. The string `~Module signature appended~.` at the end of the module's file confirms that a signature is present but it does not confirm that the signature is valid!

Signed modules are BRITTLE as the signature is outside of the defined ELF container. Thus they MAY NOT be stripped once the signature is computed and attached. Note the entire module is the signed payload, including any and all debug information present at the time of signing.

## 55.7 Loading signed modules

Modules are loaded with `insmod`, `modprobe`, `init_module()` or `finit_module()`, exactly as for unsigned modules as no processing is done in userspace. The signature checking is all done within the kernel.

## 55.8 Non-valid signatures and unsigned modules

If `CONFIG_MODULE_SIG_FORCE` is enabled or `module.sig_enforce=1` is supplied on the kernel command line, the kernel will only load validly signed modules for which it has a public key. Otherwise, it will also load modules that are unsigned. Any module for which the kernel has a key, but which proves to have a signature mismatch will not be permitted to load.

Any module that has an unparseable signature will be rejected.

## 55.9 Administering/protecting the private key

Since the private key is used to sign modules, viruses and malware could use the private key to sign modules and compromise the operating system. The private key must be either destroyed or moved to a secure location and not kept in the root node of the kernel source tree.

If you use the same private key to sign modules for multiple kernel configurations, you must ensure that the module version information is sufficient to prevent loading a module into a different kernel. Either set `CONFIG_MODVERSIONS=y` or ensure that each configuration has a different kernel release string by changing `EXTRAVERSION` or `CONFIG_LOCALVERSION`.



## **MONO(TM) BINARY KERNEL SUPPORT FOR LINUX**

To configure Linux to automatically execute Mono-based .NET binaries (in the form of .exe files) without the need to use the mono CLR wrapper, you can use the BINFMT\_MISC kernel support.

This will allow you to execute Mono-based .NET binaries just like any other program after you have done the following:

- 1) You **MUST FIRST** install the Mono CLR support, either by downloading a binary package, a source tarball or by installing from Git. Binary packages for several distributions can be found at:

<https://www.mono-project.com/download/>

Instructions for compiling Mono can be found at:

<https://www.mono-project.com/docs/compiling-mono/linux/>

Once the Mono CLR support has been installed, just check that `/usr/bin/mono` (which could be located elsewhere, for example `/usr/local/bin/mono`) is working.

- 2) You have to compile BINFMT\_MISC either as a module or into the kernel (CONFIG\_BINFMT\_MISC) and set it up properly. If you choose to compile it as a module, you will have to insert it manually with `modprobe/inmod`, as `kmod` cannot be easily supported with `binfmt_misc`. Read the file `binfmt_misc.txt` in this directory to know more about the configuration process.
- 3) Add the following entries to `/etc/rc.local` or similar script to be run at system startup:

```
# Insert BINFMT_MISC module into the kernel
if [ ! -e /proc/sys/fs/binfmt_misc/register ]; then
  /sbin/modprobe binfmt_misc
  # Some distributions, like Fedora Core, perform
  # the following command automatically when the
  # binfmt_misc module is loaded into the kernel
  # or during normal boot up (systemd-based systems).
  # Thus, it is possible that the following line
  # is not needed at all.
  mount -t binfmt_misc none /proc/sys/fs/binfmt_misc
fi

# Register support for .NET CLR binaries
if [ -e /proc/sys/fs/binfmt_misc/register ]; then
```

(continues on next page)

(continued from previous page)

```
# Replace /usr/bin/mono with the correct pathname to
# the Mono CLR runtime (usually /usr/local/bin/mono
# when compiling from sources or CVS).
echo ':CLR:M::MZ::/usr/bin/mono:' > /proc/sys/fs/binfmt_misc/
↔register
else
  echo "No binfmt_misc support"
  exit 1
fi
```

- 4) Check that .exe binaries can be ran without the need of a wrapper script, simply by launching the .exe file directly from a command prompt, for example:

```
/usr/bin/xsd.exe
```

---

**Note:** If this fails with a permission denied error, check that the .exe file has execute permissions.

---

## **NAMESPACES**

### **57.1 Namespaces compatibility list**

This document contains the information about the problems user may have when creating tasks living in different namespaces.

Here' s the summary. This matrix shows the known problems, that occur when tasks share some namespace (the columns) while living in different other namespaces (the rows):

.	UTS	IPC	VFS	PID	User	Net
UTS	X					
IPC		X	1			
VFS			X			
PID		1	1	X		
User		2	2		X	
Net						X

1. Both the IPC and the PID namespaces provide IDs to address object inside the kernel. E.g. semaphore with IPCID or process group with pid.

In both cases, tasks shouldn' t try exposing this ID to some other task living in a different namespace via a shared filesystem or IPC shmем/message. The fact is that this ID is only valid within the namespace it was obtained in and may refer to some other object in another namespace.

2. Intentionally, two equal user IDs in different user namespaces should not be equal from the VFS point of view. In other words, user 10 in one user namespace shouldn' t have the same access permissions to files, belonging to user 10 in another namespace.

The same is true for the IPC namespaces being shared - two users from different user namespaces should not access the same IPC objects even having equal UIDs.

But currently this is not so.

## 57.2 Namespaces research control

There are a lot of kinds of objects in the kernel that don't have individual limits or that have limits that are ineffective when a set of processes is allowed to switch user ids. With user namespaces enabled in a kernel for people who don't trust their users or their users programs to play nice this problems becomes more acute.

Therefore it is recommended that memory control groups be enabled in kernels that enable user namespaces, and it is further recommended that userspace configure memory control groups to limit how much memory user's they don't trust to play nice can use.

Memory control groups can be configured by installing the libcgroup package present on most distros editing `/etc/cgrules.conf`, `/etc/cgconfig.conf` and setting up `libpam-cgroup`.

## **NUMA POLICY HIT/MISS STATISTICS**

`/sys/devices/system/node/node*/numastat`

All units are pages. Hugepages have separate counters.

The `numa_hit`, `numa_miss` and `numa_foreign` counters reflect how well processes are able to allocate memory from nodes they prefer. If they succeed, `numa_hit` is incremented on the preferred node, otherwise `numa_foreign` is incremented on the preferred node and `numa_miss` on the node where allocation succeeded.

Usually preferred node is the one local to the CPU where the process executes, but restrictions such as mempolicies can change that, so there are also two counters based on CPU local node. `local_node` is similar to `numa_hit` and is incremented on allocation from a node by CPU on the same node. `other_node` is similar to `numa_miss` and is incremented on the node where allocation succeeds from a CPU from a different node. Note there is no counter analogical to `numa_foreign`.

In more detail:

<code>numa_hit</code>	A process wanted to allocate memory from this node, and succeeded.
<code>numa_miss</code>	A process wanted to allocate memory from another node, but ended up with memory from this node.
<code>numa_foreign</code>	A process wanted to allocate on this node, but ended up with memory from another node.
<code>local_node</code>	A process ran on this node's CPU, and got memory from this node.
<code>other_node</code>	A process ran on a different node's CPU and got memory from this node.
<code>interleave_hit</code>	Interleaving wanted to allocate from this node and succeeded.

For easier reading you can use the `numastat` utility from the `numactl` package (<http://oss.sgi.com/projects/libnuma/>). Note that it only works well right now on machines with a small number of CPUs.

Note that on systems with memoryless nodes (where a node has CPUs but no memory) the `numa_hit`, `numa_miss` and `numa_foreign` statistics can be skewed heavily. In the current kernel implementation, if a process prefers a memoryless node (i.e. because it is running on one of its local CPU), the implementation actually treats one of the nearest nodes with memory as the preferred node. As a result, such allocation will not increase the `numa_foreign` counter on the memoryless node,

and will skew the `numa_hit`, `numa_miss` and `numa_foreign` statistics of the nearest node.

## **PARPORT**

The `parport` code provides parallel-port support under Linux. This includes the ability to share one port between multiple device drivers.

You can pass parameters to the `parport` code to override its automatic detection of your hardware. This is particularly useful if you want to use IRQs, since in general these can't be autoprobeed successfully. By default IRQs are not used even if they **can** be probed. This is because there are a lot of people using the same IRQ for their parallel port and a sound card or network card.

The `parport` code is split into two parts: generic (which deals with port-sharing) and architecture-dependent (which deals with actually using the port).

### **59.1 Parport as modules**

If you load the `parport`` code as a module, say:

```
# insmod parport
```

to load the generic `parport` code. You then must load the architecture-dependent code with (for example):

```
# insmod parport_pc io=0x3bc,0x378,0x278 irq=none,7,auto
```

to tell the `parport` code that you want three PC-style ports, one at 0x3bc with no IRQ, one at 0x378 using IRQ 7, and one at 0x278 with an auto-detected IRQ. Currently, PC-style (`parport_pc`), Sun `bpp`, Amiga, Atari, and MFC3 hardware is supported.

PCI parallel I/O card support comes from `parport_pc`. Base I/O addresses should not be specified for supported PCI cards since they are automatically detected.

### 59.1.1 modprobe

If you use `modprobe`, you will find it useful to add lines as below to a configuration file in `/etc/modprobe.d/` directory:

```
alias parport_lowlevel parport_pc
options parport_pc io=0x378,0x278 irq=7,auto
```

`modprobe` will load `parport_pc` (with the options `io=0x378,0x278 irq=7,auto`) whenever a parallel port device driver (such as `lp`) is loaded.

Note that these are example lines only! You shouldn't in general need to specify any options to `parport_pc` in order to be able to use a parallel port.

### 59.1.2 Parport probe [optional]

In 2.2 kernels there was a module called `parport_probe`, which was used for collecting IEEE 1284 device ID information. This has now been enhanced and now lives with the IEEE 1284 support. When a parallel port is detected, the devices that are connected to it are analysed, and information is logged like this:

```
parport0: Printer, BJC-210 (Canon)
```

The probe information is available from files in `/proc/sys/dev/parport/`.

## 59.2 Parport linked into the kernel statically

If you compile the `parport` code into the kernel, then you can use kernel boot parameters to get the same effect. Add something like the following to your LILO command line:

```
parport=0x3bc parport=0x378,7 parport=0x278,auto,nofifo
```

You can have many `parport=...` statements, one for each port you want to add. Adding `parport=0` to the kernel command-line will disable `parport` support entirely. Adding `parport=auto` to the kernel command-line will make `parport` use any IRQ lines or DMA channels that it auto-detects.

### 59.3 Files in /proc

If you have configured the `/proc` filesystem into your kernel, you will see a new directory entry: `/proc/sys/dev/parport`. In there will be a directory entry for each parallel port for which `parport` is configured. In each of those directories are a collection of files describing that parallel port.

The `/proc/sys/dev/parport` directory tree looks like:

```
parport
|-- default
|  |-- spintime
```

(continues on next page)

(continued from previous page)

```
|  `-- timeslice
|-- parport0
|   |-- autoprobe
|   |-- autoprobe0
|   |-- autoprobe1
|   |-- autoprobe2
|   |-- autoprobe3
|   |-- devices
|   |  |-- active
|   |  `-- lp
|   |     `-- timeslice
|   |-- base-addr
|   |-- irq
|   |-- dma
|   |-- modes
|   `-- spintime
|-- parport1
|-- autoprobe
|-- autoprobe0
|-- autoprobe1
|-- autoprobe2
|-- autoprobe3
|-- devices
|   |-- active
|   `-- ppa
|      `-- timeslice
|-- base-addr
|-- irq
|-- dma
|-- modes
`-- spintime
```

File	Contents
devices/active	A list of the device drivers using that port. A “+” will appear by the name of the device currently using the port (it might not appear against any). The string “none” means that there are no device drivers using that port.
base-addr	Parallel port’ s base address, or addresses if the port has more than one in which case they are separated with tabs. These values might not have any sensible meaning for some ports.
irq	Parallel port’ s IRQ, or -1 if none is being used.
dma	Parallel port’ s DMA channel, or -1 if none is being used.
modes	Parallel port’ s hardware modes, comma-separated, meaning: <ul style="list-style-type: none"> <li>• <b>PCSPP</b> PC-style SPP registers are available.</li> <li>• <b>TRISTATE</b> Port is bidirectional.</li> <li>• <b>COMPAT</b> Hardware acceleration for printers is available and will be used.</li> <li>• <b>EPP</b> Hardware acceleration for EPP protocol is available and will be used.</li> <li>• <b>ECP</b> Hardware acceleration for ECP protocol is available and will be used.</li> <li>• <b>DMA</b> DMA is available and will be used.</li> </ul> Note that the current implementation will only take advantage of COMPAT and ECP modes if it has an IRQ line to use.
autoprobe	Any IEEE-1284 device ID information that has been acquired from the (non-IEEE 1284.3) device.
autoprobe[0-3]	IEEE 1284 device ID information retrieved from daisy-chain devices that conform to IEEE 1284.3.
spintime	The number of microseconds to busy-loop while waiting for the peripheral to respond. You might find that adjusting this improves performance, depending on your peripherals. This is a port-wide setting, i.e. it applies to all devices on a particular port.
timeslice	The number of milliseconds that a device driver is allowed to keep a port claimed for. This is advisory, and driver can ignore it if it must.
default/*	The defaults for spintime and timeslice. When a new port is registered, it picks up the default spintime. When a new device is registered, it picks up the default timeslice.

## 59.4 Device drivers

Once the parport code is initialised, you can attach device drivers to specific ports. Normally this happens automatically; if the lp driver is loaded it will create one lp device for each port found. You can override this, though, by using parameters either when you load the lp driver:

```
# insmod lp parport=0,2
```

or on the LILO command line:

```
lp=parport0 lp=parport2
```

Both the above examples would inform lp that you want /dev/lp0 to be the first parallel port, and /dev/lp1 to be the **third** parallel port, with no lp device associated with the second port (parport1). Note that this is different to the way older kernels worked; there used to be a static association between the I/O port address and the device name, so /dev/lp0 was always the port at 0x3bc. This is no longer the case - if you only have one port, it will default to being /dev/lp0, regardless of base address.

Also:

- If you selected the IEEE 1284 support at compile time, you can say lp=auto on the kernel command line, and lp will create devices only for those ports that seem to have printers attached.
- If you give PLIP the timid parameter, either with plip=timid on the command line, or with insmod plip timid=1 when using modules, it will avoid any ports that seem to be in use by other devices.
- IRQ autoprobng works only for a few port types at the moment.

## 59.5 Reporting printer problems with parport

If you are having problems printing, please go through these steps to try to narrow down where the problem area is.

When reporting problems with parport, really you need to give all of the messages that parport\_pc spits out when it initialises. There are several code paths:

- polling
- interrupt-driven, protocol in software
- interrupt-driven, protocol in hardware using PIO
- interrupt-driven, protocol in hardware using DMA

The kernel messages that parport\_pc logs give an indication of which code path is being used. (They could be a lot better actually..)

For normal printer protocol, having IEEE 1284 modes enabled or not should not make a difference.

To turn off the 'protocol in hardware' code paths, disable CONFIG\_PARPORT\_PC\_FIFO. Note that when they are enabled they are not necessarily **used**; it depends on whether the hardware is available, enabled by the BIOS, and detected by the driver.

So, to start with, disable CONFIG\_PARPORT\_PC\_FIFO, and load parport\_pc with irq=none. See if printing works then. It really should, because this is the simplest code path.

If that works fine, try with io=0x378 irq=7 (adjust for your hardware), to make it use interrupt-driven in-software protocol.

If **that** works fine, then one of the hardware modes isn't working right. Enable CONFIG\_FIFO (no, it isn't a module option, and yes, it should be), set the port to ECP mode in the BIOS and note the DMA channel, and try with:

```
io=0x378 irq=7 dma=none (for PIO)
io=0x378 irq=7 dma=3 (for DMA)
```

---

[philb@gnu.org](mailto:philb@gnu.org) [tim@cyberelk.net](mailto:tim@cyberelk.net)

## PERF EVENTS AND TOOL SECURITY

### 60.1 Overview

Usage of Performance Counters for Linux (`perf_events`)<sup>1, 2, 3</sup> can impose a considerable risk of leaking sensitive data accessed by monitored processes. The data leakage is possible both in scenarios of direct usage of `perf_events` system call API<sup>2</sup> and over data files generated by Perf tool user mode utility (`Perf`)<sup>3, 4</sup>. The risk depends on the nature of data that `perf_events` performance monitoring units (PMU)<sup>2</sup> and Perf collect and expose for performance analysis. Collected system and performance data may be split into several categories:

1. System hardware and software configuration data, for example: a CPU model and its cache configuration, an amount of available memory and its topology, used kernel and Perf versions, performance monitoring setup including experiment time, events configuration, Perf command line parameters, etc.
2. User and kernel module paths and their load addresses with sizes, process and thread names with their PIDs and TIDs, timestamps for captured hardware and software events.
3. Content of kernel software counters (e.g., for context switches, page faults, CPU migrations), architectural hardware performance counters (PMC)<sup>8</sup> and machine specific registers (MSR)<sup>9</sup> that provide execution metrics for various monitored parts of the system (e.g., memory controller (IMC), interconnect (QPI/UPI) or peripheral (PCIe) uncore counters) without direct attribution to any execution context state.
4. Content of architectural execution context registers (e.g., RIP, RSP, RBP on x86\_64), process user and kernel space memory addresses and data, content of various architectural MSRs that capture data from this category.

Data that belong to the fourth category can potentially contain sensitive process data. If PMUs in some monitoring modes capture values of execution context registers or data from process memory then access to such monitoring modes requires to be ordered and secured properly. So, `perf_events` performance monitoring and

---

<sup>1</sup> <https://lwn.net/Articles/337493/>

<sup>2</sup> [http://man7.org/linux/man-pages/man2/perf\\_event\\_open.2.html](http://man7.org/linux/man-pages/man2/perf_event_open.2.html)

<sup>3</sup> [http://web.eece.maine.edu/~vweaver/projects/perf\\_events/](http://web.eece.maine.edu/~vweaver/projects/perf_events/)

<sup>4</sup> [https://perf.wiki.kernel.org/index.php/Main\\_Page](https://perf.wiki.kernel.org/index.php/Main_Page)

<sup>8</sup> [https://en.wikipedia.org/wiki/Hardware\\_performance\\_counter](https://en.wikipedia.org/wiki/Hardware_performance_counter)

<sup>9</sup> [https://en.wikipedia.org/wiki/Model-specific\\_register](https://en.wikipedia.org/wiki/Model-specific_register)

observability operations are the subject for security access control management<sup>5</sup>.

## 60.2 perf\_events access control

To perform security checks, the Linux implementation splits processes into two categories<sup>6</sup> : a) privileged processes (whose effective user ID is 0, referred to as superuser or root), and b) unprivileged processes (whose effective UID is nonzero). Privileged processes bypass all kernel security permission checks so perf\_events performance monitoring is fully available to privileged processes without access, scope and resource restrictions.

Unprivileged processes are subject to a full security permission check based on the process' s credentials<sup>5</sup> (usually: effective UID, effective GID, and supplementary group list).

Linux divides the privileges traditionally associated with superuser into distinct units, known as capabilities<sup>6</sup> , which can be independently enabled and disabled on per-thread basis for processes and files of unprivileged users.

Unprivileged processes with enabled CAP\_PERFMON capability are treated as privileged processes with respect to perf\_events performance monitoring and observability operations, thus, bypass scope permissions checks in the kernel. CAP\_PERFMON implements the principle of least privilege<sup>13</sup> (POSIX 1003.1e: 2.2.2.39) for performance monitoring and observability operations in the kernel and provides a secure approach to performance monitoring and observability in the system.

For backward compatibility reasons the access to perf\_events monitoring and observability operations is also open for CAP\_SYS\_ADMIN privileged processes but CAP\_SYS\_ADMIN usage for secure monitoring and observability use cases is discouraged with respect to the CAP\_PERFMON capability. If system audit records<sup>14</sup> for a process using perf\_events system call API contain denial records of acquiring both CAP\_PERFMON and CAP\_SYS\_ADMIN capabilities then providing the process with CAP\_PERFMON capability singly is recommended as the preferred secure approach to resolve double access denial logging related to usage of performance monitoring and observability.

Unprivileged processes using perf\_events system call are also subject for PTRACE\_MODE\_READ\_REALCREDS ptrace access mode check<sup>7</sup> , whose outcome determines whether monitoring is permitted. So unprivileged processes provided with CAP\_SYS\_PTRACE capability are effectively permitted to pass the check.

Other capabilities being granted to unprivileged processes can effectively enable capturing of additional data required for later performance analysis of monitored processes or a system. For example, CAP\_SYSLOG capability permits reading kernel space memory addresses from /proc/kallsyms file.

---

<sup>5</sup> <https://www.kernel.org/doc/html/latest/security/credentials.html>

<sup>6</sup> <http://man7.org/linux/man-pages/man7/capabilities.7.html>

<sup>13</sup> <https://sites.google.com/site/fullycapable>

<sup>14</sup> <http://man7.org/linux/man-pages/man8/auditd.8.html>

<sup>7</sup> <http://man7.org/linux/man-pages/man2/ptrace.2.html>

## 60.3 Privileged Perf users groups

Mechanisms of capabilities, privileged capability-dumb files<sup>6</sup> and file system ACLs<sup>10</sup> can be used to create dedicated groups of privileged Perf users who are permitted to execute performance monitoring and observability without scope limits. The following steps can be taken to create such groups of privileged Perf users.

1. Create perf\_users group of privileged Perf users, assign perf\_users group to Perf tool executable and limit access to the executable for other users in the system who are not in the perf\_users group:

```
# groupadd perf_users
# ls -alhF
-rwxr-xr-x  2 root root  11M Oct 19 15:12 perf
# chgrp perf_users perf
# ls -alhF
-rwxr-xr-x  2 root perf_users  11M Oct 19 15:12 perf
# chmod o-rwx perf
# ls -alhF
-rwxr-x---  2 root perf_users  11M Oct 19 15:12 perf
```

2. Assign the required capabilities to the Perf tool executable file and enable members of perf\_users group with monitoring and observability privileges<sup>6</sup> :

```
# setcap "cap_perfmon,cap_sys_ptrace,cap_syslog=ep" perf
# setcap -v "cap_perfmon,cap_sys_ptrace,cap_syslog=ep" perf
perf: OK
# getcap perf
perf = cap_sys_ptrace,cap_syslog,cap_perfmon+ep
```

If the libcap installed doesn't yet support "cap\_perfmon", use "38" instead, i.e.:

```
# setcap "38,cap_ipc_lock,cap_sys_ptrace,cap_syslog=ep" perf
```

Note that you may need to have 'cap\_ipc\_lock' in the mix for tools such as 'perf top', alternatively use 'perf top -m N', to reduce the memory that it uses for the perf ring buffer, see the memory allocation section below.

Using a libcap without support for CAP\_PERFMON will make cap\_get\_flag(caps, 38, CAP\_EFFECTIVE, &val) fail, which will lead the default event to be 'cycles:u', so as a workaround explicitly ask for the 'cycles' event, i.e.:

```
# perf top -e cycles
```

To get kernel and user samples with a perf binary with just CAP\_PERFMON.

As a result, members of perf\_users group are capable of conducting performance monitoring and observability by using functionality of the configured Perf tool executable that, when executes, passes perf\_events subsystem scope checks.

This specific access control management is only available to superuser or root running processes with CAP\_SETPCAP, CAP\_SETFCAP<sup>6</sup> capabilities.

<sup>10</sup> <http://man7.org/linux/man-pages/man5/acl.5.html>

## 60.4 Unprivileged users

`perf_events` scope and access control for unprivileged processes is governed by `perf_event Paranoid`<sup>2</sup> setting:

- 1:** Impose no scope and access restrictions on using `perf_events` performance monitoring. Per-user per-cpu `perf_event mlock_kb`<sup>2</sup> locking limit is ignored when allocating memory buffers for storing performance data. This is the least secure mode since allowed monitored scope is maximized and no `perf_events` specific limits are imposed on resources allocated for performance monitoring.
- >=0:** scope includes per-process and system wide performance monitoring but excludes raw tracepoints and `ftrace` function tracepoints monitoring. CPU and system events happened when executing either in user or in kernel space can be monitored and captured for later analysis. Per-user per-cpu `perf_event mlock_kb` locking limit is imposed but ignored for unprivileged processes with `CAP_IPC_LOCK`<sup>6</sup> capability.
- >=1:** scope includes per-process performance monitoring only and excludes system wide performance monitoring. CPU and system events happened when executing either in user or in kernel space can be monitored and captured for later analysis. Per-user per-cpu `perf_event mlock_kb` locking limit is imposed but ignored for unprivileged processes with `CAP_IPC_LOCK` capability.
- >=2:** scope includes per-process performance monitoring only. CPU and system events happened when executing in user space only can be monitored and captured for later analysis. Per-user per-cpu `perf_event mlock_kb` locking limit is imposed but ignored for unprivileged processes with `CAP_IPC_LOCK` capability.

## 60.5 Resource control

### 60.5.1 Open file descriptors

The `perf_events` system call API<sup>2</sup> allocates file descriptors for every configured PMU event. Open file descriptors are a per-process accountable resource governed by the `RLIMIT_NOFILE`<sup>11</sup> limit (`ulimit -n`), which is usually derived from the login shell process. When configuring Perf collection for a long list of events on a large server system, this limit can be easily hit preventing required monitoring configuration. `RLIMIT_NOFILE` limit can be increased on per-user basis modifying content of the `limits.conf` file<sup>12</sup>. Ordinarily, a Perf sampling session (`perf record`) requires an amount of open `perf_event` file descriptors that is not less than the number of monitored events multiplied by the number of monitored CPUs.

---

<sup>11</sup> <http://man7.org/linux/man-pages/man2/getrlimit.2.html>

<sup>12</sup> <http://man7.org/linux/man-pages/man5/limits.conf.5.html>

## 60.5.2 Memory allocation

The amount of memory available to user processes for capturing performance monitoring data is governed by the `perf_event_mlock_kb`<sup>2</sup> setting. This `perf_event` specific resource setting defines overall per-cpu limits of memory allowed for mapping by the user processes to execute performance monitoring. The setting essentially extends the `RLIMIT_MEMLOCK`<sup>11</sup> limit, but only for memory regions mapped specifically for capturing monitored performance events and related data.

For example, if a machine has eight cores and `perf_event_mlock_kb` limit is set to 516 KiB, then a user process is provided with  $516 \text{ KiB} * 8 = 4128 \text{ KiB}$  of memory above the `RLIMIT_MEMLOCK` limit (`ulimit -l`) for `perf_event` mmap buffers. In particular, this means that, if the user wants to start two or more performance monitoring processes, the user is required to manually distribute the available 4128 KiB between the monitoring processes, for example, using the `-mmap-pages` Perf record mode option. Otherwise, the first started performance monitoring process allocates all available 4128 KiB and the other processes will fail to proceed due to the lack of memory.

`RLIMIT_MEMLOCK` and `perf_event_mlock_kb` resource constraints are ignored for processes with the `CAP_IPC_LOCK` capability. Thus, `perf_events/Perf` privileged users can be provided with memory above the constraints for `perf_events/Perf` performance monitoring purpose by providing the Perf executable with `CAP_IPC_LOCK` capability.

## 60.6 Bibliography



## POWER MANAGEMENT

### 61.1 Power Management Strategies

**Copyright** © 2017 Intel Corporation

**Author** Rafael J. Wysocki <[rafael.j.wysocki@intel.com](mailto:rafael.j.wysocki@intel.com)>

The Linux kernel supports two major high-level power management strategies.

One of them is based on using global low-power states of the whole system in which user space code cannot be executed and the overall system activity is significantly reduced, referred to as sleep states. The kernel puts the system into one of these states when requested by user space and the system stays in it until a special signal is received from one of designated devices, triggering a transition to the working state in which user space code can run. Because sleep states are global and the whole system is affected by the state changes, this strategy is referred to as the system-wide power management.

The other strategy, referred to as the working-state power management, is based on adjusting the power states of individual hardware components of the system, as needed, in the working state. In consequence, if this strategy is in use, the working state of the system usually does not correspond to any particular physical configuration of it, but can be treated as a metastate covering a range of different power states of the system in which the individual components of it can be either active (in use) or inactive (idle). If they are active, they have to be in power states allowing them to process data and to be accessed by software. In turn, if they are inactive, ideally, they should be in low-power states in which they may not be accessible.

If all of the system components are active, the system as a whole is regarded as “runtime active” and that situation typically corresponds to the maximum power draw (or maximum energy usage) of it. If all of them are inactive, the system as a whole is regarded as “runtime idle” which may be very close to a sleep state from the physical system configuration and power draw perspective, but then it takes much less time and effort to start executing user space code than for the same system in a sleep state. However, transitions from sleep states back to the working state can only be started by a limited set of devices, so typically the system can spend much more time in a sleep state than it can be runtime idle in one go. For this reason, systems usually use less energy in sleep states than when they are runtime idle most of the time.

Moreover, the two power management strategies address different usage scenarios. Namely, if the user indicates that the system will not be in use going forward,

for example by closing its lid (if the system is a laptop), it probably should go into a sleep state at that point. On the other hand, if the user simply goes away from the laptop keyboard, it probably should stay in the working state and use the working-state power management in case it becomes idle, because the user may come back to it at any time and then may want the system to be immediately accessible.

## 61.2 System-Wide Power Management

### 61.2.1 System Sleep States

**Copyright** © 2017 Intel Corporation

**Author** Rafael J. Wysocki <rafael.j.wysocki@intel.com>

Sleep states are global low-power states of the entire system in which user space code cannot be executed and the overall system activity is significantly reduced.

#### Sleep States That Can Be Supported

Depending on its configuration and the capabilities of the platform it runs on, the Linux kernel can support up to four system sleep states, including hibernation and up to three variants of system suspend. The sleep states that can be supported by the kernel are listed below.

#### Suspend-to-Idle

This is a generic, pure software, light-weight variant of system suspend (also referred to as S2I or S2Idle). It allows more energy to be saved relative to runtime idle by freezing user space, suspending the timekeeping and putting all I/O devices into low-power states (possibly lower-power than available in the working state), such that the processors can spend time in their deepest idle states while the system is suspended.

The system is woken up from this state by in-band interrupts, so theoretically any devices that can cause interrupts to be generated in the working state can also be set up as wakeup devices for S2Idle.

This state can be used on platforms without support for standby or suspend-to-RAM, or it can be used in addition to any of the deeper system suspend variants to provide reduced resume latency. It is always supported if the CONFIG\_SUSPEND kernel configuration option is set.

## Standby

This state, if supported, offers moderate, but real, energy savings, while providing a relatively straightforward transition back to the working state. No operating state is lost (the system core logic retains power), so the system can go back to where it left off easily enough.

In addition to freezing user space, suspending the timekeeping and putting all I/O devices into low-power states, which is done for suspend-to-idle too, nonboot CPUs are taken offline and all low-level system functions are suspended during transitions into this state. For this reason, it should allow more energy to be saved relative to suspend-to-idle, but the resume latency will generally be greater than for that state.

The set of devices that can wake up the system from this state usually is reduced relative to suspend-to-idle and it may be necessary to rely on the platform for setting up the wakeup functionality as appropriate.

This state is supported if the `CONFIG_SUSPEND` kernel configuration option is set and the support for it is registered by the platform with the core system suspend subsystem. On ACPI-based systems this state is mapped to the S1 system state defined by ACPI.

## Suspend-to-RAM

This state (also referred to as STR or S2RAM), if supported, offers significant energy savings as everything in the system is put into a low-power state, except for memory, which should be placed into the self-refresh mode to retain its contents. All of the steps carried out when entering standby are also carried out during transitions to S2RAM. Additional operations may take place depending on the platform capabilities. In particular, on ACPI-based systems the kernel passes control to the platform firmware (BIOS) as the last step during S2RAM transitions and that usually results in powering down some more low-level components that are not directly controlled by the kernel.

The state of devices and CPUs is saved and held in memory. All devices are suspended and put into low-power states. In many cases, all peripheral buses lose power when entering S2RAM, so devices must be able to handle the transition back to the “on” state.

On ACPI-based systems S2RAM requires some minimal boot-strapping code in the platform firmware to resume the system from it. This may be the case on other platforms too.

The set of devices that can wake up the system from S2RAM usually is reduced relative to suspend-to-idle and standby and it may be necessary to rely on the platform for setting up the wakeup functionality as appropriate.

S2RAM is supported if the `CONFIG_SUSPEND` kernel configuration option is set and the support for it is registered by the platform with the core system suspend subsystem. On ACPI-based systems it is mapped to the S3 system state defined by ACPI.

### Hibernation

This state (also referred to as Suspend-to-Disk or STD) offers the greatest energy savings and can be used even in the absence of low-level platform support for system suspend. However, it requires some low-level code for resuming the system to be present for the underlying CPU architecture.

Hibernation is significantly different from any of the system suspend variants. It takes three system state changes to put it into hibernation and two system state changes to resume it.

First, when hibernation is triggered, the kernel stops all system activity and creates a snapshot image of memory to be written into persistent storage. Next, the system goes into a state in which the snapshot image can be saved, the image is written out and finally the system goes into the target low-power state in which power is cut from almost all of its hardware components, including memory, except for a limited set of wakeup devices.

Once the snapshot image has been written out, the system may either enter a special low-power state (like ACPI S4), or it may simply power down itself. Powering down means minimum power draw and it allows this mechanism to work on any system. However, entering a special low-power state may allow additional means of system wakeup to be used (e.g. pressing a key on the keyboard or opening a laptop lid).

After wakeup, control goes to the platform firmware that runs a boot loader which boots a fresh instance of the kernel (control may also go directly to the boot loader, depending on the system configuration, but anyway it causes a fresh instance of the kernel to be booted). That new instance of the kernel (referred to as the restore kernel) looks for a hibernation image in persistent storage and if one is found, it is loaded into memory. Next, all activity in the system is stopped and the restore kernel overwrites itself with the image contents and jumps into a special trampoline area in the original kernel stored in the image (referred to as the image kernel), which is where the special architecture-specific low-level code is needed. Finally, the image kernel restores the system to the pre-hibernation state and allows user space to run again.

Hibernation is supported if the `CONFIG_HIBERNATION` kernel configuration option is set. However, this option can only be set if support for the given CPU architecture includes the low-level code for system resume.

### Basic sysfs Interfaces for System Suspend and Hibernation

The power management subsystem provides userspace with a unified `sysfs` interface for system sleep regardless of the underlying system architecture or platform. That interface is located in the `/sys/power/` directory (assuming that `sysfs` is mounted at `/sys`) and it consists of the following attributes (files):

**state** This file contains a list of strings representing sleep states supported by the kernel. Writing one of these strings into it causes the kernel to start a transition of the system into the sleep state represented by that string.

In particular, the “disk”, “freeze” and “standby” strings represent the hibernation, suspend-to-idle and standby sleep states, respectively. The “mem” string

is interpreted in accordance with the contents of the `mem_sleep` file described below.

If the kernel does not support any system sleep states, this file is not present.

**mem\_sleep** This file contains a list of strings representing supported system suspend variants and allows user space to select the variant to be associated with the “mem” string in the state file described above.

The strings that may be present in this file are “s2idle”, “shallow” and “deep”. The “s2idle” string always represents suspend-to-idle and, by convention, “shallow” and “deep” represent standby and suspend-to-RAM, respectively.

Writing one of the listed strings into this file causes the system suspend variant represented by it to be associated with the “mem” string in the state file. The string representing the suspend variant currently associated with the “mem” string in the state file is shown in square brackets.

If the kernel does not support system suspend, this file is not present.

**disk** This file controls the operating mode of hibernation (Suspend-to-Disk). Specifically, it tells the kernel what to do after creating a hibernation image.

Reading from it returns a list of supported options encoded as:

**platform** Put the system into a special low-power state (e.g. ACPI S4) to make additional wakeup options available and possibly allow the platform firmware to take a simplified initialization path after wakeup.

It is only available if the platform provides a special mechanism to put the system to sleep after creating a hibernation image (platforms with ACPI do that as a rule, for example).

**shutdown** Power off the system.

**reboot** Reboot the system (useful for diagnostics mostly).

**suspend** Hybrid system suspend. Put the system into the suspend sleep state selected through the `mem_sleep` file described above. If the system is successfully woken up from that state, discard the hibernation image and continue. Otherwise, use the image to restore the previous state of the system.

It is available if system suspend is supported.

**test\_resume** Diagnostic operation. Load the image as though the system had just woken up from hibernation and the currently running kernel instance was a restore kernel and follow up with full system resume.

Writing one of the strings listed above into this file causes the option represented by it to be selected.

The currently selected option is shown in square brackets, which means that the operation represented by it will be carried out after creating and saving the image when hibernation is triggered by writing `disk` to `/sys/power/state`.

If the kernel does not support hibernation, this file is not present.

**image\_size** This file controls the size of hibernation images.

It can be written a string representing a non-negative integer that will be used as a best-effort upper limit of the image size, in bytes. The hibernation core will do its best to ensure that the image size will not exceed that number, but if that turns out to be impossible to achieve, a hibernation image will still be created and its size will be as small as possible. In particular, writing ‘0’ to this file causes the size of hibernation images to be minimum.

Reading from it returns the current image size limit, which is set to around 2/5 of the available RAM size by default.

**pm\_trace** This file controls the “PM trace” mechanism saving the last suspend or resume event point in the RTC memory across reboots. It helps to debug hard lockups or reboots due to device driver failures that occur during system suspend or resume (which is more common) more effectively.

If it contains “1”, the fingerprint of each suspend/resume event point in turn will be stored in the RTC memory (overwriting the actual RTC information), so it will survive a system crash if one occurs right after storing it and it can be used later to identify the driver that caused the crash to happen.

It contains “0” by default, which may be changed to “1” by writing a string representing a nonzero integer into it.

According to the above, there are two ways to make the system go into the suspend-to-idle state. The first one is to write “freeze” directly to `/sys/power/state`. The second one is to write “s2idle” to `/sys/power/mem_sleep` and then to write “mem” to `/sys/power/state`. Likewise, there are two ways to make the system go into the standby state (the strings to write to the control files in that case are “standby” or “shallow” and “mem”, respectively) if that state is supported by the platform. However, there is only one way to make the system go into the suspend-to-RAM state (write “deep” into `/sys/power/mem_sleep` and “mem” into `/sys/power/state`).

The default suspend variant (ie. the one to be used without writing anything into `/sys/power/mem_sleep`) is either “deep” (on the majority of systems supporting suspend-to-RAM) or “s2idle”, but it can be overridden by the value of the `mem_sleep_default` parameter in the kernel command line. On some systems with ACPI, depending on the information in the ACPI tables, the default may be “s2idle” even if suspend-to-RAM is supported in principle.

### 61.2.2 System Suspend Code Flows

**Copyright** © 2020 Intel Corporation

**Author** Rafael J. Wysocki <[rafael.j.wysocki@intel.com](mailto:rafael.j.wysocki@intel.com)>

At least one global system-wide transition needs to be carried out for the system to get from the working state into one of the supported sleep states. Hibernation requires more than one transition to occur for this purpose, but the other sleep states, commonly referred to as system-wide suspend (or simply system suspend) states, need only one.

For those sleep states, the transition from the working state of the system into the target sleep state is referred to as system suspend too (in the majority of cases, whether this means a transition or a sleep state of the system should be clear from

the context) and the transition back from the sleep state into the working state is referred to as system resume.

The kernel code flows associated with the suspend and resume transitions for different sleep states of the system are quite similar, but there are some significant differences between the suspend-to-idle code flows and the code flows related to the suspend-to-RAM and standby sleep states.

The suspend-to-RAM and standby sleep states cannot be implemented without platform support and the difference between them boils down to the platform-specific actions carried out by the suspend and resume hooks that need to be provided by the platform driver to make them available. Apart from that, the suspend and resume code flows for these sleep states are mostly identical, so they both together will be referred to as platform-dependent suspend states in what follows.

### **Suspend-to-idle Suspend Code Flow**

The following steps are taken in order to transition the system from the working state to the suspend-to-idle sleep state:

1. Invoking system-wide suspend notifiers.

Kernel subsystems can register callbacks to be invoked when the suspend transition is about to occur and when the resume transition has finished.

That allows them to prepare for the change of the system state and to clean up after getting back to the working state.

2. Freezing tasks.

Tasks are frozen primarily in order to avoid unchecked hardware accesses from user space through MMIO regions or I/O registers exposed directly to it and to prevent user space from entering the kernel while the next step of the transition is in progress (which might have been problematic for various reasons).

All user space tasks are intercepted as though they were sent a signal and put into uninterruptible sleep until the end of the subsequent system resume transition.

The kernel threads that choose to be frozen during system suspend for specific reasons are frozen subsequently, but they are not intercepted. Instead, they are expected to periodically check whether or not they need to be frozen and to put themselves into uninterruptible sleep if so. [Note, however, that kernel threads can use locking and other concurrency controls available in kernel space to synchronize themselves with system suspend and resume, which can be much more precise than the freezing, so the latter is not a recommended option for kernel threads.]

3. Suspending devices and reconfiguring IRQs.

Devices are suspended in four phases called prepare, suspend, late suspend and noirq suspend (see `driverapi_pm_devices` for more information on what exactly happens in each phase).

Every device is visited in each phase, but typically it is not physically accessed in more than two of them.

The runtime PM API is disabled for every device during the late suspend phase and high-level (“action”) interrupt handlers are prevented from being invoked before the noirq suspend phase.

Interrupts are still handled after that, but they are only acknowledged to interrupt controllers without performing any device-specific actions that would be triggered in the working state of the system (those actions are deferred till the subsequent system resume transition as described below).

IRQs associated with system wakeup devices are “armed” so that the resume transition of the system is started when one of them signals an event.

#### 4. Freezing the scheduler tick and suspending timekeeping.

When all devices have been suspended, CPUs enter the idle loop and are put into the deepest available idle state. While doing that, each of them “freezes” its own scheduler tick so that the timer events associated with the tick do not occur until the CPU is woken up by another interrupt source.

The last CPU to enter the idle state also stops the timekeeping which (among other things) prevents high resolution timers from triggering going forward until the first CPU that is woken up restarts the timekeeping. That allows the CPUs to stay in the deep idle state relatively long in one go.

From this point on, the CPUs can only be woken up by non-timer hardware interrupts. If that happens, they go back to the idle state unless the interrupt that woke up one of them comes from an IRQ that has been armed for system wakeup, in which case the system resume transition is started.

### Suspend-to-idle Resume Code Flow

The following steps are taken in order to transition the system from the suspend-to-idle sleep state into the working state:

#### 1. Resuming timekeeping and unfreezing the scheduler tick.

When one of the CPUs is woken up (by a non-timer hardware interrupt), it leaves the idle state entered in the last step of the preceding suspend transition, restarts the timekeeping (unless it has been restarted already by another CPU that woke up earlier) and the scheduler tick on that CPU is unfrozen.

If the interrupt that has woken up the CPU was armed for system wakeup, the system resume transition begins.

#### 2. Resuming devices and restoring the working-state configuration of IRQs.

Devices are resumed in four phases called noirq resume, early resume, resume and complete (see `driverapi_pm_devices` for more information on what exactly happens in each phase).

Every device is visited in each phase, but typically it is not physically accessed in more than two of them.

The working-state configuration of IRQs is restored after the noirq resume phase and the runtime PM API is re-enabled for every device whose driver supports it during the early resume phase.

### 3. Thawing tasks.

Tasks frozen in step 2 of the preceding suspend transition are “thawed” , which means that they are woken up from the uninterruptible sleep that they went into at that time and user space tasks are allowed to exit the kernel.

### 4. Invoking system-wide resume notifiers.

This is analogous to step 1 of the suspend transition and the same set of callbacks is invoked at this point, but a different “notification type” parameter value is passed to them.

## **Platform-dependent Suspend Code Flow**

The following steps are taken in order to transition the system from the working state to platform-dependent suspend state:

### 1. Invoking system-wide suspend notifiers.

This step is the same as step 1 of the suspend-to-idle suspend transition described above.

### 2. Freezing tasks.

This step is the same as step 2 of the suspend-to-idle suspend transition described above.

### 3. Suspending devices and reconfiguring IRQs.

This step is analogous to step 3 of the suspend-to-idle suspend transition described above, but the arming of IRQs for system wakeup generally does not have any effect on the platform.

There are platforms that can go into a very deep low-power state internally when all CPUs in them are in sufficiently deep idle states and all I/O devices have been put into low-power states. On those platforms, suspend-to-idle can reduce system power very effectively.

On the other platforms, however, low-level components (like interrupt controllers) need to be turned off in a platform-specific way (implemented in the hooks provided by the platform driver) to achieve comparable power reduction.

That usually prevents in-band hardware interrupts from waking up the system, which must be done in a special platform-dependent way. Then, the configuration of system wakeup sources usually starts when system wakeup devices are suspended and is finalized by the platform suspend hooks later on.

### 4. Disabling non-boot CPUs.

On some platforms the suspend hooks mentioned above must run in a one-CPU configuration of the system (in particular, the hardware cannot be accessed by any code running in parallel with the platform suspend hooks that may, and often do, trap into the platform firmware in order to finalize the suspend transition).

For this reason, the CPU offline/online (CPU hotplug) framework is used to take all of the CPUs in the system, except for one (the boot CPU), offline (typically, the CPUs that have been taken offline go into deep idle states).

This means that all tasks are migrated away from those CPUs and all IRQs are rerouted to the only CPU that remains online.

### 5. Suspending core system components.

This prepares the core system components for (possibly) losing power going forward and suspends the timekeeping.

### 6. Platform-specific power removal.

This is expected to remove power from all of the system components except for the memory controller and RAM (in order to preserve the contents of the latter) and some devices designated for system wakeup.

In many cases control is passed to the platform firmware which is expected to finalize the suspend transition as needed.

## Platform-dependent Resume Code Flow

The following steps are taken in order to transition the system from a platform-dependent suspend state into the working state:

### 1. Platform-specific system wakeup.

The platform is woken up by a signal from one of the designated system wakeup devices (which need not be an in-band hardware interrupt) and control is passed back to the kernel (the working configuration of the platform may need to be restored by the platform firmware before the kernel gets control again).

### 2. Resuming core system components.

The suspend-time configuration of the core system components is restored and the timekeeping is resumed.

### 3. Re-enabling non-boot CPUs.

The CPUs disabled in step 4 of the preceding suspend transition are taken back online and their suspend-time configuration is restored.

### 4. Resuming devices and restoring the working-state configuration of IRQs.

This step is the same as step 2 of the suspend-to-idle suspend transition described above.

### 5. Thawing tasks.

This step is the same as step 3 of the suspend-to-idle suspend transition described above.

### 6. Invoking system-wide resume notifiers.

This step is the same as step 4 of the suspend-to-idle suspend transition described above.

## 61.3 Working-State Power Management

### 61.3.1 CPU Idle Time Management

**Copyright** © 2018 Intel Corporation

**Author** Rafael J. Wysocki <rafael.j.wysocki@intel.com>

#### Concepts

Modern processors are generally able to enter states in which the execution of a program is suspended and instructions belonging to it are not fetched from memory or executed. Those states are the idle states of the processor.

Since part of the processor hardware is not used in idle states, entering them generally allows power drawn by the processor to be reduced and, in consequence, it is an opportunity to save energy.

CPU idle time management is an energy-efficiency feature concerned about using the idle states of processors for this purpose.

#### Logical CPUs

CPU idle time management operates on CPUs as seen by the CPU scheduler (that is the part of the kernel responsible for the distribution of computational work in the system). In its view, CPUs are logical units. That is, they need not be separate physical entities and may just be interfaces appearing to software as individual single-core processors. In other words, a CPU is an entity which appears to be fetching instructions that belong to one sequence (program) from memory and executing them, but it need not work this way physically. Generally, three different cases can be consider here.

First, if the whole processor can only follow one sequence of instructions (one program) at a time, it is a CPU. In that case, if the hardware is asked to enter an idle state, that applies to the processor as a whole.

Second, if the processor is multi-core, each core in it is able to follow at least one program at a time. The cores need not be entirely independent of each other (for example, they may share caches), but still most of the time they work physically in parallel with each other, so if each of them executes only one program, those programs run mostly independently of each other at the same time. The entire cores are CPUs in that case and if the hardware is asked to enter an idle state, that applies to the core that asked for it in the first place, but it also may apply to a larger unit (say a “package” or a “cluster” ) that the core belongs to (in fact, it may apply to an entire hierarchy of larger units containing the core). Namely, if all of the cores in the larger unit except for one have been put into idle states at the “core level” and the remaining core asks the processor to enter an idle state, that may trigger it to put the whole larger unit into an idle state which also will affect the other cores in that unit.

Finally, each core in a multi-core processor may be able to follow more than one program in the same time frame (that is, each core may be able to fetch instruc-

tions from multiple locations in memory and execute them in the same time frame, but not necessarily entirely in parallel with each other). In that case the cores present themselves to software as “bundles” each consisting of multiple individual single-core “processors”, referred to as hardware threads (or hyper-threads specifically on Intel hardware), that each can follow one sequence of instructions. Then, the hardware threads are CPUs from the CPU idle time management perspective and if the processor is asked to enter an idle state by one of them, the hardware thread (or CPU) that asked for it is stopped, but nothing more happens, unless all of the other hardware threads within the same core also have asked the processor to enter an idle state. In that situation, the core may be put into an idle state individually or a larger unit containing it may be put into an idle state as a whole (if the other cores within the larger unit are in idle states already).

### Idle CPUs

Logical CPUs, simply referred to as “CPUs” in what follows, are regarded as idle by the Linux kernel when there are no tasks to run on them except for the special “idle” task.

Tasks are the CPU scheduler’s representation of work. Each task consists of a sequence of instructions to execute, or code, data to be manipulated while running that code, and some context information that needs to be loaded into the processor every time the task’s code is run by a CPU. The CPU scheduler distributes work by assigning tasks to run to the CPUs present in the system.

Tasks can be in various states. In particular, they are runnable if there are no specific conditions preventing their code from being run by a CPU as long as there is a CPU available for that (for example, they are not waiting for any events to occur or similar). When a task becomes runnable, the CPU scheduler assigns it to one of the available CPUs to run and if there are no more runnable tasks assigned to it, the CPU will load the given task’s context and run its code (from the instruction following the last one executed so far, possibly by another CPU). [If there are multiple runnable tasks assigned to one CPU simultaneously, they will be subject to prioritization and time sharing in order to allow them to make some progress over time.]

The special “idle” task becomes runnable if there are no other runnable tasks assigned to the given CPU and the CPU is then regarded as idle. In other words, in Linux idle CPUs run the code of the “idle” task called the idle loop. That code may cause the processor to be put into one of its idle states, if they are supported, in order to save energy, but if the processor does not support any idle states, or there is not enough time to spend in an idle state before the next wakeup event, or there are strict latency constraints preventing any of the available idle states from being used, the CPU will simply execute more or less useless instructions in a loop until it is assigned a new task to run.

## The Idle Loop

The idle loop code takes two major steps in every iteration of it. First, it calls into a code module referred to as the governor that belongs to the CPU idle time management subsystem called `CPUIidle` to select an idle state for the CPU to ask the hardware to enter. Second, it invokes another code module from the `CPUIidle` subsystem, called the driver, to actually ask the processor hardware to enter the idle state selected by the governor.

The role of the governor is to find an idle state most suitable for the conditions at hand. For this purpose, idle states that the hardware can be asked to enter by logical CPUs are represented in an abstract way independent of the platform or the processor architecture and organized in a one-dimensional (linear) array. That array has to be prepared and supplied by the `CPUIidle` driver matching the platform the kernel is running on at the initialization time. This allows `CPUIidle` governors to be independent of the underlying hardware and to work with any platforms that the Linux kernel can run on.

Each idle state present in that array is characterized by two parameters to be taken into account by the governor, the target residency and the (worst-case) exit latency. The target residency is the minimum time the hardware must spend in the given state, including the time needed to enter it (which may be substantial), in order to save more energy than it would save by entering one of the shallower idle states instead. [The “depth” of an idle state roughly corresponds to the power drawn by the processor in that state.] The exit latency, in turn, is the maximum time it will take a CPU asking the processor hardware to enter an idle state to start executing the first instruction after a wakeup from that state. Note that in general the exit latency also must cover the time needed to enter the given state in case the wakeup occurs when the hardware is entering it and it must be entered completely to be exited in an ordered manner.

There are two types of information that can influence the governor’s decisions. First of all, the governor knows the time until the closest timer event. That time is known exactly, because the kernel programs timers and it knows exactly when they will trigger, and it is the maximum time the hardware that the given CPU depends on can spend in an idle state, including the time necessary to enter and exit it. However, the CPU may be woken up by a non-timer event at any time (in particular, before the closest timer triggers) and it generally is not known when that may happen. The governor can only see how much time the CPU actually was idle after it has been woken up (that time will be referred to as the idle duration from now on) and it can use that information somehow along with the time until the closest timer to estimate the idle duration in future. How the governor uses that information depends on what algorithm is implemented by it and that is the primary reason for having more than one governor in the `CPUIidle` subsystem.

There are four `CPUIidle` governors available, `menu`, `TEO`, `ladder` and `haltpoll`. Which of them is used by default depends on the configuration of the kernel and in particular on whether or not the scheduler tick can be stopped by the idle loop. Available governors can be read from the `available_governors`, and the governor can be changed at runtime. The name of the `CPUIidle` governor currently used by the kernel can be read from the `current_governor_ro` or `current_governor` file under `/sys/devices/system/cpu/cpuidle/` in `sysfs`.

Which `CPUIidle` driver is used, on the other hand, usually depends on the platform

the kernel is running on, but there are platforms with more than one matching driver. For example, there are two drivers that can work with the majority of Intel platforms, `intel_idle` and `acpi_idle`, one with hardcoded idle states information and the other able to read that information from the system's ACPI tables, respectively. Still, even in those cases, the driver chosen at the system initialization time cannot be replaced later, so the decision on which one of them to use has to be made early (on Intel platforms the `acpi_idle` driver will be used if `intel_idle` is disabled for some reason or if it does not recognize the processor). The name of the CPUIdle driver currently used by the kernel can be read from the `current_driver` file under `/sys/devices/system/cpu/cpuidle/` in `sysfs`.

### Idle CPUs and The Scheduler Tick

The scheduler tick is a timer that triggers periodically in order to implement the time sharing strategy of the CPU scheduler. Of course, if there are multiple runnable tasks assigned to one CPU at the same time, the only way to allow them to make reasonable progress in a given time frame is to make them share the available CPU time. Namely, in rough approximation, each task is given a slice of the CPU time to run its code, subject to the scheduling class, prioritization and so on and when that time slice is used up, the CPU should be switched over to running (the code of) another task. The currently running task may not want to give the CPU away voluntarily, however, and the scheduler tick is there to make the switch happen regardless. That is not the only role of the tick, but it is the primary reason for using it.

The scheduler tick is problematic from the CPU idle time management perspective, because it triggers periodically and relatively often (depending on the kernel configuration, the length of the tick period is between 1 ms and 10 ms). Thus, if the tick is allowed to trigger on idle CPUs, it will not make sense for them to ask the hardware to enter idle states with target residencies above the tick period length. Moreover, in that case the idle duration of any CPU will never exceed the tick period length and the energy used for entering and exiting idle states due to the tick wakeups on idle CPUs will be wasted.

Fortunately, it is not really necessary to allow the tick to trigger on idle CPUs, because (by definition) they have no tasks to run except for the special "idle" one. In other words, from the CPU scheduler perspective, the only user of the CPU time on them is the idle loop. Since the time of an idle CPU need not be shared between multiple runnable tasks, the primary reason for using the tick goes away if the given CPU is idle. Consequently, it is possible to stop the scheduler tick entirely on idle CPUs in principle, even though that may not always be worth the effort.

Whether or not it makes sense to stop the scheduler tick in the idle loop depends on what is expected by the governor. First, if there is another (non-tick) timer due to trigger within the tick range, stopping the tick clearly would be a waste of time, even though the timer hardware may not need to be reprogrammed in that case. Second, if the governor is expecting a non-timer wakeup within the tick range, stopping the tick is not necessary and it may even be harmful. Namely, in that case the governor will select an idle state with the target residency within the time until the expected wakeup, so that state is going to be relatively shallow. The governor really cannot select a deep idle state then, as that would contradict

its own expectation of a wakeup in short order. Now, if the wakeup really occurs shortly, stopping the tick would be a waste of time and in this case the timer hardware would need to be reprogrammed, which is expensive. On the other hand, if the tick is stopped and the wakeup does not occur any time soon, the hardware may spend indefinite amount of time in the shallow idle state selected by the governor, which will be a waste of energy. Hence, if the governor is expecting a wakeup of any kind within the tick range, it is better to allow the tick trigger. Otherwise, however, the governor will select a relatively deep idle state, so the tick should be stopped so that it does not wake up the CPU too early.

In any case, the governor knows what it is expecting and the decision on whether or not to stop the scheduler tick belongs to it. Still, if the tick has been stopped already (in one of the previous iterations of the loop), it is better to leave it as is and the governor needs to take that into account.

The kernel can be configured to disable stopping the scheduler tick in the idle loop altogether. That can be done through the build-time configuration of it (by unsetting the `CONFIG_NO_HZ_IDLE` configuration option) or by passing `nohz=off` to it in the command line. In both cases, as the stopping of the scheduler tick is disabled, the governor's decisions regarding it are simply ignored by the idle loop code and the tick is never stopped.

The systems that run kernels configured to allow the scheduler tick to be stopped on idle CPUs are referred to as tickless systems and they are generally regarded as more energy-efficient than the systems running kernels in which the tick cannot be stopped. If the given system is tickless, it will use the menu governor by default and if it is not tickless, the default `CPUIdle` governor on it will be ladder.

## The menu Governor

The menu governor is the default `CPUIdle` governor for tickless systems. It is quite complex, but the basic principle of its design is straightforward. Namely, when invoked to select an idle state for a CPU (i.e. an idle state that the CPU will ask the processor hardware to enter), it attempts to predict the idle duration and uses the predicted value for idle state selection.

It first obtains the time until the closest timer event with the assumption that the scheduler tick will be stopped. That time, referred to as the sleep length in what follows, is the upper bound on the time before the next CPU wakeup. It is used to determine the sleep length range, which in turn is needed to get the sleep length correction factor.

The menu governor maintains two arrays of sleep length correction factors. One of them is used when tasks previously running on the given CPU are waiting for some I/O operations to complete and the other one is used when that is not the case. Each array contains several correction factor values that correspond to different sleep length ranges organized so that each range represented in the array is approximately 10 times wider than the previous one.

The correction factor for the given sleep length range (determined before selecting the idle state for the CPU) is updated after the CPU has been woken up and the closer the sleep length is to the observed idle duration, the closer to 1 the correction factor becomes (it must fall between 0 and 1 inclusive). The sleep length

is multiplied by the correction factor for the range that it falls into to obtain the first approximation of the predicted idle duration.

Next, the governor uses a simple pattern recognition algorithm to refine its idle duration prediction. Namely, it saves the last 8 observed idle duration values and, when predicting the idle duration next time, it computes the average and variance of them. If the variance is small (smaller than 400 square milliseconds) or it is small relative to the average (the average is greater than 6 times the standard deviation), the average is regarded as the “typical interval” value. Otherwise, the longest of the saved observed idle duration values is discarded and the computation is repeated for the remaining ones. Again, if the variance of them is small (in the above sense), the average is taken as the “typical interval” value and so on, until either the “typical interval” is determined or too many data points are disregarded, in which case the “typical interval” is assumed to equal “infinity” (the maximum unsigned integer value). The “typical interval” computed this way is compared with the sleep length multiplied by the correction factor and the minimum of the two is taken as the predicted idle duration.

Then, the governor computes an extra latency limit to help “interactive” workloads. It uses the observation that if the exit latency of the selected idle state is comparable with the predicted idle duration, the total time spent in that state probably will be very short and the amount of energy to save by entering it will be relatively small, so likely it is better to avoid the overhead related to entering that state and exiting it. Thus selecting a shallower state is likely to be a better option then. The first approximation of the extra latency limit is the predicted idle duration itself which additionally is divided by a value depending on the number of tasks that previously ran on the given CPU and now they are waiting for I/O operations to complete. The result of that division is compared with the latency limit coming from the power management quality of service, or PM QoS, framework and the minimum of the two is taken as the limit for the idle states’ exit latency.

Now, the governor is ready to walk the list of idle states and choose one of them. For this purpose, it compares the target residency of each state with the predicted idle duration and the exit latency of it with the computed latency limit. It selects the state with the target residency closest to the predicted idle duration, but still below it, and exit latency that does not exceed the limit.

In the final step the governor may still need to refine the idle state selection if it has not decided to stop the scheduler tick. That happens if the idle duration predicted by it is less than the tick period and the tick has not been stopped already (in a previous iteration of the idle loop). Then, the sleep length used in the previous computations may not reflect the real time until the closest timer event and if it really is greater than that time, the governor may need to select a shallower state with a suitable target residency.

## The Timer Events Oriented (TEO) Governor

The timer events oriented (TEO) governor is an alternative CPUIdle governor for tickless systems. It follows the same basic strategy as the menu one: it always tries to find the deepest idle state suitable for the given conditions. However, it applies a different approach to that problem.

First, it does not use sleep length correction factors, but instead it attempts to correlate the observed idle duration values with the available idle states and use that information to pick up the idle state that is most likely to “match” the upcoming CPU idle interval. Second, it does not take the tasks that were running on the given CPU in the past and are waiting on some I/O operations to complete now at all (there is no guarantee that they will run on the same CPU when they become runnable again) and the pattern detection code in it avoids taking timer wakeups into account. It also only uses idle duration values less than the current time till the closest timer (with the scheduler tick excluded) for that purpose.

Like in the menu governor case, the first step is to obtain the sleep length, which is the time until the closest timer event with the assumption that the scheduler tick will be stopped (that also is the upper bound on the time until the next CPU wakeup). That value is then used to preselect an idle state on the basis of three metrics maintained for each idle state provided by the CPUIdle driver: hits, misses and early\_hits.

The hits and misses metrics measure the likelihood that a given idle state will “match” the observed (post-wakeup) idle duration if it “matches” the sleep length. They both are subject to decay (after a CPU wakeup) every time the target residency of the idle state corresponding to them is less than or equal to the sleep length and the target residency of the next idle state is greater than the sleep length (that is, when the idle state corresponding to them “matches” the sleep length). The hits metric is increased if the former condition is satisfied and the target residency of the given idle state is less than or equal to the observed idle duration and the target residency of the next idle state is greater than the observed idle duration at the same time (that is, it is increased when the given idle state “matches” both the sleep length and the observed idle duration). In turn, the misses metric is increased when the given idle state “matches” the sleep length only and the observed idle duration is too short for its target residency.

The early\_hits metric measures the likelihood that a given idle state will “match” the observed (post-wakeup) idle duration if it does not “match” the sleep length. It is subject to decay on every CPU wakeup and it is increased when the idle state corresponding to it “matches” the observed (post-wakeup) idle duration and the target residency of the next idle state is less than or equal to the sleep length (i.e. the idle state “matching” the sleep length is deeper than the given one).

The governor walks the list of idle states provided by the CPUIdle driver and finds the last (deepest) one with the target residency less than or equal to the sleep length. Then, the hits and misses metrics of that idle state are compared with each other and it is preselected if the hits one is greater (which means that that idle state is likely to “match” the observed idle duration after CPU wakeup). If the misses one is greater, the governor preselects the shallower idle state with the maximum early\_hits metric (or if there are multiple shallower idle states with equal early\_hits metric which also is the maximum, the shallowest of them will be preselected). [If there is a wakeup latency constraint coming from the PM

QoS framework which is hit before reaching the deepest idle state with the target residency within the sleep length, the deepest idle state with the exit latency within the constraint is preselected without consulting the hits, misses and `early_hits` metrics.]

Next, the governor takes several idle duration values observed most recently into consideration and if at least a half of them are greater than or equal to the target residency of the preselected idle state, that idle state becomes the final candidate to ask for. Otherwise, the average of the most recent idle duration values below the target residency of the preselected idle state is computed and the governor walks the idle states shallower than the preselected one and finds the deepest of them with the target residency within that average. That idle state is then taken as the final candidate to ask for.

Still, at this point the governor may need to refine the idle state selection if it has not decided to stop the scheduler tick. That generally happens if the target residency of the idle state selected so far is less than the tick period and the tick has not been stopped already (in a previous iteration of the idle loop). Then, like in the menu governor case, the sleep length used in the previous computations may not reflect the real time until the closest timer event and if it really is greater than that time, a shallower state with a suitable target residency may need to be selected.

### Representation of Idle States

For the CPU idle time management purposes all of the physical idle states supported by the processor have to be represented as a one-dimensional array of `struct cpuidle_state` objects each allowing an individual (logical) CPU to ask the processor hardware to enter an idle state of certain properties. If there is a hierarchy of units in the processor, one `struct cpuidle_state` object can cover a combination of idle states supported by the units at different levels of the hierarchy. In that case, the target residency and exit latency parameters of it, must reflect the properties of the idle state at the deepest level (i.e. the idle state of the unit containing all of the other units).

For example, take a processor with two cores in a larger unit referred to as a “module” and suppose that asking the hardware to enter a specific idle state (say “X”) at the “core” level by one core will trigger the module to try to enter a specific idle state of its own (say “MX”) if the other core is in idle state “X” already. In other words, asking for idle state “X” at the “core” level gives the hardware a license to go as deep as to idle state “MX” at the “module” level, but there is no guarantee that this is going to happen (the core asking for idle state “X” may just end up in that state by itself instead). Then, the target residency of the `struct cpuidle_state` object representing idle state “X” must reflect the minimum time to spend in idle state “MX” of the module (including the time needed to enter it), because that is the minimum time the CPU needs to be idle to save any energy in case the hardware enters that state. Analogously, the exit latency parameter of that object must cover the exit time of idle state “MX” of the module (and usually its entry time too), because that is the maximum delay between a wakeup signal and the time the CPU will start to execute the first new instruction (assuming that both cores in the module will always be ready to execute instructions as soon as the module becomes operational as a whole).

There are processors without direct coordination between different levels of the hierarchy of units inside them, however. In those cases asking for an idle state at the “core” level does not automatically affect the “module” level, for example, in any way and the CPUIdle driver is responsible for the entire handling of the hierarchy. Then, the definition of the idle state objects is entirely up to the driver, but still the physical properties of the idle state that the processor hardware finally goes into must always follow the parameters used by the governor for idle state selection (for instance, the actual exit latency of that idle state must not exceed the exit latency parameter of the idle state object selected by the governor).

In addition to the target residency and exit latency idle state parameters discussed above, the objects representing idle states each contain a few other parameters describing the idle state and a pointer to the function to run in order to ask the hardware to enter that state. Also, for each `struct cpuidle_state` object, there is a corresponding `struct cpuidle_state_usage` one containing usage statistics of the given idle state. That information is exposed by the kernel via `sysfs`.

For each CPU in the system, there is a `/sys/devices/system/cpu<N>/cpuidle/` directory in `sysfs`, where the number `<N>` is assigned to the given CPU at the initialization time. That directory contains a set of subdirectories called `state0`, `state1` and so on, up to the number of idle state objects defined for the given CPU minus one. Each of these directories corresponds to one idle state object and the larger the number in its name, the deeper the (effective) idle state represented by it. Each of them contains a number of files (attributes) representing the properties of the idle state object corresponding to it, as follows:

**above** Total number of times this idle state had been asked for, but the observed idle duration was certainly too short to match its target residency.

**below** Total number of times this idle state had been asked for, but certainly a deeper idle state would have been a better match for the observed idle duration.

**desc** Description of the idle state.

**disable** Whether or not this idle state is disabled.

**default\_status** The default status of this state, “enabled” or “disabled” .

**latency** Exit latency of the idle state in microseconds.

**name** Name of the idle state.

**power** Power drawn by hardware in this idle state in milliwatts (if specified, 0 otherwise).

**residency** Target residency of the idle state in microseconds.

**time** Total time spent in this idle state by the given CPU (as measured by the kernel) in microseconds.

**usage** Total number of times the hardware has been asked by the given CPU to enter this idle state.

The `desc` and `name` files both contain strings. The difference between them is that the name is expected to be more concise, while the description may be longer and it may contain white space or special characters. The other files listed above contain integer numbers.

The `disable` attribute is the only writeable one. If it contains 1, the given idle state is disabled for this particular CPU, which means that the governor will never select it for this particular CPU and the `CPUIdle` driver will never ask the hardware to enter it for that CPU as a result. However, disabling an idle state for one CPU does not prevent it from being asked for by the other CPUs, so it must be disabled for all of them in order to never be asked for by any of them. [Note that, due to the way the `ladder` governor is implemented, disabling an idle state prevents that governor from selecting any idle states deeper than the disabled one too.]

If the `disable` attribute contains 0, the given idle state is enabled for this particular CPU, but it still may be disabled for some or all of the other CPUs in the system at the same time. Writing 1 to it causes the idle state to be disabled for this particular CPU and writing 0 to it allows the governor to take it into consideration for the given CPU and the driver to ask for it, unless that state was disabled globally in the driver (in which case it cannot be used at all).

The `power` attribute is not defined very well, especially for idle state objects representing combinations of idle states at different levels of the hierarchy of units in the processor, and it generally is hard to obtain idle state power numbers for complex hardware, so `power` often contains 0 (not available) and if it contains a nonzero number, that number may not be very accurate and it should not be relied on for anything meaningful.

The number in the `time` file generally may be greater than the total time really spent by the given CPU in the given idle state, because it is measured by the kernel and it may not cover the cases in which the hardware refused to enter this idle state and entered a shallower one instead of it (or even it did not enter any idle state at all). The kernel can only measure the time span between asking the hardware to enter an idle state and the subsequent wakeup of the CPU and it cannot say what really happened in the meantime at the hardware level. Moreover, if the idle state object in question represents a combination of idle states at different levels of the hierarchy of units in the processor, the kernel can never say how deep the hardware went down the hierarchy in any particular case. For these reasons, the only reliable way to find out how much time has been spent by the hardware in different idle states supported by it is to use idle state residency counters in the hardware, if available.

### Power Management Quality of Service for CPUs

The power management quality of service (PM QoS) framework in the Linux kernel allows kernel code and user space processes to set constraints on various energy-efficiency features of the kernel to prevent performance from dropping below a required level.

CPU idle time management can be affected by PM QoS in two ways, through the global CPU latency limit and through the resume latency constraints for individual CPUs. Kernel code (e.g. device drivers) can set both of them with the help of special internal interfaces provided by the PM QoS framework. User space can modify the former by opening the `cpu_dma_latency` special device file under `/dev/` and writing a binary value (interpreted as a signed 32-bit integer) to it. In turn, the resume latency constraint for a CPU can be modified from user space by writing a string (representing a signed 32-bit integer)

to the `power/pm_qos_resume_latency_us` file under `/sys/devices/system/cpu/cpu<N>/` in `sysfs`, where the CPU number `<N>` is allocated at the system initialization time. Negative values will be rejected in both cases and, also in both cases, the written integer number will be interpreted as a requested PM QoS constraint in microseconds.

The requested value is not automatically applied as a new constraint, however, as it may be less restrictive (greater in this particular case) than another constraint previously requested by someone else. For this reason, the PM QoS framework maintains a list of requests that have been made so far for the global CPU latency limit and for each individual CPU, aggregates them and applies the effective (minimum in this particular case) value as the new constraint.

In fact, opening the `cpu_dma_latency` special device file causes a new PM QoS request to be created and added to a global priority list of CPU latency limit requests and the file descriptor coming from the “open” operation represents that request. If that file descriptor is then used for writing, the number written to it will be associated with the PM QoS request represented by it as a new requested limit value. Next, the priority list mechanism will be used to determine the new effective value of the entire list of requests and that effective value will be set as a new CPU latency limit. Thus requesting a new limit value will only change the real limit if the effective “list” value is affected by it, which is the case if it is the minimum of the requested values in the list.

The process holding a file descriptor obtained by opening the `cpu_dma_latency` special device file controls the PM QoS request associated with that file descriptor, but it controls this particular PM QoS request only.

Closing the `cpu_dma_latency` special device file or, more precisely, the file descriptor obtained while opening it, causes the PM QoS request associated with that file descriptor to be removed from the global priority list of CPU latency limit requests and destroyed. If that happens, the priority list mechanism will be used again, to determine the new effective value for the whole list and that value will become the new limit.

In turn, for each CPU there is one resume latency PM QoS request associated with the `power/pm_qos_resume_latency_us` file under `/sys/devices/system/cpu/cpu<N>/` in `sysfs` and writing to it causes this single PM QoS request to be updated regardless of which user space process does that. In other words, this PM QoS request is shared by the entire user space, so access to the file associated with it needs to be arbitrated to avoid confusion. [Arguably, the only legitimate use of this mechanism in practice is to pin a process to the CPU in question and let it use the `sysfs` interface to control the resume latency constraint for it.] It is still only a request, however. It is an entry in a priority list used to determine the effective value to be set as the resume latency constraint for the CPU in question every time the list of requests is updated this way or another (there may be other requests coming from kernel code in that list).

CPU idle time governors are expected to regard the minimum of the global (effective) CPU latency limit and the effective resume latency constraint for the given CPU as the upper limit for the exit latency of the idle states that they are allowed to select for that CPU. They should never select any idle states with exit latency beyond that limit.

### Idle States Control Via Kernel Command Line

In addition to the `sysfs` interface allowing individual idle states to be disabled for individual CPUs, there are kernel command line parameters affecting CPU idle time management.

The `cpuidle.off=1` kernel command line option can be used to disable the CPU idle time management entirely. It does not prevent the idle loop from running on idle CPUs, but it prevents the CPU idle time governors and drivers from being invoked. If it is added to the kernel command line, the idle loop will ask the hardware to enter idle states on idle CPUs via the CPU architecture support code that is expected to provide a default mechanism for this purpose. That default mechanism usually is the least common denominator for all of the processors implementing the architecture (i.e. CPU instruction set) in question, however, so it is rather crude and not very energy-efficient. For this reason, it is not recommended for production use.

The `cpuidle.governor=` kernel command line switch allows the CPUIdle governor to use to be specified. It has to be appended with a string matching the name of an available governor (e.g. `cpuidle.governor=menu`) and that governor will be used instead of the default one. It is possible to force the menu governor to be used on the systems that use the ladder governor by default this way, for example.

The other kernel command line parameters controlling CPU idle time management described below are only relevant for the x86 architecture and some of them affect Intel processors only.

The x86 architecture support code recognizes three kernel command line options related to CPU idle time management: `idle=poll`, `idle=halt`, and `idle=nomwait`. The first two of them disable the `acpi_idle` and `intel_idle` drivers altogether, which effectively causes the entire CPUIdle subsystem to be disabled and makes the idle loop invoke the architecture support code to deal with idle CPUs. How it does that depends on which of the two parameters is added to the kernel command line. In the `idle=halt` case, the architecture support code will use the HLT instruction of the CPUs (which, as a rule, suspends the execution of the program and causes the hardware to attempt to enter the shallowest available idle state) for this purpose, and if `idle=poll` is used, idle CPUs will execute a more or less lightweight'' sequence of instructions in a tight loop. [Note that using `idle=poll` is somewhat drastic in many cases, as preventing idle CPUs from saving almost any energy at all may not be the only effect of it. For example, on Intel hardware it effectively prevents CPUs from using P-states (see CPU Performance Scaling) that require any number of CPUs in a package to be idle, so it very well may hurt single-thread computations performance as well as energy-efficiency. Thus using it for performance reasons may not be a good idea at all.]

The `idle=nomwait` option disables the `intel_idle` driver and causes `acpi_idle` to be used (as long as all of the information needed by it is there in the system's ACPI tables), but it is not allowed to use the MWAIT instruction of the CPUs to ask the hardware to enter idle states.

In addition to the architecture-level kernel command line options affecting CPU idle time management, there are parameters affecting individual CPUIdle drivers that can be passed to them via the kernel command line. Specifically,

the `intel_idle.max_cstate=<n>` and `processor.max_cstate=<n>` parameters, where `<n>` is an idle state index also used in the name of the given state's directory in `sysfs` (see Representation of Idle States), causes the `intel_idle` and `acpi_idle` drivers, respectively, to discard all of the idle states deeper than idle state `<n>`. In that case, they will never ask for any of those idle states or expose them to the governor. [The behavior of the two drivers is different for `<n>` equal to 0. Adding `intel_idle.max_cstate=0` to the kernel command line disables the `intel_idle` driver and allows `acpi_idle` to be used, whereas `processor.max_cstate=0` is equivalent to `processor.max_cstate=1`. Also, the `acpi_idle` driver is part of the processor kernel module that can be loaded separately and `max_cstate=<n>` can be passed to it as a module parameter when it is loaded.]

### 61.3.2 intel\_idle CPU Idle Time Management Driver

**Copyright** © 2020 Intel Corporation

**Author** Rafael J. Wysocki <[rafael.j.wysocki@intel.com](mailto:rafael.j.wysocki@intel.com)>

#### General Information

`intel_idle` is a part of the CPU idle time management subsystem in the Linux kernel (CPUIdle). It is the default CPU idle time management driver for the Nehalem and later generations of Intel processors, but the level of support for a particular processor model in it depends on whether or not it recognizes that processor model and may also depend on information coming from the platform firmware. [To understand `intel_idle` it is necessary to know how CPUIdle works in general, so this is the time to get familiar with CPU Idle Time Management if you have not done that yet.]

`intel_idle` uses the `MWAIT` instruction to inform the processor that the logical CPU executing it is idle and so it may be possible to put some of the processor's functional blocks into low-power states. That instruction takes two arguments (passed in the `EAX` and `ECX` registers of the target CPU), the first of which, referred to as a hint, can be used by the processor to determine what can be done (for details refer to Intel Software Developer's Manual<sup>1</sup>). Accordingly, `intel_idle` refuses to work with processors in which the support for the `MWAIT` instruction has been disabled (for example, via the platform firmware configuration menu) or which do not support that instruction at all.

`intel_idle` is not modular, so it cannot be unloaded, which means that the only way to pass early-configuration-time parameters to it is via the kernel command line.

---

<sup>1</sup> Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B, <https://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-software-developer-vol-2b-manual.html>

### Enumeration of Idle States

Each MWAIT hint value is interpreted by the processor as a license to reconfigure itself in a certain way in order to save energy. The processor configurations (with reduced power draw) resulting from that are referred to as C-states (in the ACPI terminology) or idle states. The list of meaningful MWAIT hint values and idle states (i.e. low-power configurations of the processor) corresponding to them depends on the processor model and it may also depend on the configuration of the platform.

In order to create a list of available idle states required by the CPUIdle subsystem (see Representation of Idle States in CPU Idle Time Management), `intel_idle` can use two sources of information: static tables of idle states for different processor models included in the driver itself and the ACPI tables of the system. The former are always used if the processor model at hand is recognized by `intel_idle` and the latter are used if that is required for the given processor model (which is the case for all server processor models recognized by `intel_idle`) or if the processor model is not recognized. [There is a module parameter that can be used to make the driver use the ACPI tables with any processor model recognized by it; see below.]

If the ACPI tables are going to be used for building the list of available idle states, `intel_idle` first looks for a `_CST` object under one of the ACPI objects corresponding to the CPUs in the system (refer to the ACPI specification<sup>2</sup> for the description of `_CST` and its output package). Because the CPUIdle subsystem expects that the list of idle states supplied by the driver will be suitable for all of the CPUs handled by it and `intel_idle` is registered as the CPUIdle driver for all of the CPUs in the system, the driver looks for the first `_CST` object returning at least one valid idle state description and such that all of the idle states included in its return package are of the FFH (Functional Fixed Hardware) type, which means that the MWAIT instruction is expected to be used to tell the processor that it can enter one of them. The return package of that `_CST` is then assumed to be applicable to all of the other CPUs in the system and the idle state descriptions extracted from it are stored in a preliminary list of idle states coming from the ACPI tables. [This step is skipped if `intel_idle` is configured to ignore the ACPI tables; see below.]

Next, the first (index 0) entry in the list of available idle states is initialized to represent a “polling idle state” (a pseudo-idle state in which the target CPU continuously fetches and executes instructions), and the subsequent (real) idle state entries are populated as follows.

If the processor model at hand is recognized by `intel_idle`, there is a (static) table of idle state descriptions for it in the driver. In that case, the “internal” table is the primary source of information on idle states and the information from it is copied to the final list of available idle states. If using the ACPI tables for the enumeration of idle states is not required (depending on the processor model), all of the listed idle state are enabled by default (so all of them will be taken into consideration by CPUIdle governors during CPU idle state selection). Otherwise, some of the listed idle states may not be enabled by default if there are no matching entries in the preliminary list of idle states coming from the ACPI tables. In that case user space still can enable them later (on a per-CPU basis) with the help of the `disable` idle state attribute in `sysfs` (see Representation of Idle States in CPU Idle Time

---

<sup>2</sup> Advanced Configuration and Power Interface (ACPI) Specification, <https://uefi.org/specifications>

Management). This basically means that the idle states “known” to the driver may not be enabled by default if they have not been exposed by the platform firmware (through the ACPI tables).

If the given processor model is not recognized by `intel_idle`, but it supports MWAIT, the preliminary list of idle states coming from the ACPI tables is used for building the final list that will be supplied to the `CPUIdle` core during driver registration. For each idle state in that list, the description, MWAIT hint and exit latency are copied to the corresponding entry in the final list of idle states. The name of the idle state represented by it (to be returned by the name idle state attribute in `sysfs`) is “CX\_ACPI”, where X is the index of that idle state in the final list (note that the minimum value of X is 1, because 0 is reserved for the “polling” state), and its target residency is based on the exit latency value. Specifically, for C1-type idle states the exit latency value is also used as the target residency (for compatibility with the majority of the “internal” tables of idle states for various processor models recognized by `intel_idle`) and for the other idle state types (C2 and C3) the target residency value is 3 times the exit latency (again, that is because it reflects the target residency to exit latency ratio in the majority of cases for the processor models recognized by `intel_idle`). All of the idle states in the final list are enabled by default in this case.

## Initialization

The initialization of `intel_idle` starts with checking if the kernel command line options forbid the use of the MWAIT instruction. If that is the case, an error code is returned right away.

The next step is to check whether or not the processor model is known to the driver, which determines the idle states enumeration method (see above), and whether or not the processor supports MWAIT (the initialization fails if that is not the case). Then, the MWAIT support in the processor is enumerated through `CPUID` and the driver initialization fails if the level of support is not as expected (for example, if the total number of MWAIT substates returned is 0).

Next, if the driver is not configured to ignore the ACPI tables (see below), the idle states information provided by the platform firmware is extracted from them.

Then, `CPUIdle` device objects are allocated for all CPUs and the list of available idle states is created as explained above.

Finally, `intel_idle` is registered with the help of `cpuidle_register_driver()` as the `CPUIdle` driver for all CPUs in the system and a CPU online callback for configuring individual CPUs is registered via `cpuhp_setup_state()`, which (among other things) causes the callback routine to be invoked for all of the CPUs present in the system at that time (each CPU executes its own instance of the callback routine). That routine registers a `CPUIdle` device for the CPU running it (which enables the `CPUIdle` subsystem to operate that CPU) and optionally performs some CPU-specific initialization actions that may be required for the given processor model.

### Kernel Command Line Options and Module Parameters

The x86 architecture support code recognizes three kernel command line options related to CPU idle time management: `idle=poll`, `idle=halt`, and `idle=nomwait`. If any of them is present in the kernel command line, the `MWAIT` instruction is not allowed to be used, so the initialization of `intel_idle` will fail.

Apart from that there are four module parameters recognized by `intel_idle` itself that can be set via the kernel command line (they cannot be updated via `sysfs`, so that is the only way to change their values).

The `max_cstate` parameter value is the maximum idle state index in the list of idle states supplied to the `CPUIde` core during the registration of the driver. It is also the maximum number of regular (non-polling) idle states that can be used by `intel_idle`, so the enumeration of idle states is terminated after finding that number of usable idle states (the other idle states that potentially might have been used if `max_cstate` had been greater are not taken into consideration at all). Setting `max_cstate` can prevent `intel_idle` from exposing idle states that are regarded as “too deep” for some reason to the `CPUIde` core, but it does so by making them effectively invisible until the system is shut down and started again which may not always be desirable. In practice, it is only really necessary to do that if the idle states in question cannot be enabled during system startup, because in the working state of the system the CPU power management quality of service (PM QoS) feature can be used to prevent `CPUIde` from touching those idle states even if they have been enumerated (see Power Management Quality of Service for CPUs in CPU Idle Time Management). Setting `max_cstate` to 0 causes the `intel_idle` initialization to fail.

The `no_acpi` and `use_acpi` module parameters (recognized by `intel_idle` if the kernel has been configured with ACPI support) can be set to make the driver ignore the system’s ACPI tables entirely or use them for all of the recognized processor models, respectively (they both are unset by default and `use_acpi` has no effect if `no_acpi` is set).

The value of the `states_off` module parameter (0 by default) represents a list of idle states to be disabled by default in the form of a bitmask.

Namely, the positions of the bits that are set in the `states_off` value are the indices of idle states to be disabled by default (as reflected by the names of the corresponding idle state directories in `sysfs`, `state0`, `state1` … `state<i>` …, where `<i>` is the index of the given idle state; see Representation of Idle States in CPU Idle Time Management).

For example, if `states_off` is equal to 3, the driver will disable idle states 0 and 1 by default, and if it is equal to 8, idle state 3 will be disabled by default and so on (bit positions beyond the maximum idle state index are ignored).

The idle states disabled this way can be enabled (on a per-CPU basis) from user space via `sysfs`.

## Core and Package Levels of Idle States

Typically, in a processor supporting the MWAIT instruction there are (at least) two levels of idle states (or C-states). One level, referred to as “core C-states”, covers individual cores in the processor, whereas the other level, referred to as “package C-states”, covers the entire processor package and it may also involve other components of the system (GPUs, memory controllers, I/O hubs etc.).

Some of the MWAIT hint values allow the processor to use core C-states only (most importantly, that is the case for the MWAIT hint value corresponding to the C1 idle state), but the majority of them give it a license to put the target core (i.e. the core containing the logical CPU executing MWAIT with the given hint value) into a specific core C-state and then (if possible) to enter a specific package C-state at the deeper level. For example, the MWAIT hint value representing the C3 idle state allows the processor to put the target core into the low-power state referred to as “core C3” (or CC3), which happens if all of the logical CPUs (SMT siblings) in that core have executed MWAIT with the C3 hint value (or with a hint value representing a deeper idle state), and in addition to that (in the majority of cases) it gives the processor a license to put the entire package (possibly including some non-CPU components such as a GPU or a memory controller) into the low-power state referred to as “package C3” (or PC3), which happens if all of the cores have gone into the CC3 state and (possibly) some additional conditions are satisfied (for instance, if the GPU is covered by PC3, it may be required to be in a certain GPU-specific low-power state for PC3 to be reachable).

As a rule, there is no simple way to make the processor use core C-states only if the conditions for entering the corresponding package C-states are met, so the logical CPU executing MWAIT with a hint value that is not core-level only (like for C1) must always assume that this may cause the processor to enter a package C-state. [That is why the exit latency and target residency values corresponding to the majority of MWAIT hint values in the “internal” tables of idle states in `intel_idle` reflect the properties of package C-states.] If using package C-states is not desirable at all, either PM QoS or the `max_cstate` module parameter of `intel_idle` described above must be used to restrict the range of permissible idle states to the ones with core-level only MWAIT hint values (like C1).

## References

### 61.3.3 CPU Performance Scaling

**Copyright** © 2017 Intel Corporation

**Author** Rafael J. Wysocki <[rafael.j.wysocki@intel.com](mailto:rafael.j.wysocki@intel.com)>

### The Concept of CPU Performance Scaling

The majority of modern processors are capable of operating in a number of different clock frequency and voltage configurations, often referred to as Operating Performance Points or P-states (in ACPI terminology). As a rule, the higher the clock frequency and the higher the voltage, the more instructions can be retired by the CPU over a unit of time, but also the higher the clock frequency and the higher the voltage, the more energy is consumed over a unit of time (or the more power is drawn) by the CPU in the given P-state. Therefore there is a natural tradeoff between the CPU capacity (the number of instructions that can be executed over a unit of time) and the power drawn by the CPU.

In some situations it is desirable or even necessary to run the program as fast as possible and then there is no reason to use any P-states different from the highest one (i.e. the highest-performance frequency/voltage configuration available). In some other cases, however, it may not be necessary to execute instructions so quickly and maintaining the highest available CPU capacity for a relatively long time without utilizing it entirely may be regarded as wasteful. It also may not be physically possible to maintain maximum CPU capacity for too long for thermal or power supply capacity reasons or similar. To cover those cases, there are hardware interfaces allowing CPUs to be switched between different frequency/voltage configurations or (in the ACPI terminology) to be put into different P-states.

Typically, they are used along with algorithms to estimate the required CPU capacity, so as to decide which P-states to put the CPUs into. Of course, since the utilization of the system generally changes over time, that has to be done repeatedly on a regular basis. The activity by which this happens is referred to as CPU performance scaling or CPU frequency scaling (because it involves adjusting the CPU clock frequency).

### CPU Performance Scaling in Linux

The Linux kernel supports CPU performance scaling by means of the CPUFreq (CPU Frequency scaling) subsystem that consists of three layers of code: the core, scaling governors and scaling drivers.

The CPUFreq core provides the common code infrastructure and user space interfaces for all platforms that support CPU performance scaling. It defines the basic framework in which the other components operate.

Scaling governors implement algorithms to estimate the required CPU capacity. As a rule, each governor implements one, possibly parametrized, scaling algorithm.

Scaling drivers talk to the hardware. They provide scaling governors with information on the available P-states (or P-state ranges in some cases) and access platform-specific hardware interfaces to change CPU P-states as requested by scaling governors.

In principle, all available scaling governors can be used with every scaling driver. That design is based on the observation that the information used by performance scaling algorithms for P-state selection can be represented in a platform-independent form in the majority of cases, so it should be possible to use the same performance scaling algorithm implemented in exactly the same way regardless

of which scaling driver is used. Consequently, the same set of scaling governors should be suitable for every supported platform.

However, that observation may not hold for performance scaling algorithms based on information provided by the hardware itself, for example through feedback registers, as that information is typically specific to the hardware interface it comes from and may not be easily represented in an abstract, platform-independent way. For this reason, CPUFreq allows scaling drivers to bypass the governor layer and implement their own performance scaling algorithms. That is done by the `intel_pstate` scaling driver.

### **CPUFreq Policy Objects**

In some cases the hardware interface for P-state control is shared by multiple CPUs. That is, for example, the same register (or set of registers) is used to control the P-state of multiple CPUs at the same time and writing to it affects all of those CPUs simultaneously.

Sets of CPUs sharing hardware P-state control interfaces are represented by CPUFreq as `struct cpufreq_policy` objects. For consistency, `struct cpufreq_policy` is also used when there is only one CPU in the given set.

The CPUFreq core maintains a pointer to a `struct cpufreq_policy` object for every CPU in the system, including CPUs that are currently offline. If multiple CPUs share the same hardware P-state control interface, all of the pointers corresponding to them point to the same `struct cpufreq_policy` object.

CPUFreq uses `struct cpufreq_policy` as its basic data type and the design of its user space interface is based on the policy concept.

### **CPU Initialization**

First of all, a scaling driver has to be registered for CPUFreq to work. It is only possible to register one scaling driver at a time, so the scaling driver is expected to be able to handle all CPUs in the system.

The scaling driver may be registered before or after CPU registration. If CPUs are registered earlier, the driver core invokes the CPUFreq core to take a note of all of the already registered CPUs during the registration of the scaling driver. In turn, if any CPUs are registered after the registration of the scaling driver, the CPUFreq core will be invoked to take note of them at their registration time.

In any case, the CPUFreq core is invoked to take note of any logical CPU it has not seen so far as soon as it is ready to handle that CPU. [Note that the logical CPU may be a physical single-core processor, or a single core in a multicore processor, or a hardware thread in a physical processor or processor core. In what follows “CPU” always means “logical CPU” unless explicitly stated otherwise and the word “processor” is used to refer to the physical part possibly including multiple logical CPUs.]

Once invoked, the CPUFreq core checks if the policy pointer is already set for the given CPU and if so, it skips the policy object creation. Otherwise, a new policy

object is created and initialized, which involves the creation of a new policy directory in `sysfs`, and the policy pointer corresponding to the given CPU is set to the new policy object's address in memory.

Next, the scaling driver's `->init()` callback is invoked with the policy pointer of the new CPU passed to it as the argument. That callback is expected to initialize the performance scaling hardware interface for the given CPU (or, more precisely, for the set of CPUs sharing the hardware interface it belongs to, represented by its policy object) and, if the policy object it has been called for is new, to set parameters of the policy, like the minimum and maximum frequencies supported by the hardware, the table of available frequencies (if the set of supported P-states is not a continuous range), and the mask of CPUs that belong to the same policy (including both online and offline CPUs). That mask is then used by the core to populate the policy pointers for all of the CPUs in it.

The next major initialization step for a new policy object is to attach a scaling governor to it (to begin with, that is the default scaling governor determined by the kernel configuration, but it may be changed later via `sysfs`). First, a pointer to the new policy object is passed to the governor's `->init()` callback which is expected to initialize all of the data structures necessary to handle the given policy and, possibly, to add a governor `sysfs` interface to it. Next, the governor is started by invoking its `->start()` callback.

That callback is expected to register per-CPU utilization update callbacks for all of the online CPUs belonging to the given policy with the CPU scheduler. The utilization update callbacks will be invoked by the CPU scheduler on important events, like task enqueue and dequeue, on every iteration of the scheduler tick or generally whenever the CPU utilization may change (from the scheduler's perspective). They are expected to carry out computations needed to determine the P-state to use for the given policy going forward and to invoke the scaling driver to make changes to the hardware in accordance with the P-state selection. The scaling driver may be invoked directly from scheduler context or asynchronously, via a kernel thread or workqueue, depending on the configuration and capabilities of the scaling driver and the governor.

Similar steps are taken for policy objects that are not new, but were “inactive” previously, meaning that all of the CPUs belonging to them were offline. The only practical difference in that case is that the `CPUFreq` core will attempt to use the scaling governor previously used with the policy that became “inactive” (and is re-initialized now) instead of the default governor.

In turn, if a previously offline CPU is being brought back online, but some other CPUs sharing the policy object with it are online already, there is no need to re-initialize the policy object at all. In that case, it only is necessary to restart the scaling governor so that it can take the new online CPU into account. That is achieved by invoking the governor's `->stop` and `->start()` callbacks, in this order, for the entire policy.

As mentioned before, the `intel_pstate` scaling driver bypasses the scaling governor layer of `CPUFreq` and provides its own P-state selection algorithms. Consequently, if `intel_pstate` is used, scaling governors are not attached to new policy objects. Instead, the driver's `->setpolicy()` callback is invoked to register per-CPU utilization update callbacks for each policy. These callbacks are invoked by the CPU scheduler in the same way as for scaling governors, but in the `intel_pstate` case

they both determine the P-state to use and change the hardware configuration accordingly in one go from scheduler context.

The policy objects created during CPU initialization and other data structures associated with them are torn down when the scaling driver is unregistered (which happens when the kernel module containing it is unloaded, for example) or when the last CPU belonging to the given policy is unregistered.

### Policy Interface in sysfs

During the initialization of the kernel, the CPUFreq core creates a sysfs directory (kobject) called `cpufreq` under `/sys/devices/system/cpu/`.

That directory contains a `policyX` subdirectory (where `X` represents an integer number) for every policy object maintained by the CPUFreq core. Each `policyX` directory is pointed to by `cpufreq` symbolic links under `/sys/devices/system/cpu/cpuY/` (where `Y` represents an integer that may be different from the one represented by `X`) for all of the CPUs associated with (or belonging to) the given policy. The `policyX` directories in `/sys/devices/system/cpu/cpufreq` each contain policy-specific attributes (files) to control CPUFreq behavior for the corresponding policy objects (that is, for all of the CPUs associated with them).

Some of those attributes are generic. They are created by the CPUFreq core and their behavior generally does not depend on what scaling driver is in use and what scaling governor is attached to the given policy. Some scaling drivers also add driver-specific attributes to the policy directories in sysfs to control policy-specific aspects of driver behavior.

The generic attributes under `/sys/devices/system/cpu/cpufreq/policyX/` are the following:

**affected\_cpus** List of online CPUs belonging to this policy (i.e. sharing the hardware performance scaling interface represented by the `policyX` policy object).

**bios\_limit** If the platform firmware (BIOS) tells the OS to apply an upper limit to CPU frequencies, that limit will be reported through this attribute (if present).

The existence of the limit may be a result of some (often unintentional) BIOS settings, restrictions coming from a service processor or another BIOS/HW-based mechanisms.

This does not cover ACPI thermal limitations which can be discovered through a generic thermal driver.

This attribute is not present if the scaling driver in use does not support it.

**cpuinfo\_cur\_freq** Current frequency of the CPUs belonging to this policy as obtained from the hardware (in KHz).

This is expected to be the frequency the hardware actually runs at. If that frequency cannot be determined, this attribute should not be present.

**cpuinfo\_max\_freq** Maximum possible operating frequency the CPUs belonging to this policy can run at (in kHz).

**cpuinfo\_min\_freq** Minimum possible operating frequency the CPUs belonging to this policy can run at (in kHz).

**cpuinfo\_transition\_latency** The time it takes to switch the CPUs belonging to this policy from one P-state to another, in nanoseconds.

If unknown or if known to be so high that the scaling driver does not work with the ondemand governor, -1 (CPUFREQ\_ETERNAL) will be returned by reads from this attribute.

**related\_cpus** List of all (online and offline) CPUs belonging to this policy.

**scaling\_available\_governors** List of CPUFreq scaling governors present in the kernel that can be attached to this policy or (if the intel\_pstate scaling driver is in use) list of scaling algorithms provided by the driver that can be applied to this policy.

[Note that some governors are modular and it may be necessary to load a kernel module for the governor held by it to become available and be listed by this attribute.]

**scaling\_cur\_freq** Current frequency of all of the CPUs belonging to this policy (in kHz).

In the majority of cases, this is the frequency of the last P-state requested by the scaling driver from the hardware using the scaling interface provided by it, which may or may not reflect the frequency the CPU is actually running at (due to hardware design and other limitations).

Some architectures (e.g. x86) may attempt to provide information more precisely reflecting the current CPU frequency through this attribute, but that still may not be the exact current CPU frequency as seen by the hardware at the moment.

**scaling\_driver** The scaling driver currently in use.

**scaling\_governor** The scaling governor currently attached to this policy or (if the intel\_pstate scaling driver is in use) the scaling algorithm provided by the driver that is currently applied to this policy.

This attribute is read-write and writing to it will cause a new scaling governor to be attached to this policy or a new scaling algorithm provided by the scaling driver to be applied to it (in the intel\_pstate case), as indicated by the string written to this attribute (which must be one of the names listed by the scaling\_available\_governors attribute described above).

**scaling\_max\_freq** Maximum frequency the CPUs belonging to this policy are allowed to be running at (in kHz).

This attribute is read-write and writing a string representing an integer to it will cause a new limit to be set (it must not be lower than the value of the scaling\_min\_freq attribute).

**scaling\_min\_freq** Minimum frequency the CPUs belonging to this policy are allowed to be running at (in kHz).

This attribute is read-write and writing a string representing a non-negative integer to it will cause a new limit to be set (it must not be higher than the value of the scaling\_max\_freq attribute).

**scaling\_setspeed** This attribute is functional only if the userspace scaling governor is attached to the given policy.

It returns the last frequency requested by the governor (in kHz) or can be written to in order to set a new frequency for the policy.

## Generic Scaling Governors

CPUFreq provides generic scaling governors that can be used with all scaling drivers. As stated before, each of them implements a single, possibly parametrized, performance scaling algorithm.

Scaling governors are attached to policy objects and different policy objects can be handled by different scaling governors at the same time (although that may lead to suboptimal results in some cases).

The scaling governor for a given policy object can be changed at any time with the help of the `scaling_governor` policy attribute in `sysfs`.

Some governors expose `sysfs` attributes to control or fine-tune the scaling algorithms implemented by them. Those attributes, referred to as governor tunables, can be either global (system-wide) or per-policy, depending on the scaling driver in use. If the driver requires governor tunables to be per-policy, they are located in a subdirectory of each policy directory. Otherwise, they are located in a subdirectory under `/sys/devices/system/cpu/cpufreq/`. In either case the name of the subdirectory containing the governor tunables is the name of the governor providing them.

### performance

When attached to a policy object, this governor causes the highest frequency, within the `scaling_max_freq` policy limit, to be requested for that policy.

The request is made once at that time the governor for the policy is set to performance and whenever the `scaling_max_freq` or `scaling_min_freq` policy limits change after that.

### powersave

When attached to a policy object, this governor causes the lowest frequency, within the `scaling_min_freq` policy limit, to be requested for that policy.

The request is made once at that time the governor for the policy is set to powersave and whenever the `scaling_max_freq` or `scaling_min_freq` policy limits change after that.

### userspace

This governor does not do anything by itself. Instead, it allows user space to set the CPU frequency for the policy it is attached to by writing to the `scaling_setspeed` attribute of that policy.

### schedutil

This governor uses CPU utilization data available from the CPU scheduler. It generally is regarded as a part of the CPU scheduler, so it can access the scheduler's internal data structures directly.

It runs entirely in scheduler context, although in some cases it may need to invoke the scaling driver asynchronously when it decides that the CPU frequency should be changed for a given policy (that depends on whether or not the driver is capable of changing the CPU frequency from scheduler context).

The actions of this governor for a particular CPU depend on the scheduling class invoking its utilization update callback for that CPU. If it is invoked by the RT or deadline scheduling classes, the governor will increase the frequency to the allowed maximum (that is, the `scaling_max_freq` policy limit). In turn, if it is invoked by the CFS scheduling class, the governor will use the Per-Entity Load Tracking (PELT) metric for the root control group of the given CPU as the CPU utilization estimate (see the Per-entity load tracking LWN.net article<sup>1</sup> for a description of the PELT mechanism). Then, the new CPU frequency to apply is computed in accordance with the formula

$$f = 1.25 * f_0 * util / max$$

where `util` is the PELT number, `max` is the theoretical maximum of `util`, and `f_0` is either the maximum possible CPU frequency for the given policy (if the PELT number is frequency-invariant), or the current CPU frequency (otherwise).

This governor also employs a mechanism allowing it to temporarily bump up the CPU frequency for tasks that have been waiting on I/O most recently, called “IO-wait boosting”. That happens when the `SCHED_CPUFREQ_IOWAIT` flag is passed by the scheduler to the governor callback which causes the frequency to go up to the allowed maximum immediately and then draw back to the value returned by the above formula over time.

This governor exposes only one tunable:

**rate\_limit\_us** Minimum time (in microseconds) that has to pass between two consecutive runs of governor computations (default: 1000 times the scaling driver's transition latency).

The purpose of this tunable is to reduce the scheduler context overhead of the governor which might be excessive without it.

This governor generally is regarded as a replacement for the older `ondemand` and `conservative` governors (described below), as it is simpler and more tightly integrated with the CPU scheduler, its overhead in terms of CPU context switches and similar is less significant, and it uses the scheduler's own CPU utilization metric,

---

<sup>1</sup> Jonathan Corbet, Per-entity load tracking, <https://lwn.net/Articles/531853/>

so in principle its decisions should not contradict the decisions made by the other parts of the scheduler.

## ondemand

This governor uses CPU load as a CPU frequency selection metric.

In order to estimate the current CPU load, it measures the time elapsed between consecutive invocations of its worker routine and computes the fraction of that time in which the given CPU was not idle. The ratio of the non-idle (active) time to the total CPU time is taken as an estimate of the load.

If this governor is attached to a policy shared by multiple CPUs, the load is estimated for all of them and the greatest result is taken as the load estimate for the entire policy.

The worker routine of this governor has to run in process context, so it is invoked asynchronously (via a workqueue) and CPU P-states are updated from there if necessary. As a result, the scheduler context overhead from this governor is minimum, but it causes additional CPU context switches to happen relatively often and the CPU P-state updates triggered by it can be relatively irregular. Also, it affects its own CPU load metric by running code that reduces the CPU idle time (even though the CPU idle time is only reduced very slightly by it).

It generally selects CPU frequencies proportional to the estimated load, so that the value of the `cpuinfo_max_freq` policy attribute corresponds to the load of 1 (or 100%), and the value of the `cpuinfo_min_freq` policy attribute corresponds to the load of 0, unless when the load exceeds a (configurable) speedup threshold, in which case it will go straight for the highest frequency it is allowed to use (the `scaling_max_freq` policy limit).

This governor exposes the following tunables:

**sampling\_rate** This is how often the governor's worker routine should run, in microseconds.

Typically, it is set to values of the order of 10000 (10 ms). Its default value is equal to the value of `cpuinfo_transition_latency` for each policy this governor is attached to (but since the unit here is greater by 1000, this means that the time represented by `sampling_rate` is 1000 times greater than the transition latency by default).

If this tunable is per-policy, the following shell command sets the time represented by it to be 750 times as high as the transition latency:

```
# echo `((${cat cpuinfo_transition_latency} * 750 / 1000)) >_
↪ondemand/sampling_rate
```

**up\_threshold** If the estimated CPU load is above this value (in percent), the governor will set the frequency to the maximum value allowed for the policy. Otherwise, the selected frequency will be proportional to the estimated CPU load.

**ignore\_nice\_load** If set to 1 (default 0), it will cause the CPU load estimation code to treat the CPU time spent on executing tasks with “nice” levels greater than

0 as CPU idle time.

This may be useful if there are tasks in the system that should not be taken into account when deciding what frequency to run the CPUs at. Then, to make that happen it is sufficient to increase the “nice” level of those tasks above 0 and set this attribute to 1.

**sampling\_down\_factor** Temporary multiplier, between 1 (default) and 100 inclusive, to apply to the `sampling_rate` value if the CPU load goes above `up_threshold`.

This causes the next execution of the governor’s worker routine (after setting the frequency to the allowed maximum) to be delayed, so the frequency stays at the maximum level for a longer time.

Frequency fluctuations in some bursty workloads may be avoided this way at the cost of additional energy spent on maintaining the maximum CPU capacity.

**powersave\_bias** Reduction factor to apply to the original frequency target of the governor (including the maximum value used when the `up_threshold` value is exceeded by the estimated CPU load) or sensitivity threshold for the AMD frequency sensitivity powersave bias driver (`drivers/cpufreq/amd_freq_sensitivity.c`), between 0 and 1000 inclusive.

If the AMD frequency sensitivity powersave bias driver is not loaded, the effective frequency to apply is given by

$$f * (1 - \text{powersave\_bias} / 1000)$$

where `f` is the governor’s original frequency target. The default value of this attribute is 0 in that case.

If the AMD frequency sensitivity powersave bias driver is loaded, the value of this attribute is 400 by default and it is used in a different way.

On Family 16h (and later) AMD processors there is a mechanism to get a measured workload sensitivity, between 0 and 100% inclusive, from the hardware. That value can be used to estimate how the performance of the workload running on a CPU will change in response to frequency changes.

The performance of a workload with the sensitivity of 0 (memory-bound or IO-bound) is not expected to increase at all as a result of increasing the CPU frequency, whereas workloads with the sensitivity of 100% (CPU-bound) are expected to perform much better if the CPU frequency is increased.

If the workload sensitivity is less than the threshold represented by the `powersave_bias` value, the sensitivity powersave bias driver will cause the governor to select a frequency lower than its original target, so as to avoid over-provisioning workloads that will not benefit from running at higher CPU frequencies.

## **conservative**

This governor uses CPU load as a CPU frequency selection metric.

It estimates the CPU load in the same way as the ondemand governor described above, but the CPU frequency selection algorithm implemented by it is different.

Namely, it avoids changing the frequency significantly over short time intervals which may not be suitable for systems with limited power supply capacity (e.g. battery-powered). To achieve that, it changes the frequency in relatively small steps, one step at a time, up or down - depending on whether or not a (configurable) threshold has been exceeded by the estimated CPU load.

This governor exposes the following tunables:

**freq\_step** Frequency step in percent of the maximum frequency the governor is allowed to set (the `scaling_max_freq` policy limit), between 0 and 100 (5 by default).

This is how much the frequency is allowed to change in one go. Setting it to 0 will cause the default frequency step (5 percent) to be used and setting it to 100 effectively causes the governor to periodically switch the frequency between the `scaling_min_freq` and `scaling_max_freq` policy limits.

**down\_threshold** Threshold value (in percent, 20 by default) used to determine the frequency change direction.

If the estimated CPU load is greater than this value, the frequency will go up (by `freq_step`). If the load is less than this value (and the `sampling_down_factor` mechanism is not in effect), the frequency will go down. Otherwise, the frequency will not be changed.

**sampling\_down\_factor** Frequency decrease deferral factor, between 1 (default) and 10 inclusive.

It effectively causes the frequency to go down `sampling_down_factor` times slower than it ramps up.

## **Frequency Boost Support**

### **Background**

Some processors support a mechanism to raise the operating frequency of some cores in a multicore package temporarily (and above the sustainable frequency threshold for the whole package) under certain conditions, for example if the whole chip is not fully utilized and below its intended thermal or power budget.

Different names are used by different vendors to refer to this functionality. For Intel processors it is referred to as “Turbo Boost”, AMD calls it “Turbo-Core” or (in technical documentation) “Core Performance Boost” and so on. As a rule, it also is implemented differently by different vendors. The simple term “frequency boost” is used here for brevity to refer to all of those implementations.

The frequency boost mechanism may be either hardware-based or software-based. If it is hardware-based (e.g. on x86), the decision to trigger the boosting is made

by the hardware (although in general it requires the hardware to be put into a special state in which it can control the CPU frequency within certain limits). If it is software-based (e.g. on ARM), the scaling driver decides whether or not to trigger boosting and when to do that.

### The boost File in sysfs

This file is located under `/sys/devices/system/cpu/cpufreq/` and controls the “boost” setting for the whole system. It is not present if the underlying scaling driver does not support the frequency boost mechanism (or supports it, but provides a driver-specific interface for controlling it, like `intel_pstate`).

If the value in this file is 1, the frequency boost mechanism is enabled. This means that either the hardware can be put into states in which it is able to trigger boosting (in the hardware-based case), or the software is allowed to trigger boosting (in the software-based case). It does not mean that boosting is actually in use at the moment on any CPUs in the system. It only means a permission to use the frequency boost mechanism (which still may never be used for other reasons).

If the value in this file is 0, the frequency boost mechanism is disabled and cannot be used at all.

The only values that can be written to this file are 0 and 1.

### Rationale for Boost Control Knob

The frequency boost mechanism is generally intended to help to achieve optimum CPU performance on time scales below software resolution (e.g. below the scheduler tick interval) and it is demonstrably suitable for many workloads, but it may lead to problems in certain situations.

For this reason, many systems make it possible to disable the frequency boost mechanism in the platform firmware (BIOS) setup, but that requires the system to be restarted for the setting to be adjusted as desired, which may not be practical at least in some cases. For example:

1. Boosting means overclocking the processor, although under controlled conditions. Generally, the processor’s energy consumption increases as a result of increasing its frequency and voltage, even temporarily. That may not be desirable on systems that switch to power sources of limited capacity, such as batteries, so the ability to disable the boost mechanism while the system is running may help there (but that depends on the workload too).
2. In some situations deterministic behavior is more important than performance or energy consumption (or both) and the ability to disable boosting while the system is running may be useful then.
3. To examine the impact of the frequency boost mechanism itself, it is useful to be able to run tests with and without boosting, preferably without restarting the system in the meantime.
4. Reproducible results are important when running benchmarks. Since the boosting functionality depends on the load of the whole package, single-

thread performance may vary because of it which may lead to unreproducible results sometimes. That can be avoided by disabling the frequency boost mechanism before running benchmarks sensitive to that issue.

### Legacy AMD cpb Knob

The AMD powernow-k8 scaling driver supports a `sysfs` knob very similar to the global boost one. It is used for disabling/enabling the “Core Performance Boost” feature of some AMD processors.

If present, that knob is located in every `CPUFreq` policy directory in `sysfs` (`/sys/devices/system/cpu/cpufreq/policyX/`) and is called `cpb`, which indicates a more fine grained control interface. The actual implementation, however, works on the system-wide basis and setting that knob for one policy causes the same value of it to be set for all of the other policies at the same time.

That knob is still supported on AMD processors that support its underlying hardware feature, but it may be configured out of the kernel (via the `CONFIG_X86_ACPI_CPUFREQ_CPB` configuration option) and the global boost knob is present regardless. Thus it is always possible use the `boost` knob instead of the `cpb` one which is highly recommended, as that is more consistent with what all of the other systems do (and the `cpb` knob may not be supported any more in the future).

The `cpb` knob is never present for any processors without the underlying hardware feature (e.g. all Intel ones), even if the `CONFIG_X86_ACPI_CPUFREQ_CPB` configuration option is set.

## References

### 61.3.4 intel\_pstate CPU Performance Scaling Driver

**Copyright** © 2017 Intel Corporation

**Author** Rafael J. Wysocki <[rafael.j.wysocki@intel.com](mailto:rafael.j.wysocki@intel.com)>

## General Information

`intel_pstate` is a part of the CPU performance scaling subsystem in the Linux kernel (`CPUFreq`). It is a scaling driver for the Sandy Bridge and later generations of Intel processors. Note, however, that some of those processors may not be supported. [To understand `intel_pstate` it is necessary to know how `CPUFreq` works in general, so this is the time to read CPU Performance Scaling if you have not done that yet.]

For the processors supported by `intel_pstate`, the P-state concept is broader than just an operating frequency or an operating performance point (see the LinuxCon Europe 2015 presentation by Kristen Accardi<sup>1</sup> for more information about that). For this reason, the representation of P-states used by `intel_pstate` internally

---

<sup>1</sup> Kristen Accardi, Balancing Power and Performance in the Linux Kernel, [https://events.static.linuxfound.org/sites/events/files/slides/LinuxConEurope\\_2015.pdf](https://events.static.linuxfound.org/sites/events/files/slides/LinuxConEurope_2015.pdf)

follows the hardware specification (for details refer to Intel Software Developer's Manual<sup>2</sup>). However, the CPUFreq core uses frequencies for identifying operating performance points of CPUs and frequencies are involved in the user space interface exposed by it, so `intel_pstate` maps its internal representation of P-states to frequencies too (fortunately, that mapping is unambiguous). At the same time, it would not be practical for `intel_pstate` to supply the CPUFreq core with a table of available frequencies due to the possible size of it, so the driver does not do that. Some functionality of the core is limited by that.

Since the hardware P-state selection interface used by `intel_pstate` is available at the logical CPU level, the driver always works with individual CPUs. Consequently, if `intel_pstate` is in use, every CPUFreq policy object corresponds to one logical CPU and CPUFreq policies are effectively equivalent to CPUs. In particular, this means that they become “inactive” every time the corresponding CPU is taken offline and need to be re-initialized when it goes back online.

`intel_pstate` is not modular, so it cannot be unloaded, which means that the only way to pass early-configuration-time parameters to it is via the kernel command line. However, its configuration can be adjusted via `sysfs` to a great extent. In some configurations it even is possible to unregister it via `sysfs` which allows another CPUFreq scaling driver to be loaded and registered (see below).

### Operation Modes

`intel_pstate` can operate in three different modes: in the active mode with or without hardware-managed P-states support and in the passive mode. Which of them will be in effect depends on what kernel command line options are used and on the capabilities of the processor.

#### Active Mode

This is the default operation mode of `intel_pstate` for processors with hardware-managed P-states (HWP) support. If it works in this mode, the `scaling_driver` policy attribute in `sysfs` for all CPUFreq policies contains the string “`intel_pstate`”

In this mode the driver bypasses the scaling governors layer of CPUFreq and provides its own scaling algorithms for P-state selection. Those algorithms can be applied to CPUFreq policies in the same way as generic scaling governors (that is, through the `scaling_governor` policy attribute in `sysfs`). [Note that different P-state selection algorithms may be chosen for different policies, but that is not recommended.]

They are not generic scaling governors, but their names are the same as the names of some of those governors. Moreover, confusingly enough, they generally do not work in the same way as the generic governors they share the names with. For example, the powersave P-state selection algorithm provided by `intel_pstate` is not a counterpart of the generic powersave governor (roughly, it corresponds to the `schedutil` and `ondemand` governors).

---

<sup>2</sup> Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3: System Programming Guide, <https://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-software-developer-system-programming-manual-325384.html>

There are two P-state selection algorithms provided by `intel_pstate` in the active mode: `powersave` and `performance`. The way they both operate depends on whether or not the hardware-managed P-states (HWP) feature has been enabled in the processor and possibly on the processor model.

Which of the P-state selection algorithms is used by default depends on the `CONFIG_CPU_FREQ_DEFAULT_GOV_PERFORMANCE` kernel configuration option. Namely, if that option is set, the `performance` algorithm will be used by default, and the other one will be used by default if it is not set.

### **Active Mode With HWP**

If the processor supports the HWP feature, it will be enabled during the processor initialization and cannot be disabled after that. It is possible to avoid enabling it by passing the `intel_pstate=no_hwp` argument to the kernel in the command line.

If the HWP feature has been enabled, `intel_pstate` relies on the processor to select P-states by itself, but still it can give hints to the processor's internal P-state selection logic. What those hints are depends on which P-state selection algorithm has been applied to the given policy (or to the CPU it corresponds to).

Even though the P-state selection is carried out by the processor automatically, `intel_pstate` registers utilization update callbacks with the CPU scheduler in this mode. However, they are not used for running a P-state selection algorithm, but for periodic updates of the current CPU frequency information to be made available from the `scaling_cur_freq` policy attribute in `sysfs`.

### **HWP + performance**

In this configuration `intel_pstate` will write 0 to the processor's Energy-Performance Preference (EPP) knob (if supported) or its Energy-Performance Bias (EPB) knob (otherwise), which means that the processor's internal P-state selection logic is expected to focus entirely on performance.

This will override the EPP/EPB setting coming from the `sysfs` interface (see `Energy vs Performance Hints` below).

Also, in this configuration the range of P-states available to the processor's internal P-state selection logic is always restricted to the upper boundary (that is, the maximum P-state that the driver is allowed to use).

### **HWP + powersave**

In this configuration `intel_pstate` will set the processor's Energy-Performance Preference (EPP) knob (if supported) or its Energy-Performance Bias (EPB) knob (otherwise) to whatever value it was previously set to via `sysfs` (or whatever default value it was set to by the platform firmware). This usually causes the processor's internal P-state selection logic to be less performance-focused.

### Active Mode Without HWP

This operation mode is optional for processors that do not support the HWP feature or when the `intel_pstate=no_hwp` argument is passed to the kernel in the command line. The active mode is used in those cases if the `intel_pstate=active` argument is passed to the kernel in the command line. In this mode `intel_pstate` may refuse to work with processors that are not recognized by it. [Note that `intel_pstate` will never refuse to work with any processor with the HWP feature enabled.]

In this mode `intel_pstate` registers utilization update callbacks with the CPU scheduler in order to run a P-state selection algorithm, either `powersave` or `performance`, depending on the `scaling_governor` policy setting in `sysfs`. The current CPU frequency information to be made available from the `scaling_cur_freq` policy attribute in `sysfs` is periodically updated by those utilization update callbacks too.

#### performance

Without HWP, this P-state selection algorithm is always the same regardless of the processor model and platform configuration.

It selects the maximum P-state it is allowed to use, subject to limits set via `sysfs`, every time the driver configuration for the given CPU is updated (e.g. via `sysfs`).

This is the default P-state selection algorithm if the `CONFIG_CPU_FREQ_DEFAULT_GOV_PERFORMANCE` kernel configuration option is set.

#### powersave

Without HWP, this P-state selection algorithm is similar to the algorithm implemented by the generic `schedutil` scaling governor except that the utilization metric used by it is based on numbers coming from feedback registers of the CPU. It generally selects P-states proportional to the current CPU utilization.

This algorithm is run by the driver's utilization update callback for the given CPU when it is invoked by the CPU scheduler, but not more often than every 10 ms. Like in the `performance` case, the hardware configuration is not touched if the new P-state turns out to be the same as the current one.

This is the default P-state selection algorithm if the `CONFIG_CPU_FREQ_DEFAULT_GOV_PERFORMANCE` kernel configuration option is not set.

## Passive Mode

This is the default operation mode of `intel_pstate` for processors without hardware-managed P-states (HWP) support. It is always used if the `intel_pstate=passive` argument is passed to the kernel in the command line regardless of whether or not the given processor supports HWP. [Note that the `intel_pstate=no_hwp` setting implies `intel_pstate=passive` if it is used without `intel_pstate=active`.] Like in the active mode without HWP support, in this mode `intel_pstate` may refuse to work with processors that are not recognized by it.

If the driver works in this mode, the `scaling_driver` policy attribute in `sysfs` for all CPUFreq policies contains the string `"intel_cpufreq"`. Then, the driver behaves like a regular CPUFreq scaling driver. That is, it is invoked by generic scaling governors when necessary to talk to the hardware in order to change the P-state of a CPU (in particular, the `schedutil` governor can invoke it directly from scheduler context).

While in this mode, `intel_pstate` can be used with all of the (generic) scaling governors listed by the `scaling_available_governors` policy attribute in `sysfs` (and the P-state selection algorithms described above are not used). Then, it is responsible for the configuration of policy objects corresponding to CPUs and provides the CPUFreq core (and the scaling governors attached to the policy objects) with accurate information on the maximum and minimum operating frequencies supported by the hardware (including the so-called "turbo" frequency ranges). In other words, in the passive mode the entire range of available P-states is exposed by `intel_pstate` to the CPUFreq core. However, in this mode the driver does not register utilization update callbacks with the CPU scheduler and the `scaling_cur_freq` information comes from the CPUFreq core (and is the last frequency selected by the current scaling governor for the given policy).

## Turbo P-states Support

In the majority of cases, the entire range of P-states available to `intel_pstate` can be divided into two sub-ranges that correspond to different types of processor behavior, above and below a boundary that will be referred to as the "turbo threshold" in what follows.

The P-states above the turbo threshold are referred to as "turbo P-states" and the whole sub-range of P-states they belong to is referred to as the "turbo range". These names are related to the Turbo Boost technology allowing a multicore processor to opportunistically increase the P-state of one or more cores if there is enough power to do that and if that is not going to cause the thermal envelope of the processor package to be exceeded.

Specifically, if software sets the P-state of a CPU core within the turbo range (that is, above the turbo threshold), the processor is permitted to take over performance scaling control for that core and put it into turbo P-states of its choice going forward. However, that permission is interpreted differently by different processor generations. Namely, the Sandy Bridge generation of processors will never use any P-states above the last one set by software for the given core, even if it is within the turbo range, whereas all of the later processor generations will take it

as a license to use any P-states from the turbo range, even above the one set by software. In other words, on those processors setting any P-state from the turbo range will enable the processor to put the given core into all turbo P-states up to and including the maximum supported one as it sees fit.

One important property of turbo P-states is that they are not sustainable. More precisely, there is no guarantee that any CPUs will be able to stay in any of those states indefinitely, because the power distribution within the processor package may change over time or the thermal envelope it was designed for might be exceeded if a turbo P-state was used for too long.

In turn, the P-states below the turbo threshold generally are sustainable. In fact, if one of them is set by software, the processor is not expected to change it to a lower one unless in a thermal stress or a power limit violation situation (a higher P-state may still be used if it is set for another CPU in the same package at the same time, for example).

Some processors allow multiple cores to be in turbo P-states at the same time, but the maximum P-state that can be set for them generally depends on the number of cores running concurrently. The maximum turbo P-state that can be set for 3 cores at the same time usually is lower than the analogous maximum P-state for 2 cores, which in turn usually is lower than the maximum turbo P-state that can be set for 1 core. The one-core maximum turbo P-state is thus the maximum supported one overall.

The maximum supported turbo P-state, the turbo threshold (the maximum supported non-turbo P-state) and the minimum supported P-state are specific to the processor model and can be determined by reading the processor's model-specific registers (MSRs). Moreover, some processors support the Configurable TDP (Thermal Design Power) feature and, when that feature is enabled, the turbo threshold effectively becomes a configurable value that can be set by the platform firmware.

Unlike `_PSS` objects in the ACPI tables, `intel_pstate` always exposes the entire range of available P-states, including the whole turbo range, to the `CPUFreq` core and (in the passive mode) to generic scaling governors. This generally causes turbo P-states to be set more often when `intel_pstate` is used relative to ACPI-based CPU performance scaling (see below for more information).

Moreover, since `intel_pstate` always knows what the real turbo threshold is (even if the Configurable TDP feature is enabled in the processor), its `no_turbo` attribute in `sysfs` (described below) should work as expected in all cases (that is, if set to disable turbo P-states, it always should prevent `intel_pstate` from using them).

### Processor Support

To handle a given processor `intel_pstate` requires a number of different pieces of information on it to be known, including:

- The minimum supported P-state.
- The maximum supported non-turbo P-state.
- Whether or not turbo P-states are supported at all.

- The maximum supported one-core turbo P-state (if turbo P-states are supported).
- The scaling formula to translate the driver's internal representation of P-states into frequencies and the other way around.

Generally, ways to obtain that information are specific to the processor model or family. Although it often is possible to obtain all of it from the processor itself (using model-specific registers), there are cases in which hardware manuals need to be consulted to get to it too.

For this reason, there is a list of supported processors in `intel_pstate` and the driver initialization will fail if the detected processor is not in that list, unless it supports the HWP feature. [The interface to obtain all of the information listed above is the same for all of the processors supporting the HWP feature, which is why they all are supported by `intel_pstate`.]

## User Space Interface in `sysfs`

### Global Attributes

`intel_pstate` exposes several global attributes (files) in `sysfs` to control its functionality at the system level. They are located in the `/sys/devices/system/cpu/intel_pstate/` directory and affect all CPUs.

Some of them are not present if the `intel_pstate=per_cpu_perf_limits` argument is passed to the kernel in the command line.

**max\_perf\_pct** Maximum P-state the driver is allowed to set in percent of the maximum supported performance level (the highest supported turbo P-state).

This attribute will not be exposed if the `intel_pstate=per_cpu_perf_limits` argument is present in the kernel command line.

**min\_perf\_pct** Minimum P-state the driver is allowed to set in percent of the maximum supported performance level (the highest supported turbo P-state).

This attribute will not be exposed if the `intel_pstate=per_cpu_perf_limits` argument is present in the kernel command line.

**num\_pstates** Number of P-states supported by the processor (between 0 and 255 inclusive) including both turbo and non-turbo P-states (see Turbo P-states Support).

The value of this attribute is not affected by the `no_turbo` setting described below.

This attribute is read-only.

**turbo\_pct** Ratio of the turbo range size to the size of the entire range of supported P-states, in percent.

This attribute is read-only.

**no\_turbo** If set (equal to 1), the driver is not allowed to set any turbo P-states (see Turbo P-states Support). If unset (equal to 0, which is the default), turbo P-states can be set by the driver. [Note that `intel_pstate` does not support

the general `boost` attribute (supported by some other scaling drivers) which is replaced by this one.]

This attribute does not affect the maximum supported frequency value supplied to the `CPUFreq` core and exposed via the policy interface, but it affects the maximum possible value of per-policy P-state limits (see Interpretation of Policy Attributes below for details).

**hwp\_dynamic\_boost** This attribute is only present if `intel_pstate` works in the active mode with the HWP feature enabled in the processor. If set (equal to 1), it causes the minimum P-state limit to be increased dynamically for a short time whenever a task previously waiting on I/O is selected to run on a given logical CPU (the purpose of this mechanism is to improve performance).

This setting has no effect on logical CPUs whose minimum P-state limit is directly set to the highest non-turbo P-state or above it.

**status** Operation mode of the driver: “active” , “passive” or “off” .

“**active**” The driver is functional and in the active mode.

“**passive**” The driver is functional and in the passive mode.

“**off**” The driver is not functional (it is not registered as a scaling driver with the `CPUFreq` core).

This attribute can be written to in order to change the driver’ s operation mode or to unregister it. The string written to it must be one of the possible values of it and, if successful, the write will cause the driver to switch over to the operation mode represented by that string - or to be unregistered in the “off” case. [Actually, switching over from the active mode to the passive mode or the other way around causes the driver to be unregistered and registered again with a different set of callbacks, so all of its settings (the global as well as the per-policy ones) are then reset to their default values, possibly depending on the target operation mode.]

That only is supported in some configurations, though (for example, if the HWP feature is enabled in the processor, the operation mode of the driver cannot be changed), and if it is not supported in the current configuration, writes to this attribute will fail with an appropriate error.

### Interpretation of Policy Attributes

The interpretation of some `CPUFreq` policy attributes described in CPU Performance Scaling is special with `intel_pstate` as the current scaling driver and it generally depends on the driver’ s operation mode.

First of all, the values of the `cpuinfo_max_freq`, `cpuinfo_min_freq` and `scaling_cur_freq` attributes are produced by applying a processor-specific multiplier to the internal P-state representation used by `intel_pstate`. Also, the values of the `scaling_max_freq` and `scaling_min_freq` attributes are capped by the frequency corresponding to the maximum P-state that the driver is allowed to set.

If the `no_turbo` global attribute is set, the driver is not allowed to use turbo P-states, so the maximum value of `scaling_max_freq` and `scaling_min_freq` is limited to the maximum non-turbo P-state frequency. Accordingly, setting `no_turbo`

causes `scaling_max_freq` and `scaling_min_freq` to go down to that value if they were above it before. However, the old values of `scaling_max_freq` and `scaling_min_freq` will be restored after unsetting `no_turbo`, unless these attributes have been written to after `no_turbo` was set.

If `no_turbo` is not set, the maximum possible value of `scaling_max_freq` and `scaling_min_freq` corresponds to the maximum supported turbo P-state, which also is the value of `cpuinfo_max_freq` in either case.

Next, the following policy attributes have special meaning if `intel_pstate` works in the active mode:

**scaling\_available\_governors** List of P-state selection algorithms provided by `intel_pstate`.

**scaling\_governor** P-state selection algorithm provided by `intel_pstate` currently in use with the given policy.

**scaling\_cur\_freq** Frequency of the average P-state of the CPU represented by the given policy for the time interval between the last two invocations of the driver's utilization update callback by the CPU scheduler for that CPU.

One more policy attribute is present if the HWP feature is enabled in the processor:

**base\_frequency** Shows the base frequency of the CPU. Any frequency above this will be in the turbo frequency range.

The meaning of these attributes in the passive mode is the same as for other scaling drivers.

Additionally, the value of the `scaling_driver` attribute for `intel_pstate` depends on the operation mode of the driver. Namely, it is either “`intel_pstate`” (in the active mode) or “`intel_cpufreq`” (in the passive mode).

## Coordination of P-State Limits

`intel_pstate` allows P-state limits to be set in two ways: with the help of the `max_perf_pct` and `min_perf_pct` global attributes or via the `scaling_max_freq` and `scaling_min_freq` CPUFreq policy attributes. The coordination between those limits is based on the following rules, regardless of the current operation mode of the driver:

1. All CPUs are affected by the global limits (that is, none of them can be requested to run faster than the global maximum and none of them can be requested to run slower than the global minimum).
2. Each individual CPU is affected by its own per-policy limits (that is, it cannot be requested to run faster than its own per-policy maximum and it cannot be requested to run slower than its own per-policy minimum). The effective performance depends on whether the platform supports per core P-states, hyper-threading is enabled and on current performance requests from other CPUs. When platform doesn't support per core P-states, the effective performance can be more than the policy limits set on a CPU, if other CPUs are requesting higher performance at that moment. Even with per core P-states support, when hyper-threading is enabled, if the sibling CPU is requesting

higher performance, the other siblings will get higher performance than their policy limits.

3. The global and per-policy limits can be set independently.

If the HWP feature is enabled in the processor, the resulting effective values are written into its registers whenever the limits change in order to request its internal P-state selection logic to always set P-states within these limits. Otherwise, the limits are taken into account by scaling governors (in the passive mode) and by the driver every time before setting a new P-state for a CPU.

Additionally, if the `intel_pstate=per_cpu_perf_limits` command line argument is passed to the kernel, `max_perf_pct` and `min_perf_pct` are not exposed at all and the only way to set the limits is by using the policy attributes.

### Energy vs Performance Hints

If `intel_pstate` works in the active mode with the HWP feature enabled in the processor, additional attributes are present in every `CPUFreq` policy directory in `sysfs`. They are intended to allow user space to help `intel_pstate` to adjust the processor's internal P-state selection logic by focusing it on performance or on energy-efficiency, or somewhere between the two extremes:

**energy\_performance\_preference** Current value of the energy vs performance hint for the given policy (or the CPU represented by it).

The hint can be changed by writing to this attribute.

**energy\_performance\_available\_preferences** List of strings that can be written to the `energy_performance_preference` attribute.

They represent different energy vs performance hints and should be self-explanatory, except that `default` represents whatever hint value was set by the platform firmware.

Strings written to the `energy_performance_preference` attribute are internally translated to integer values written to the processor's Energy-Performance Preference (EPP) knob (if supported) or its Energy-Performance Bias (EPB) knob.

[Note that tasks may be migrated from one CPU to another by the scheduler's load-balancing algorithm and if different energy vs performance hints are set for those CPUs, that may lead to undesirable outcomes. To avoid such issues it is better to set the same energy vs performance hint for all CPUs or to pin every task potentially sensitive to them to a specific CPU.]

## intel\_pstate vs acpi-cpufreq

On the majority of systems supported by `intel_pstate`, the ACPI tables provided by the platform firmware contain `_PSS` objects returning information that can be used for CPU performance scaling (refer to the ACPI specification<sup>3</sup> for details on the `_PSS` objects and the format of the information returned by them).

The information returned by the ACPI `_PSS` objects is used by the `acpi-cpufreq` scaling driver. On systems supported by `intel_pstate` the `acpi-cpufreq` driver uses the same hardware CPU performance scaling interface, but the set of P-states it can use is limited by the `_PSS` output.

On those systems each `_PSS` object returns a list of P-states supported by the corresponding CPU which basically is a subset of the P-states range that can be used by `intel_pstate` on the same system, with one exception: the whole turbo range is represented by one item in it (the topmost one). By convention, the frequency returned by `_PSS` for that item is greater by 1 MHz than the frequency of the highest non-turbo P-state listed by it, but the corresponding P-state representation (following the hardware specification) returned for it matches the maximum supported turbo P-state (or is the special value 255 meaning essentially “go as high as you can get”).

The list of P-states returned by `_PSS` is reflected by the table of available frequencies supplied by `acpi-cpufreq` to the `CPUFreq` core and scaling governors and the minimum and maximum supported frequencies reported by it come from that list as well. In particular, given the special representation of the turbo range described above, this means that the maximum supported frequency reported by `acpi-cpufreq` is higher by 1 MHz than the frequency of the highest supported non-turbo P-state listed by `_PSS` which, of course, affects decisions made by the scaling governors, except for `powersave` and `performance`.

For example, if a given governor attempts to select a frequency proportional to estimated CPU load and maps the load of 100% to the maximum supported frequency (possibly multiplied by a constant), then it will tend to choose P-states below the turbo threshold if `acpi-cpufreq` is used as the scaling driver, because in that case the turbo range corresponds to a small fraction of the frequency band it can use (1 MHz vs 1 GHz or more). In consequence, it will only go to the turbo range for the highest loads and the other loads above 50% that might benefit from running at turbo frequencies will be given non-turbo P-states instead.

One more issue related to that may appear on systems supporting the Configurable TDP feature allowing the platform firmware to set the turbo threshold. Namely, if that is not coordinated with the lists of P-states returned by `_PSS` properly, there may be more than one item corresponding to a turbo P-state in those lists and there may be a problem with avoiding the turbo range (if desirable or necessary). Usually, to avoid using turbo P-states overall, `acpi-cpufreq` simply avoids using the topmost state listed by `_PSS`, but that is not sufficient when there are other turbo P-states in the list returned by it.

Apart from the above, `acpi-cpufreq` works like `intel_pstate` in the passive mode, except that the number of P-states it can set is limited to the ones listed by the ACPI `_PSS` objects.

---

<sup>3</sup> Advanced Configuration and Power Interface Specification, [https://uefi.org/sites/default/files/resources/ACPI\\_6\\_3\\_final\\_Jan30.pdf](https://uefi.org/sites/default/files/resources/ACPI_6_3_final_Jan30.pdf)

### Kernel Command Line Options for intel\_pstate

Several kernel command line options can be used to pass early-configuration-time parameters to `intel_pstate` in order to enforce specific behavior of it. All of them have to be prepended with the `intel_pstate=` prefix.

**disable** Do not register `intel_pstate` as the scaling driver even if the processor is supported by it.

**passive** Register `intel_pstate` in the passive mode to start with.

This option implies the `no_hwp` one described below.

**force** Register `intel_pstate` as the scaling driver instead of `acpi-cpufreq` even if the latter is preferred on the given system.

This may prevent some platform features (such as thermal controls and power capping) that rely on the availability of ACPI P-states information from functioning as expected, so it should be used with caution.

This option does not work with processors that are not supported by `intel_pstate` and on platforms where the `pcc-cpufreq` scaling driver is used instead of `acpi-cpufreq`.

**no\_hwp** Do not enable the hardware-managed P-states (HWP) feature even if it is supported by the processor.

**hwp\_only** Register `intel_pstate` as the scaling driver only if the hardware-managed P-states (HWP) feature is supported by the processor.

**support\_acpi\_ppc** Take ACPI `_PPC` performance limits into account.

If the preferred power management profile in the FADT (Fixed ACPI Description Table) is set to “Enterprise Server” or “Performance Server”, the ACPI `_PPC` limits are taken into account by default and this option has no effect.

**per\_cpu\_perf\_limits** Use per-logical-CPU P-State limits (see Coordination of P-state Limits for details).

### Diagnostics and Tuning

#### Trace Events

There are two static trace events that can be used for `intel_pstate` diagnostics. One of them is the `cpu_frequency` trace event generally used by CPUFreq, and the other one is the `pstate_sample` trace event specific to `intel_pstate`. Both of them are triggered by `intel_pstate` only if it works in the active mode.

The following sequence of shell commands can be used to enable them and see their output (if the kernel is generally configured to support event tracing):

```
# cd /sys/kernel/debug/tracing/
# echo 1 > events/power/pstate_sample/enable
# echo 1 > events/power/cpu_frequency/enable
# cat trace
```

(continues on next page)

(continued from previous page)

```
gnome-terminal--4510 [001] ..s. 1177.680733: pstate_sample: core_
↳busy=107 scaled=94 from=26 to=26 mperf=1143818 aperf=1230607
↳tsc=29838618 freq=2474476
cat-5235 [002] ..s. 1177.681723: cpu_frequency: state=2900000 cpu_id=2
```

If `intel_pstate` works in the passive mode, the `cpu_frequency` trace event will be triggered either by the `schedutil` scaling governor (for the policies it is attached to), or by the `CPUFreq` core (for the policies with other scaling governors).

## ftrace

The `ftrace` interface can be used for low-level diagnostics of `intel_pstate`. For example, to check how often the function to set a P-state is called, the `ftrace` filter can be set to `intel_pstate_set_pstate()`:

```
# cd /sys/kernel/debug/tracing/
# cat available_filter_functions | grep -i pstate
intel_pstate_set_pstate
intel_pstate_cpu_init
...
# echo intel_pstate_set_pstate > set_ftrace_filter
# echo function > current_tracer
# cat trace | head -15
# tracer: function
#
# entries-in-buffer/entries-written: 80/80 #P:4
#
#          _-----=> irqs-off
#          /_-----=> need-resched
#          |/_---=> hardirq/softirq
#          ||/_--=> preempt-depth
#          |||/_   delay
#          TASK-PID  CPU#  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
#          |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
#          Xorg-3129 [000] ..s. 2537.644844: intel_pstate_set_pstate <-
↳intel_pstate_timer_func
gnome-terminal--4510 [002] ..s. 2537.649844: intel_pstate_set_pstate <-
↳intel_pstate_timer_func
gnome-shell-3409 [001] ..s. 2537.650850: intel_pstate_set_pstate <-
↳intel_pstate_timer_func
<idle>-0 [000] ..s. 2537.654843: intel_pstate_set_pstate <-
↳intel_pstate_timer_func
```

### References

#### 61.3.5 Legacy Documentation of CPU Performance Scaling Drivers

Included below are historic documents describing assorted CPU performance scaling drivers. They are reproduced verbatim, with the original white space formatting and indentation preserved, except for the added leading space character in every line of text.

#### AMD PowerNow! Drivers

PowerNow! and Cool'n'Quiet are AMD names for frequency management capabilities in AMD processors. As the hardware implementation changes in new generations of the processors, there is a different `cpu-freq` driver for each generation.

Note that the driver's will not load on the "wrong" hardware, so it is safe to try each driver in turn when in doubt as to which is the correct driver.

Note that the functionality to change frequency (and voltage) is not available in all processors. The drivers will refuse to load on processors without this capability. The capability is detected with the `cpuid` instruction.

The drivers use BIOS supplied tables to obtain frequency and voltage information appropriate for a particular platform. Frequency transitions will be unavailable if the BIOS does not supply these tables.

6th Generation: `powernow-k6`

7th Generation: `powernow-k7`: Athlon, Duron, Geode.

8th Generation: `powernow-k8`: Athlon, Athlon 64, Opteron, Sempron. Documentation on this functionality in 8th generation processors is available in the "BIOS and Kernel Developer's Guide", publication 26094, in chapter 9, available for download from [www.amd.com](http://www.amd.com).

BIOS supplied data, for `powernow-k7` and for `powernow-k8`, may be from either the PSB table or from ACPI objects. The ACPI support is only available if the kernel config sets `CONFIG_ACPI_PROCESSOR`. The `powernow-k8` driver will attempt to use ACPI if so configured, and fall back to PST if that fails.

The `powernow-k7` driver will try to use the PSB support first, and fall back to ACPI if the PSB support fails. A module parameter, `acpi_force`, is provided to force ACPI support to be used instead of PSB support.

## cpufreq-nforce2

The cpufreq-nforce2 driver changes the FSB on nVidia nForce2 platforms.

This works better than on other platforms, because the FSB of the CPU can be controlled independently from the PCI/AGP clock.

The module has two options:

```
fid:      multiplier * 10 (for example 8.5 = 85)
min_fsb:  minimum FSB
```

If not set, fid is calculated from the current CPU speed and the FSB. min\_fsb defaults to FSB at boot time - 50 MHz.

**IMPORTANT:** The available range is limited downwards!  
Also the minimum available FSB can differ, for systems booting with 200 MHz, 150 should always work.

## pcc-cpufreq

```
/*
 * pcc-cpufreq.txt - PCC interface documentation
 *
 * Copyright (C) 2009 Red Hat, Matthew Garrett <mjg@redhat.com>
 * Copyright (C) 2009 Hewlett-Packard Development Company, L.P.
 *   Nagananda Chumbalkar <nagananda.chumbalkar@hp.com>
 */
```

### Processor Clocking Control Driver

-----

#### Contents:

-----

1. Introduction
  - 1.1 PCC interface
    - 1.1.1 Get Average Frequency
    - 1.1.2 Set Desired Frequency
  - 1.2 Platforms affected
2. Driver and /sys details
  - 2.1 scaling\_available\_frequencies
  - 2.2 cpuinfo\_transition\_latency
  - 2.3 cpuinfo\_cur\_freq
  - 2.4 related\_cpus
3. Caveats

#### 1. Introduction:

-----

Processor Clocking Control (PCC) is an interface between the platform firmware and OSPM. It is a mechanism for coordinating processor performance (ie: frequency) between the platform firmware and the OS.

The PCC driver (pcc-cpufreq) allows OSPM to take advantage of the PCC interface.

(continues on next page)

(continued from previous page)

OS utilizes the PCC interface to inform platform firmware what frequency,  
→the  
OS wants for a logical processor. The platform firmware attempts to achieve  
the requested frequency. If the request for the target frequency could not,  
→be  
satisfied by platform firmware, then it usually means that power budget  
conditions are in place, and "power capping" is taking place.

### 1.1 PCC interface:

-----  
The complete PCC specification is available here:  
<https://acpica.org/sites/acpica/files/Processor-Clocking-Control-v1p0.pdf>

PCC relies on a shared memory region that provides a channel for,  
→communication  
between the OS and platform firmware. PCC also implements a "doorbell" that  
is used by the OS to inform the platform firmware that a command has been  
sent.

The ACPI PCCH() method is used to discover the location of the PCC shared  
memory region. The shared memory region header contains the "command" and  
"status" interface. PCCH() also contains details on how to access the,  
→platform  
doorbell.

The following commands are supported by the PCC interface:

- \* Get Average Frequency
- \* Set Desired Frequency

The ACPI PCCP() method is implemented for each logical processor and is  
used to discover the offsets for the input and output buffers in the shared  
memory region.

When PCC mode is enabled, the platform will not expose processor,  
→performance  
or throttle states (`_PSS`, `_TSS` and related ACPI objects) to OSPM.  
→Therefore,  
the native P-state driver (such as `acpi-cpufreq` for Intel, `powernow-k8` for  
AMD) will not load.

However, OSPM remains in control of policy. The governor (eg: "ondemand")  
computes the required performance for each processor based on server,  
→workload.

The PCC driver fills in the command interface, and the input buffer and  
communicates the request to the platform firmware. The platform firmware is  
responsible for delivering the requested performance.

Each PCC command is "global" in scope and can affect all the logical CPUs,  
→in  
the system. Therefore, PCC is capable of performing "group" updates. With,  
→PCC  
the OS is capable of getting/setting the frequency of all the logical CPUs,  
→in  
the system with a single call to the BIOS.

(continues on next page)

(continued from previous page)

### 1.1.1 Get Average Frequency:

-----  
 This command is used by the OSPM to query the running frequency of the processor since the last time this command was completed. The output buffer indicates the average unhalted frequency of the logical processor,   
 ↪ expressed as

a percentage of the nominal (ie: maximum) CPU frequency. The output buffer also signifies if the CPU frequency is limited by a power budget condition.

### 1.1.2 Set Desired Frequency:

-----  
 This command is used by the OSPM to communicate to the platform firmware,   
 ↪ the

desired frequency for a logical processor. The output buffer is currently ignored by OSPM. The next invocation of "Get Average Frequency" will inform OSPM if the desired frequency was achieved or not.

## 1.2 Platforms affected:

-----  
 The PCC driver will load on any system where the platform firmware:   
 \* supports the PCC interface, and the associated PCCH() and PCCP() methods   
 \* assumes responsibility for managing the hardware clocking controls in,   
 ↪ order   
 to deliver the requested processor performance

Currently, certain HP ProLiant platforms implement the PCC interface. On,   
 ↪ those   
 platforms PCC is the "default" choice.

However, it is possible to disable this interface via a BIOS setting. In such an instance, as is also the case on platforms where the PCC interface is not implemented, the PCC driver will fail to load silently.

## 2. Driver and /sys details:

-----  
 When the driver loads, it merely prints the lowest and the highest CPU frequencies supported by the platform firmware.

The PCC driver loads with a message such as:   
 pcc-cpufreq: (v1.00.00) driver loaded with frequency limits: 1600 MHz, 2933 MHz

This means that the OSPM can request the CPU to run at any frequency in between the limits (1600 MHz, and 2933 MHz) specified in the message.

Internally, there is no need for the driver to convert the "target",   
 ↪ frequency   
 to a corresponding P-state.

The VERSION number for the driver will be of the format v.xy.ab.   
 eg: 1.00.02

```

-----
|      |
|      -- this will increase with bug fixes/enhancements to the driver
|-- this is the version of the PCC specification the driver adheres to

```

(continues on next page)

The following is a brief discussion on some of the fields exported via the /sys filesystem and how their values are affected by the PCC driver:

#### 2.1 scaling\_available\_frequencies:

scaling\_available\_frequencies is not created in /sys. No intermediate frequencies need to be listed because the BIOS will try to achieve any frequency, within limits, requested by the governor. A frequency does not have to be strictly associated with a P-state.

#### 2.2 cpuinfo\_transition\_latency:

The cpuinfo\_transition\_latency field is 0. The PCC specification does not include a field to expose this value currently.

#### 2.3 cpuinfo\_cur\_freq:

A) Often cpuinfo\_cur\_freq will show a value different than what is declared in the scaling\_available\_frequencies or scaling\_cur\_freq, or scaling\_max\_freq.

This is due to "turbo boost" available on recent Intel processors. If certain conditions are met the BIOS can achieve a slightly higher speed than requested by OSPM. An example:

```
scaling_cur_freq      : 2933000
cpuinfo_cur_freq      : 3196000
```

B) There is a round-off error associated with the cpuinfo\_cur\_freq value. Since the driver obtains the current frequency as a "percentage" (%) of the nominal frequency from the BIOS, sometimes, the values displayed by scaling\_cur\_freq and cpuinfo\_cur\_freq may not match. An example:

```
scaling_cur_freq      : 1600000
cpuinfo_cur_freq      : 1583000
```

In this example, the nominal frequency is 2933 MHz. The driver obtains the current frequency, cpuinfo\_cur\_freq, as 54% of the nominal frequency:

54% of 2933 MHz = 1583 MHz

Nominal frequency is the maximum frequency of the processor, and it usually corresponds to the frequency of the P0 P-state.

#### 2.4 related\_cpus:

The related\_cpus field is identical to affected\_cpus.

```
affected_cpus  : 4
related_cpus   : 4
```

Currently, the PCC driver does not evaluate \_PSD. The platforms that support

(continued from previous page)

```
PCC do not implement SW_ALL. So OSPM doesn't need to perform any
↳coordination
to ensure that the same frequency is requested of all dependent CPUs.
```

### 3. Caveats:

```
-----
The "cpufreq_stats" module in its present form cannot be loaded and
expected to work with the PCC driver. Since the "cpufreq_stats" module
provides information wrt each P-state, it is not applicable to the PCC
↳driver.
```

## 61.3.6 Intel Performance and Energy Bias Hint

**Copyright** © 2019 Intel Corporation

**Author** Rafael J. Wysocki <[rafael.j.wysocki@intel.com](mailto:rafael.j.wysocki@intel.com)>

The Performance and Energy Bias Hint (EPB) allows software to specify its preference with respect to the power-performance tradeoffs present in the processor. Generally, the EPB is expected to be set by user space (directly via sysfs or with the help of the `x86_energy_perf_policy` tool), but there are two reasons for the kernel to update it.

First, there are systems where the platform firmware resets the EPB during system-wide transitions from sleep states back into the working state effectively causing the previous EPB updates by user space to be lost. Thus the kernel needs to save the current EPB values for all CPUs during system-wide transitions to sleep states and restore them on the way back to the working state. That can be achieved by saving EPB for secondary CPUs when they are taken offline during transitions into system sleep states and for the boot CPU in a syscore suspend operation, so that it can be restored for the boot CPU in a syscore resume operation and for the other CPUs when they are brought back online. However, CPUs that are already offline when a system-wide PM transition is started are not taken offline again, but their EPB values may still be reset by the platform firmware during the transition, so in fact it is necessary to save the EPB of any CPU taken offline and to restore it when the given CPU goes back online at all times.

Second, on many systems the initial EPB value coming from the platform firmware is 0 ( 'performance' ) and at least on some of them that is because the platform firmware does not initialize EPB at all with the assumption that the OS will do that anyway. That sometimes is problematic, as it may cause the system battery to drain too fast, for example, so it is better to adjust it on CPU bring-up and if the initial EPB value for a given CPU is 0, the kernel changes it to 6 ( 'normal' ).

### Intel Performance and Energy Bias Attribute in sysfs

The Intel Performance and Energy Bias Hint (EPB) value for a given (logical) CPU can be checked or updated through a `sysfs` attribute (file) under `/sys/devices/system/cpu/cpu<N>/power/`, where the CPU number `<N>` is allocated at the system initialization time:

**energy\_perf\_bias** Shows the current EPB value for the CPU in a sliding scale 0 - 15, where a value of 0 corresponds to a hint preference for highest performance and a value of 15 corresponds to the maximum energy savings.

In order to update the EPB value for the CPU, this attribute can be written to, either with a number in the 0 - 15 sliding scale above, or with one of the strings: “performance”, “balance-performance”, “normal”, “balance-power”, “power” that represent values reflected by their meaning.

This attribute is present for all online CPUs supporting the EPB feature.

Note that while the EPB interface to the processor is defined at the logical CPU level, the physical register backing it may be shared by multiple CPUs (for example, SMT siblings or cores in one package). For this reason, updating the EPB value for one CPU may cause the EPB values for other CPUs to change.

### 61.3.7 Intel(R) Speed Select Technology User Guide

The Intel(R) Speed Select Technology (Intel(R) SST) provides a powerful new collection of features that give more granular control over CPU performance. With Intel(R) SST, one server can be configured for power and performance for a variety of diverse workload requirements.

Refer to the links below for an overview of the technology:

- <https://www.intel.com/content/www/us/en/architecture-and-technology/speed-select-technology-article.html>
- <https://builders.intel.com/docs/networkbuilders/intel-speed-select-technology-base-frequency.pdf>

These capabilities are further enhanced in some of the newer generations of server platforms where these features can be enumerated and controlled dynamically without pre-configuring via BIOS setup options. This dynamic configuration is done via mailbox commands to the hardware. One way to enumerate and configure these features is by using the Intel Speed Select utility.

This document explains how to use the Intel Speed Select tool to enumerate and control Intel(R) SST features. This document gives example commands and explains how these commands change the power and performance profile of the system under test. Using this tool as an example, customers can replicate the messaging implemented in the tool in their production software.

## intel-speed-select configuration tool

Most Linux distribution packages may include the “intel-speed-select” tool. If not, it can be built by downloading the Linux kernel tree from kernel.org. Once downloaded, the tool can be built without building the full kernel.

From the kernel tree, run the following commands:

```
# cd tools/power/x86/intel-speed-select/  
# make  
# make install
```

## Getting Help

To get help with the tool, execute the command below:

```
# intel-speed-select --help
```

The top-level help describes arguments and features. Notice that there is a multi-level help structure in the tool. For example, to get help for the feature “perf-profile” :

```
# intel-speed-select perf-profile --help
```

To get help on a command, another level of help is provided. For example for the command info “info” :

```
# intel-speed-select perf-profile info --help
```

## Summary of platform capability

To check the current platform and driver capabilities, execute:

```
#intel-speed-select --info
```

For example on a test system:

```
# intel-speed-select --info  
Intel(R) Speed Select Technology  
Executing on CPU model: X  
Platform: API version : 1  
Platform: Driver version : 1  
Platform: mbox supported : 1  
Platform: mmio supported : 1  
Intel(R) SST-PP (feature perf-profile) is supported  
TDP level change control is unlocked, max level: 4  
Intel(R) SST-TF (feature turbo-freq) is supported  
Intel(R) SST-BF (feature base-freq) is not supported  
Intel(R) SST-CP (feature core-power) is supported
```

### Intel(R) Speed Select Technology - Performance Profile (Intel(R) SST-PP)

This feature allows configuration of a server dynamically based on workload performance requirements. This helps users during deployment as they do not have to choose a specific server configuration statically. This Intel(R) Speed Select Technology - Performance Profile (Intel(R) SST-PP) feature introduces a mechanism that allows multiple optimized performance profiles per system. Each profile defines a set of CPUs that need to be online and rest offline to sustain a guaranteed base frequency. Once the user issues a command to use a specific performance profile and meet CPU online/offline requirement, the user can expect a change in the base frequency dynamically. This feature is called “perf-profile” when using the Intel Speed Select tool.

#### Number or performance levels

There can be multiple performance profiles on a system. To get the number of profiles, execute the command below:

```
# intel-speed-select perf-profile get-config-levels
Intel(R) Speed Select Technology
Executing on CPU model: X
package-0
  die-0
    cpu-0
      get-config-levels:4
package-1
  die-0
    cpu-14
      get-config-levels:4
```

On this system under test, there are 4 performance profiles in addition to the base performance profile (which is performance level 0).

#### Lock/Unlock status

Even if there are multiple performance profiles, it is possible that they are locked. If they are locked, users cannot issue a command to change the performance state. It is possible that there is a BIOS setup to unlock or check with your system vendor.

To check if the system is locked, execute the following command:

```
# intel-speed-select perf-profile get-lock-status
Intel(R) Speed Select Technology
Executing on CPU model: X
package-0
  die-0
    cpu-0
      get-lock-status:0
package-1
  die-0
```

(continues on next page)

(continued from previous page)

```
cpu-14
  get-lock-status:0
```

In this case, lock status is 0, which means that the system is unlocked.

## Properties of a performance level

To get properties of a specific performance level (For example for the level 0, below), execute the command below:

```
# intel-speed-select perf-profile info -l 0
Intel(R) Speed Select Technology
Executing on CPU model: X
package-0
  die-0
    cpu-0
      perf-profile-level-0
        cpu-count:28
        enable-cpu-mask:000003ff,f0003fff
        enable-cpu-list:0,1,2,3,4,5,6,7,8,9,10,11,12,13,28,29,30,31,32,33,
↪34,35,36,37,38,39,40,41
        thermal-design-power-ratio:26
        base-frequency(MHz):2600
        speed-select-turbo-freq:disabled
        speed-select-base-freq:disabled
        ...
        ...
```

Here -l option is used to specify a performance level.

If the option -l is omitted, then this command will print information about all the performance levels. The above command is printing properties of the performance level 0.

For this performance profile, the list of CPUs displayed by the “enable-cpu-mask/enable-cpu-list” at the max can be “online.” When that condition is met, then base frequency of 2600 MHz can be maintained. To understand more, execute “intel-speed-select perf-profile info” for performance level 4:

```
# intel-speed-select perf-profile info -l 4
Intel(R) Speed Select Technology
Executing on CPU model: X
package-0
  die-0
    cpu-0
      perf-profile-level-4
        cpu-count:28
        enable-cpu-mask:000000fa,f0000faf
        enable-cpu-list:0,1,2,3,5,7,8,9,10,11,28,29,30,31,33,35,36,37,38,39
        thermal-design-power-ratio:28
        base-frequency(MHz):2800
        speed-select-turbo-freq:disabled
        speed-select-base-freq:unsupported
```

(continues on next page)

(continued from previous page)

```
...  
...
```

There are fewer CPUs in the “enable-cpu-mask/enable-cpu-list” . Consequently, if the user only keeps these CPUs online and the rest “offline,” then the base frequency is increased to 2.8 GHz compared to 2.6 GHz at performance level 0.

### Get current performance level

To get the current performance level, execute:

```
# intel-speed-select perf-profile get-config-current-level  
Intel(R) Speed Select Technology  
Executing on CPU model: X  
package-0  
  die-0  
    cpu-0  
      get-config-current_level:0
```

First verify that the base\_frequency displayed by the cpufreq sysfs is correct:

```
# cat /sys/devices/system/cpu/cpu0/cpufreq/base_frequency  
2600000
```

This matches the base-frequency (MHz) field value displayed from the “perf-profile info” command for performance level 0(cpufreq frequency is in KHz).

To check if the average frequency is equal to the base frequency for a 100% busy workload, disable turbo:

```
# echo 1 > /sys/devices/system/cpu/intel_pstate/no_turbo
```

Then runs a busy workload on all CPUs, for example:

```
#stress -c 64
```

To verify the base frequency, run turbostat:

```
#turbostat -c 0-13 --show Package,Core,CPU,Bzy_MHz -i 1
```

Package	Core	CPU	Bzy_MHz
-	-	-	2600
0	0	0	2600
0	1	1	2600
0	2	2	2600
0	3	3	2600
0	4	4	2600
.	.	.	.

## Changing performance level

To the change the performance level to 4, execute:

```
# intel-speed-select -d perf-profile set-config-level -l 4 -o
Intel(R) Speed Select Technology
Executing on CPU model: X
package-0
  die-0
    cpu-0
      perf-profile
        set_tdp_level:success
```

In the command above, “-o” is optional. If it is specified, then it will also offline CPUs which are not present in the `enable_cpu_mask` for this performance level.

Now if the `base_frequency` is checked:

```
#cat /sys/devices/system/cpu/cpu0/cpufreq/base_frequency
2800000
```

Which shows that the base frequency now increased from 2600 MHz at performance level 0 to 2800 MHz at performance level 4. As a result, any workload, which can use fewer CPUs, can see a boost of 200 MHz compared to performance level 0.

## Check presence of other Intel(R) SST features

Each of the performance profiles also specifies weather there is support of other two Intel(R) SST features (Intel(R) Speed Select Technology - Base Frequency (Intel(R) SST-BF) and Intel(R) Speed Select Technology - Turbo Frequency (Intel SST-TF)).

For example, from the output of “perf-profile info” above, for level 0 and level 4:

**For level 0::** speed-select-turbo-freq:disabled speed-select-base-freq:disabled

**For level 4::** speed-select-turbo-freq:disabled speed-select-base-freq:unsupported

Given these results, the “speed-select-base-freq” (Intel(R) SST-BF) in level 4 changed from “disabled” to “unsupported” compared to performance level 0.

This means that at performance level 4, the “speed-select-base-freq” feature is not supported. However, at performance level 0, this feature is “supported”, but currently “disabled”, meaning the user has not activated this feature. Whereas “speed-select-turbo-freq”(Intel(R) SST-TF) is supported at both performance levels, but currently not activated by the user.

The Intel(R) SST-BF and the Intel(R) SST-TF features are built on a foundation technology called Intel(R) Speed Select Technology - Core Power (Intel(R) SST-CP). The platform firmware enables this feature when Intel(R) SST-BF or Intel(R) SST-TF is supported on a platform.

### Intel(R) Speed Select Technology Core Power (Intel(R) SST-CP)

Intel(R) Speed Select Technology Core Power (Intel(R) SST-CP) is an interface that allows users to define per core priority. This defines a mechanism to distribute power among cores when there is a power constrained scenario. This defines a class of service (CLOS) configuration.

The user can configure up to 4 class of service configurations. Each CLOS group configuration allows definitions of parameters, which affects how the frequency can be limited and power is distributed. Each CPU core can be tied to a class of service and hence an associated priority. The granularity is at core level not at per CPU level.

#### Enable CLOS based prioritization

To use CLOS based prioritization feature, firmware must be informed to enable and use a priority type. There is a default per platform priority type, which can be changed with optional command line parameter.

To enable and check the options, execute:

```
# intel-speed-select core-power enable --help
Intel(R) Speed Select Technology
Executing on CPU model: X
Enable core-power for a package/die
  Clos Enable: Specify priority type with [--priority|-p]
                0: Proportional, 1: Ordered
```

There are two types of priority types:

- Ordered

Priority for ordered throttling is defined based on the index of the assigned CLOS group. Where CLOS0 gets highest priority (throttled last).

Priority order is: CLOS0 > CLOS1 > CLOS2 > CLOS3.

- Proportional

When proportional priority is used, there is an additional parameter called `frequency_weight`, which can be specified per CLOS group. The goal of proportional priority is to provide each core with the requested min., then distribute all remaining (excess/deficit) budgets in proportion to a defined weight. This proportional priority can be configured using “core-power config” command.

To enable with the platform default priority type, execute:

```
# intel-speed-select core-power enable
Intel(R) Speed Select Technology
Executing on CPU model: X
package-0
  die-0
    cpu-0
      core-power
        enable:success
```

(continues on next page)

(continued from previous page)

```
package-1
  die-0
    cpu-6
      core-power
        enable:success
```

The scope of this enable is per package or die scoped when a package contains multiple dies. To check if CLOS is enabled and get priority type, “core-power info” command can be used. For example to check the status of core-power feature on CPU 0, execute:

```
# intel-speed-select -c 0 core-power info
Intel(R) Speed Select Technology
Executing on CPU model: X
package-0
  die-0
    cpu-0
      core-power
        support-status:supported
        enable-status:enabled
        clos-enable-status:enabled
        priority-type:proportional
package-1
  die-0
    cpu-24
      core-power
        support-status:supported
        enable-status:enabled
        clos-enable-status:enabled
        priority-type:proportional
```

### Configuring CLOS groups

Each CLOS group has its own attributes including min, max, freq\_weight and desired. These parameters can be configured with “core-power config” command. Defaults will be used if user skips setting a parameter except clos id, which is mandatory. To check core-power config options, execute:

```
# intel-speed-select core-power config --help
Intel(R) Speed Select Technology
Executing on CPU model: X
Set core-power configuration for one of the four clos ids
  Specify targeted clos id with [--clos|-c]
  Specify clos Proportional Priority [--weight|-w]
  Specify clos min in MHz with [--min|-n]
  Specify clos max in MHz with [--max|-m]
```

For example:

```
# intel-speed-select core-power config -c 0
Intel(R) Speed Select Technology
Executing on CPU model: X
clos epp is not specified, default: 0
```

(continues on next page)

(continued from previous page)

```
clos frequency weight is not specified, default: 0
clos min is not specified, default: 0 MHz
clos max is not specified, default: 25500 MHz
clos desired is not specified, default: 0
package-0
  die-0
    cpu-0
      core-power
        config:success
package-1
  die-0
    cpu-6
      core-power
        config:success
```

The user has the option to change defaults. For example, the user can change the “min” and set the base frequency to always get guaranteed base frequency.

### Get the current CLOS configuration

To check the current configuration, “core-power get-config” can be used. For example, to get the configuration of CLOS 0:

```
# intel-speed-select core-power get-config -c 0
Intel(R) Speed Select Technology
Executing on CPU model: X
package-0
  die-0
    cpu-0
      core-power
        clos:0
        epp:0
        clos-proportional-priority:0
        clos-min:0 MHz
        clos-max:Max Turbo frequency
        clos-desired:0 MHz
package-1
  die-0
    cpu-24
      core-power
        clos:0
        epp:0
        clos-proportional-priority:0
        clos-min:0 MHz
        clos-max:Max Turbo frequency
        clos-desired:0 MHz
```

## Associating a CPU with a CLOS group

To associate a CPU to a CLOS group “core-power assoc” command can be used:

```
# intel-speed-select core-power assoc --help
Intel(R) Speed Select Technology
Executing on CPU model: X
Associate a clos id to a CPU
    Specify targeted clos id with [--clos|-c]
```

For example to associate CPU 10 to CLOS group 3, execute:

```
# intel-speed-select -c 10 core-power assoc -c 3
Intel(R) Speed Select Technology
Executing on CPU model: X
package-0
  die-0
    cpu-10
      core-power
        assoc:success
```

Once a CPU is associated, its sibling CPUs are also associated to a CLOS group. Once associated, avoid changing Linux “cpufreq” subsystem scaling frequency limits.

To check the existing association for a CPU, “core-power get-assoc” command can be used. For example, to get association of CPU 10, execute:

```
# intel-speed-select -c 10 core-power get-assoc
Intel(R) Speed Select Technology
Executing on CPU model: X
package-1
  die-0
    cpu-10
      get-assoc
        clos:3
```

This shows that CPU 10 is part of a CLOS group 3.

## Disable CLOS based prioritization

To disable, execute:

```
# intel-speed-select core-power disable
```

Some features like Intel(R) SST-TF can only be enabled when CLOS based prioritization is enabled. For this reason, disabling while Intel(R) SST-TF is enabled can cause Intel(R) SST-TF to fail. This will cause the “disable” command to display an error if Intel(R) SST-TF is already enabled. In turn, to disable, the Intel(R) SST-TF feature must be disabled first.

### Intel(R) Speed Select Technology - Base Frequency (Intel(R) SST-BF)

The Intel(R) Speed Select Technology - Base Frequency (Intel(R) SST-BF) feature lets the user control base frequency. If some critical workload threads demand constant high guaranteed performance, then this feature can be used to execute the thread at higher base frequency on specific sets of CPUs (high priority CPUs) at the cost of lower base frequency (low priority CPUs) on other CPUs. This feature does not require offline of the low priority CPUs.

The support of Intel(R) SST-BF depends on the Intel(R) Speed Select Technology - Performance Profile (Intel(R) SST-PP) performance level configuration. It is possible that only certain performance levels support Intel(R) SST-BF. It is also possible that only base performance level (level = 0) has support of Intel SST-BF. Consequently, first select the desired performance level to enable this feature.

In the system under test here, Intel(R) SST-BF is supported at the base performance level 0, but currently disabled. For example for the level 0:

```
# intel-speed-select -c 0 perf-profile info -l 0
Intel(R) Speed Select Technology
Executing on CPU model: X
package-0
  die-0
    cpu-0
      perf-profile-level-0
        ...
          speed-select-base-freq:disabled
            ...
```

Before enabling Intel(R) SST-BF and measuring its impact on a workload performance, execute some workload and measure performance and get a baseline performance to compare against.

Here the user wants more guaranteed performance. For this reason, it is likely that turbo is disabled. To disable turbo, execute:

```
#echo 1 > /sys/devices/system/cpu/intel_pstate/no_turbo
```

Based on the output of the “intel-speed-select perf-profile info -l 0” base frequency of guaranteed frequency 2600 MHz.

### Measure baseline performance for comparison

To compare, pick a multi-threaded workload where each thread can be scheduled on separate CPUs. “Hackbench pipe” test is a good example on how to improve performance using Intel(R) SST-BF.

Below, the workload is measuring average scheduler wakeup latency, so a lower number means better performance:

```
# taskset -c 3,4 perf bench -r 100 sched pipe
# Running 'sched/pipe' benchmark:
# Executed 1000000 pipe operations between two processes
```

(continues on next page)

(continued from previous page)

```
Total time: 6.102 [sec]
6.102445 usecs/op
163868 ops/sec
```

While running the above test, if we take turbostat output, it will show us that 2 of the CPUs are busy and reaching max. frequency (which would be the base frequency as the turbo is disabled). The turbostat output:

```
#turbostat -c 0-13 --show Package,Core,CPU,Bzy_MHz -i 1
Package      Core      CPU      Bzy_MHz
0            0         0        1000
0            1         1        1005
0            2         2        1000
0            3         3        2600
0            4         4        2600
0            5         5        1000
0            6         6        1000
0            7         7        1005
0            8         8        1005
0            9         9        1000
0           10        10        1000
0           11        11         995
0           12        12        1000
0           13        13        1000
```

From the above turbostat output, both CPU 3 and 4 are very busy and reaching full guaranteed frequency of 2600 MHz.

### Intel(R) SST-BF Capabilities

To get capabilities of Intel(R) SST-BF for the current performance level 0, execute:

```
# intel-speed-select base-freq info -l 0
Intel(R) Speed Select Technology
Executing on CPU model: X
package-0
  die-0
    cpu-0
      speed-select-base-freq
        high-priority-base-frequency(MHz):3000
        high-priority-cpu-mask:00000216,00002160
        high-priority-cpu-list:5,6,8,13,33,34,36,41
        low-priority-base-frequency(MHz):2400
        tjunction-temperature(C):125
        thermal-design-power(W):205
```

The above capabilities show that there are some CPUs on this system that can offer base frequency of 3000 MHz compared to the standard base frequency at this performance levels. Nevertheless, these CPUs are fixed, and they are presented via high-priority-cpu-list/high-priority-cpu-mask. But if this Intel(R) SST-BF feature is selected, the low priorities CPUs (which are not in high-priority-cpu-list) can only offer up to 2400 MHz. As a result, if this clipping of low priority CPUs is acceptable, then the user can enable Intel SST-BF feature particularly for the

above “sched pipe” workload since only two CPUs are used, they can be scheduled on high priority CPUs and can get boost of 400 MHz.

### Enable Intel(R) SST-BF

To enable Intel(R) SST-BF feature, execute:

```
# intel-speed-select base-freq enable -a
Intel(R) Speed Select Technology
Executing on CPU model: X
package-0
  die-0
    cpu-0
      base-freq
        enable:success
package-1
  die-0
    cpu-14
      base-freq
        enable:success
```

In this case, -a option is optional. This not only enables Intel(R) SST-BF, but it also adjusts the priority of cores using Intel(R) Speed Select Technology Core Power (Intel(R) SST-CP) features. This option sets the minimum performance of each Intel(R) Speed Select Technology - Performance Profile (Intel(R) SST-PP) class to maximum performance so that the hardware will give maximum performance possible for each CPU.

If -a option is not used, then the following steps are required before enabling Intel(R) SST-BF:

- Discover Intel(R) SST-BF and note low and high priority base frequency
- Note the high priority CPU list
- Enable CLOS using core-power feature set
- Configure CLOS parameters. Use CLOS.min to set to minimum performance
- Subscribe desired CPUs to CLOS groups

With this configuration, if the same workload is executed by pinning the workload to high priority CPUs (CPU 5 and 6 in this case):

```
#taskset -c 5,6 perf bench -r 100 sched pipe
# Running 'sched/pipe' benchmark:
# Executed 1000000 pipe operations between two processes
  Total time: 5.627 [sec]
    5.627922 usecs/op
    177685 ops/sec
```

This way, by enabling Intel(R) SST-BF, the performance of this benchmark is improved (latency reduced) by 7.79%. From the turbostat output, it can be observed that the high priority CPUs reached 3000 MHz compared to 2600 MHz. The turbostat output:

```
#turbostat -c 0-13 --show Package,Core,CPU,Bzy_MHz -i 1
Package      Core      CPU      Bzy_MHz
0            0         0        2151
0            1         1        2166
0            2         2        2175
0            3         3        2175
0            4         4        2175
0            5         5        3000
0            6         6        3000
0            7         7        2180
0            8         8        2662
0            9         9        2176
0           10        10       2175
0           11        11       2176
0           12        12       2176
0           13        13       2661
```

### Disable Intel(R) SST-BF

To disable the Intel(R) SST-BF feature, execute:

```
# intel-speed-select base-freq disable -a
```

### Intel(R) Speed Select Technology - Turbo Frequency (Intel(R) SST-TF)

This feature enables the ability to set different “All core turbo ratio limits” to cores based on the priority. By using this feature, some cores can be configured to get higher turbo frequency by designating them as high priority at the cost of lower or no turbo frequency on the low priority cores.

For this reason, this feature is only useful when system is busy utilizing all CPUs, but the user wants some configurable option to get high performance on some CPUs.

The support of Intel(R) Speed Select Technology - Turbo Frequency (Intel(R) SST-TF) depends on the Intel(R) Speed Select Technology - Performance Profile (Intel SST-PP) performance level configuration. It is possible that only a certain performance level supports Intel(R) SST-TF. It is also possible that only the base performance level (level = 0) has the support of Intel(R) SST-TF. Hence, first select the desired performance level to enable this feature.

In the system under test here, Intel(R) SST-TF is supported at the base performance level 0, but currently disabled:

```
# intel-speed-select -c 0 perf-profile info -l 0
Intel(R) Speed Select Technology
package-0
die-0
  cpu-0
    perf-profile-level-0
    ...
    ...
```

(continues on next page)

(continued from previous page)

```
speed-select-turbo-freq:disabled
...
...
```

To check if performance can be improved using Intel(R) SST-TF feature, get the turbo frequency properties with Intel(R) SST-TF enabled and compare to the base turbo capability of this system.

### Get Base turbo capability

To get the base turbo capability of performance level 0, execute:

```
# intel-speed-select perf-profile info -l 0
Intel(R) Speed Select Technology
Executing on CPU model: X
package-0
die-0
  cpu-0
    perf-profile-level-0
      ...
      ...
      turbo-ratio-limits-sse
        bucket-0
          core-count:2
          max-turbo-frequency(MHz):3200
        bucket-1
          core-count:4
          max-turbo-frequency(MHz):3100
        bucket-2
          core-count:6
          max-turbo-frequency(MHz):3100
        bucket-3
          core-count:8
          max-turbo-frequency(MHz):3100
        bucket-4
          core-count:10
          max-turbo-frequency(MHz):3100
        bucket-5
          core-count:12
          max-turbo-frequency(MHz):3100
        bucket-6
          core-count:14
          max-turbo-frequency(MHz):3100
        bucket-7
          core-count:16
          max-turbo-frequency(MHz):3100
```

Based on the data above, when all the CPUS are busy, the max. frequency of 3100 MHz can be achieved. If there is some busy workload on cpu 0 - 11 (e.g. stress) and on CPU 12 and 13, execute “hackbench pipe” workload:

```
# taskset -c 12,13 perf bench -r 100 sched pipe
# Running 'sched/pipe' benchmark:
```

(continues on next page)

(continued from previous page)

```
# Executed 1000000 pipe operations between two processes
  Total time: 5.705 [sec]
    5.705488 usecs/op
    175269 ops/sec
```

The turbostat output:

```
#turbostat -c 0-13 --show Package,Core,CPU,Bzy_MHz -i 1
Package      Core      CPU      Bzy_MHz
0            0         0        3000
0            1         1        3000
0            2         2        3000
0            3         3        3000
0            4         4        3000
0            5         5        3100
0            6         6        3100
0            7         7        3000
0            8         8        3100
0            9         9        3000
0           10        10        3000
0           11        11        3000
0           12        12        3100
0           13        13        3100
```

Based on turbostat output, the performance is limited by frequency cap of 3100 MHz. To check if the hackbench performance can be improved for CPU 12 and CPU 13, first check the capability of the Intel(R) SST-TF feature for this performance level.

### Get Intel(R) SST-TF Capability

To get the capability, the “turbo-freq info” command can be used:

```
# intel-speed-select turbo-freq info -l 0
Intel(R) Speed Select Technology
Executing on CPU model: X
package-0
  die-0
    cpu-0
      speed-select-turbo-freq
        bucket-0
          high-priority-cores-count:2
          high-priority-max-frequency(MHz):3200
          high-priority-max-avx2-frequency(MHz):3200
          high-priority-max-avx512-frequency(MHz):3100
        bucket-1
          high-priority-cores-count:4
          high-priority-max-frequency(MHz):3100
          high-priority-max-avx2-frequency(MHz):3000
          high-priority-max-avx512-frequency(MHz):2900
        bucket-2
          high-priority-cores-count:6
          high-priority-max-frequency(MHz):3100
```

(continues on next page)

(continued from previous page)

```
high-priority-max-avx2-frequency(MHz):3000
high-priority-max-avx512-frequency(MHz):2900
speed-select-turbo-freq-clip-frequencies
low-priority-max-frequency(MHz):2600
low-priority-max-avx2-frequency(MHz):2400
low-priority-max-avx512-frequency(MHz):2100
```

Based on the output above, there is an Intel(R) SST-TF bucket for which there are two high priority cores. If only two high priority cores are set, then max. turbo frequency on those cores can be increased to 3200 MHz. This is 100 MHz more than the base turbo capability for all cores.

In turn, for the hackbench workload, two CPUs can be set as high priority and rest as low priority. One side effect is that once enabled, the low priority cores will be clipped to a lower frequency of 2600 MHz.

### Enable Intel(R) SST-TF

To enable Intel(R) SST-TF, execute:

```
# intel-speed-select -c 12,13 turbo-freq enable -a
Intel(R) Speed Select Technology
Executing on CPU model: X
package-0
  die-0
    cpu-12
      turbo-freq
        enable:success
package-0
  die-0
    cpu-13
      turbo-freq
        enable:success
package--1
  die-0
    cpu-63
      turbo-freq --auto
        enable:success
```

In this case, the option “-a” is optional. If set, it enables Intel(R) SST-TF feature and also sets the CPUs to high and low priority using Intel Speed Select Technology Core Power (Intel(R) SST-CP) features. The CPU numbers passed with “-c” arguments are marked as high priority, including its siblings.

If -a option is not used, then the following steps are required before enabling Intel(R) SST-TF:

- Discover Intel(R) SST-TF and note buckets of high priority cores and maximum frequency
- Enable CLOS using core-power feature set - Configure CLOS parameters
- Subscribe desired CPUs to CLOS groups making sure that high priority cores are set to the maximum frequency

If the same hackbench workload is executed, schedule hackbench threads on high priority CPUs:

```
#taskset -c 12,13 perf bench -r 100 sched pipe
# Running 'sched/pipe' benchmark:
# Executed 1000000 pipe operations between two processes
  Total time: 5.510 [sec]
    5.510165 usecs/op
    180826 ops/sec
```

This improved performance by around 3.3% improvement on a busy system. Here the turbostat output will show that the CPU 12 and CPU 13 are getting 100 MHz boost. The turbostat output:

```
#turbostat -c 0-13 --show Package,Core,CPU,Bzy_MHz -i 1
Package      Core    CPU    Bzy_MHz
...
0            12     12     3200
0            13     13     3200
```



## LINUX PLUG AND PLAY DOCUMENTATION

**Author** Adam Belay <ambx1@neo.rr.com>

**Last updated** Oct. 16, 2002

### 62.1 Overview

Plug and Play provides a means of detecting and setting resources for legacy or otherwise unconfigurable devices. The Linux Plug and Play Layer provides these services to compatible drivers.

### 62.2 The User Interface

The Linux Plug and Play user interface provides a means to activate PnP devices for legacy and user level drivers that do not support Linux Plug and Play. The user interface is integrated into sysfs.

In addition to the standard sysfs file the following are created in each device' s directory: - id - displays a list of support EISA IDs - options - displays possible resource configurations - resources - displays currently allocated resources and allows resource changes

#### 62.2.1 activating a device

```
# echo "auto" > resources
```

this will invoke the automatic resource config system to activate the device

### 62.2.2 manually activating a device

```
# echo "manual <depnum> <mode>" > resources
```

<depnum> - the configuration number

<mode> - static or dynamic  
static = for next boot  
dynamic = now

### 62.2.3 disabling a device

```
# echo "disable" > resources
```

EXAMPLE:

Suppose you need to activate the floppy disk controller.

1. change to the proper directory, in my case it is /driver/bus/pnp/devices/00:0f:

```
# cd /driver/bus/pnp/devices/00:0f
# cat name
PC standard floppy disk controller
```

2. check if the device is already active:

```
# cat resources
DISABLED
```

- Notice the string "DISABLED". This means the device is not active.

3. check the device's possible configurations (optional):

```
# cat options
Dependent: 01 - Priority acceptable
  port 0x3f0-0x3f0, align 0x7, size 0x6, 16-bit address decoding
  port 0x3f7-0x3f7, align 0x0, size 0x1, 16-bit address decoding
  irq 6
  dma 2 8-bit compatible
Dependent: 02 - Priority acceptable
  port 0x370-0x370, align 0x7, size 0x6, 16-bit address decoding
  port 0x377-0x377, align 0x0, size 0x1, 16-bit address decoding
  irq 6
  dma 2 8-bit compatible
```

4. now activate the device:

```
# echo "auto" > resources
```

5. finally check if the device is active:

```
# cat resources
io 0x3f0-0x3f5
io 0x3f7-0x3f7
irq 6
dma 2
```

also there are a series of kernel parameters:

```
pnp_reserve_irq=irq1[,irq2] ....
pnp_reserve_dma=dma1[,dma2] ....
pnp_reserve_io=io1,size1[,io2,size2] ....
pnp_reserve_mem=mem1,size1[,mem2,size2] ....
```

## 62.3 The Unified Plug and Play Layer

All Plug and Play drivers, protocols, and services meet at a central location called the Plug and Play Layer. This layer is responsible for the exchange of information between PnP drivers and PnP protocols. Thus it automatically forwards commands to the proper protocol. This makes writing PnP drivers significantly easier.

The following functions are available from the Plug and Play Layer:

**pnp\_get\_protocol** increments the number of uses by one

**pnp\_put\_protocol** deincrements the number of uses by one

**pnp\_register\_protocol** use this to register a new PnP protocol

**pnp\_unregister\_protocol** use this function to remove a PnP protocol from the Plug and Play Layer

**pnp\_register\_driver** adds a PnP driver to the Plug and Play Layer

this includes driver model integration returns zero for success or a negative error number for failure; count calls to the .add() method if you need to know how many devices bind to the driver

**pnp\_unregister\_driver** removes a PnP driver from the Plug and Play Layer

## 62.4 Plug and Play Protocols

This section contains information for PnP protocol developers.

The following Protocols are currently available in the computing world:

- **PNPBIOS:** used for system devices such as serial and parallel ports.
- **ISAPNP:** provides PnP support for the ISA bus
- **ACPI:** among its many uses, ACPI provides information about system level devices.

It is meant to replace the PNPBIOS. It is not currently supported by Linux Plug and Play but it is planned to be in the near future.

Requirements for a Linux PnP protocol: 1. the protocol must use EISA IDs 2. the protocol must inform the PnP Layer of a device' s current configuration

- the ability to set resources is optional but preferred.

The following are PnP protocol related functions:

**pnnp\_add\_device** use this function to add a PnP device to the PnP layer

only call this function when all wanted values are set in the pnp\_dev structure

**pnnp\_init\_device** call this to initialize the PnP structure

**pnnp\_remove\_device** call this to remove a device from the Plug and Play Layer. it will fail if the device is still in use. automatically will free mem used by the device and related structures

**pnnp\_add\_id** adds an EISA ID to the list of supported IDs for the specified device

For more information consult the source of a protocol such as /drivers/pnp/pnpbios/core.c.

## 62.5 Linux Plug and Play Drivers

This section contains information for Linux PnP driver developers.

### 62.5.1 The New Way

1. first make a list of supported EISA IDS

ex:

```
static const struct pnp_id pnp_dev_table[] = {
    /* Standard LPT Printer Port */
    {.id = "PNP0400", .driver_data = 0},
    /* ECP Printer Port */
    {.id = "PNP0401", .driver_data = 0},
    {.id = ""}
};
```

Please note that the character 'X' can be used as a wild card in the function portion (last four characters).

ex:

```
/* Unknown PnP modems */
{ "PNPCXXX", UNKNOWN_DEV },
```

Supported PnP card IDs can optionally be defined. ex:

```
static const struct pnp_id pnp_card_table[] = {
    { "ANYDEVS", 0 },
    { "", 0 }
};
```

2. Optionally define probe and remove functions. It may make sense not to define these functions if the driver already has a reliable method of detecting the resources, such as the parport\_pc driver.

ex:

```
static int
serial_pnp_probe(struct pnp_dev * dev, const struct pnp_id *card_id,
↳const
                struct pnp_id *dev_id)
{
. . .
```

ex:

```
static void serial_pnp_remove(struct pnp_dev * dev)
{
. . .
```

consult /drivers/serial/8250\_pnp.c for more information.

### 3. create a driver structure

ex:

```
static struct pnp_driver serial_pnp_driver = {
    .name           = "serial",
    .card_id_table  = pnp_card_table,
    .id_table       = pnp_dev_table,
    .probe         = serial_pnp_probe,
    .remove        = serial_pnp_remove,
};
```

- name and id\_table cannot be NULL.

### 4. register the driver

ex:

```
static int __init serial8250_pnp_init(void)
{
    return pnp_register_driver(&serial_pnp_driver);
}
```

## 62.5.2 The Old Way

A series of compatibility functions have been created to make it easy to convert ISAPNP drivers. They should serve as a temporary solution only.

They are as follows:

```
struct pnp_card *pnp_find_card(unsigned short vendor,
                              unsigned short device,
                              struct pnp_card *from)

struct pnp_dev *pnp_find_dev(struct pnp_card *card,
                             unsigned short vendor,
                             unsigned short function,
                             struct pnp_dev *from)
```



## **RAPIDIO SUBSYSTEM GUIDE**

**Author** Matt Porter

### **63.1 Introduction**

RapidIO is a high speed switched fabric interconnect with features aimed at the embedded market. RapidIO provides support for memory-mapped I/O as well as message-based transactions over the switched fabric network. RapidIO has a standardized discovery mechanism not unlike the PCI bus standard that allows simple detection of devices in a network.

This documentation is provided for developers intending to support RapidIO on new architectures, write new drivers, or to understand the subsystem internals.

### **63.2 Known Bugs and Limitations**

#### **63.2.1 Bugs**

None. ;)

#### **63.2.2 Limitations**

1. Access/management of RapidIO memory regions is not supported
2. Multiple host enumeration is not supported

### **63.3 RapidIO driver interface**

Drivers are provided a set of calls in order to interface with the subsystem to gather info on devices, request/map memory region resources, and manage mailboxes/doorbells.

### 63.3.1 Functions

int **rio\_local\_read\_config\_32**(struct rio\_mport \* port, u32 offset, u32  
\* data)  
Read 32 bits from local configuration space

#### Parameters

**struct rio\_mport \* port** Master port  
**u32 offset** Offset into local configuration space  
**u32 \* data** Pointer to read data into

#### Description

Reads 32 bits of data from the specified offset within the local device' s configuration space.

int **rio\_local\_write\_config\_32**(struct rio\_mport \* port, u32 offset,  
u32 data)  
Write 32 bits to local configuration space

#### Parameters

**struct rio\_mport \* port** Master port  
**u32 offset** Offset into local configuration space  
**u32 data** Data to be written

#### Description

Writes 32 bits of data to the specified offset within the local device' s configuration space.

int **rio\_local\_read\_config\_16**(struct rio\_mport \* port, u32 offset, u16  
\* data)  
Read 16 bits from local configuration space

#### Parameters

**struct rio\_mport \* port** Master port  
**u32 offset** Offset into local configuration space  
**u16 \* data** Pointer to read data into

#### Description

Reads 16 bits of data from the specified offset within the local device' s configuration space.

int **rio\_local\_write\_config\_16**(struct rio\_mport \* port, u32 offset,  
u16 data)  
Write 16 bits to local configuration space

#### Parameters

**struct rio\_mport \* port** Master port  
**u32 offset** Offset into local configuration space  
**u16 data** Data to be written

**Description**

Writes 16 bits of data to the specified offset within the local device' s configuration space.

```
int rio_local_read_config_8(struct rio_mport * port, u32 offset, u8 * data)
    Read 8 bits from local configuration space
```

**Parameters**

**struct rio\_mport \* port** Master port  
**u32 offset** Offset into local configuration space  
**u8 \* data** Pointer to read data into

**Description**

Reads 8 bits of data from the specified offset within the local device' s configuration space.

```
int rio_local_write_config_8(struct rio_mport * port, u32 offset, u8 data)
    Write 8 bits to local configuration space
```

**Parameters**

**struct rio\_mport \* port** Master port  
**u32 offset** Offset into local configuration space  
**u8 data** Data to be written

**Description**

Writes 8 bits of data to the specified offset within the local device' s configuration space.

```
int rio_read_config_32(struct rio_dev * rdev, u32 offset, u32 * data)
    Read 32 bits from configuration space
```

**Parameters**

**struct rio\_dev \* rdev** RIO device  
**u32 offset** Offset into device configuration space  
**u32 \* data** Pointer to read data into

**Description**

Reads 32 bits of data from the specified offset within the RIO device' s configuration space.

```
int rio_write_config_32(struct rio_dev * rdev, u32 offset, u32 data)
    Write 32 bits to configuration space
```

**Parameters**

**struct rio\_dev \* rdev** RIO device  
**u32 offset** Offset into device configuration space  
**u32 data** Data to be written

### Description

Writes 32 bits of data to the specified offset within the RIO device's configuration space.

```
int rio_read_config_16(struct rio_dev * rdev, u32 offset, u16 * data)
    Read 16 bits from configuration space
```

### Parameters

**struct rio\_dev \* rdev** RIO device  
**u32 offset** Offset into device configuration space  
**u16 \* data** Pointer to read data into

### Description

Reads 16 bits of data from the specified offset within the RIO device's configuration space.

```
int rio_write_config_16(struct rio_dev * rdev, u32 offset, u16 data)
    Write 16 bits to configuration space
```

### Parameters

**struct rio\_dev \* rdev** RIO device  
**u32 offset** Offset into device configuration space  
**u16 data** Data to be written

### Description

Writes 16 bits of data to the specified offset within the RIO device's configuration space.

```
int rio_read_config_8(struct rio_dev * rdev, u32 offset, u8 * data)
    Read 8 bits from configuration space
```

### Parameters

**struct rio\_dev \* rdev** RIO device  
**u32 offset** Offset into device configuration space  
**u8 \* data** Pointer to read data into

### Description

Reads 8 bits of data from the specified offset within the RIO device's configuration space.

```
int rio_write_config_8(struct rio_dev * rdev, u32 offset, u8 data)
    Write 8 bits to configuration space
```

### Parameters

**struct rio\_dev \* rdev** RIO device  
**u32 offset** Offset into device configuration space  
**u8 data** Data to be written

**Description**

Writes 8 bits of data to the specified offset within the RIO device' s configuration space.

```
int rio_send_doorbell(struct rio_dev * rdev, u16 data)
    Send a doorbell message to a device
```

**Parameters**

**struct rio\_dev \* rdev** RIO device

**u16 data** Doorbell message data

**Description**

Send a doorbell message to a RIO device. The doorbell message has a 16-bit info field provided by the **data** argument.

```
void rio_init_mbox_res(struct resource * res, int start, int end)
    Initialize a RIO mailbox resource
```

**Parameters**

**struct resource \* res** resource struct

**int start** start of mailbox range

**int end** end of mailbox range

**Description**

This function is used to initialize the fields of a resource for use as a mailbox resource. It initializes a range of mailboxes using the start and end arguments.

```
void rio_init_dbell_res(struct resource * res, u16 start, u16 end)
    Initialize a RIO doorbell resource
```

**Parameters**

**struct resource \* res** resource struct

**u16 start** start of doorbell range

**u16 end** end of doorbell range

**Description**

This function is used to initialize the fields of a resource for use as a doorbell resource. It initializes a range of doorbell messages using the start and end arguments.

```
RIO_DEVICE(dev, ven)
    macro used to describe a specific RIO device
```

**Parameters**

**dev** the 16 bit RIO device ID

**ven** the 16 bit RIO vendor ID

**Description**

This macro is used to create a struct `rio_device_id` that matches a specific device. The assembly vendor and assembly device fields will be set to `RIO_ANY_ID`.

`int rio_add_outb_message(struct rio_mport * mport, struct rio_dev * rdev,  
int mbox, void * buffer, size_t len)`  
Add RIO message to an outbound mailbox queue

### Parameters

`struct rio_mport * mport` RIO master port containing the outbound queue

`struct rio_dev * rdev` RIO device the message is be sent to

`int mbox` The outbound mailbox queue

`void * buffer` Pointer to the message buffer

`size_t len` Length of the message buffer

### Description

Adds a RIO message buffer to an outbound mailbox queue for transmission. Returns 0 on success.

`int rio_add_inb_buffer(struct rio_mport * mport, int mbox, void * buffer)`  
Add buffer to an inbound mailbox queue

### Parameters

`struct rio_mport * mport` Master port containing the inbound mailbox

`int mbox` The inbound mailbox number

`void * buffer` Pointer to the message buffer

### Description

Adds a buffer to an inbound mailbox queue for reception. Returns 0 on success.

`void * rio_get_inb_message(struct rio_mport * mport, int mbox)`  
Get A RIO message from an inbound mailbox queue

### Parameters

`struct rio_mport * mport` Master port containing the inbound mailbox

`int mbox` The inbound mailbox number

### Description

Get a RIO message from an inbound mailbox queue. Returns 0 on success.

`const char * rio_name(struct rio_dev * rdev)`  
Get the unique RIO device identifier

### Parameters

`struct rio_dev * rdev` RIO device

### Description

Get the unique RIO device identifier. Returns the device identifier string.

`void * rio_get_drvdata(struct rio_dev * rdev)`  
Get RIO driver specific data

### Parameters

`struct rio_dev * rdev` RIO device

**Description**

Get RIO driver specific data. Returns a pointer to the driver specific data.

```
void rio_set_drvdata(struct rio_dev * rdev, void * data)
    Set RIO driver specific data
```

**Parameters**

```
struct rio_dev * rdev RIO device
void * data Pointer to driver specific data
```

**Description**

Set RIO driver specific data. device struct driver data pointer is set to the **data** argument.

```
struct rio_dev * rio_dev_get(struct rio_dev * rdev)
    Increments the reference count of the RIO device structure
```

**Parameters**

```
struct rio_dev * rdev RIO device being referenced
```

**Description**

Each live reference to a device should be refcounted.

Drivers for RIO devices should normally record such references in their probe() methods, when they bind to a device, and release them by calling rio\_dev\_put(), in their disconnect() methods.

```
void rio_dev_put(struct rio_dev * rdev)
    Release a use of the RIO device structure
```

**Parameters**

```
struct rio_dev * rdev RIO device being disconnected
```

**Description**

Must be called when a user of a device is finished with it. When the last user of the device calls this function, the memory of the device is freed.

```
int rio_register_driver(struct rio_driver * rdrv)
    register a new RIO driver
```

**Parameters**

```
struct rio_driver * rdrv the RIO driver structure to register
```

Adds a struct rio\_driver to the list of registered drivers. Returns a negative value on error, otherwise 0. If no error occurred, the driver remains registered even if no device was claimed during registration.

```
void rio_unregister_driver(struct rio_driver * rdrv)
    unregister a RIO driver
```

**Parameters**

```
struct rio_driver * rdrv the RIO driver structure to unregister
```

Deletes the struct `rio_driver` from the list of registered RIO drivers, gives it a chance to clean up by calling its `remove()` function for each device it was responsible for, and marks those devices as driverless.

`u16 rio_local_get_device_id(struct rio_mport * port)`  
Get the base/extended device id for a port

### Parameters

**struct rio\_mport \* port** RIO master port from which to get the deviceid

### Description

Reads the base/extended device id from the local device implementing the master port. Returns the 8/16-bit device id.

`int rio_query_mport(struct rio_mport * port, struct rio_mport_attr * mport_attr)`  
Query mport device attributes

### Parameters

**struct rio\_mport \* port** mport device to query

**struct rio\_mport\_attr \* mport\_attr** mport attributes data structure

### Description

Returns attributes of specified mport through the pointer to attributes data structure.

`struct rio_net * rio_alloc_net(struct rio_mport * mport)`  
Allocate and initialize a new RIO network data structure

### Parameters

**struct rio\_mport \* mport** Master port associated with the RIO network

### Description

Allocates a RIO network structure, initializes per-network list heads, and adds the associated master port to the network list of associated master ports. Returns a RIO network pointer on success or NULL on failure.

`void rio_local_set_device_id(struct rio_mport * port, u16 did)`  
Set the base/extended device id for a port

### Parameters

**struct rio\_mport \* port** RIO master port

**u16 did** Device ID value to be written

### Description

Writes the base/extended device id from a device.

`int rio_add_device(struct rio_dev * rdev)`  
Adds a RIO device to the device model

### Parameters

**struct rio\_dev \* rdev** RIO device

**Description**

Adds the RIO device to the global device list and adds the RIO device to the RIO device list. Creates the generic sysfs nodes for an RIO device.

```
int rio_request_inb_mbox(struct rio_mport * mport, void * dev_id,
                        int mbox, int entries, void (*minb)(struct
                        rio_mport * mport, void *dev_id, int mbox,
                        int slot))
    request inbound mailbox service
```

**Parameters**

**struct rio\_mport \* mport** RIO master port from which to allocate the mailbox resource

**void \* dev\_id** Device specific pointer to pass on event

**int mbox** Mailbox number to claim

**int entries** Number of entries in inbound mailbox queue

**void (\*) (struct rio\_mport \* mport, void \*dev\_id, int mbox, int slot) minb**  
Callback to execute when inbound message is received

**Description**

Requests ownership of an inbound mailbox resource and binds a callback function to the resource. Returns 0 on success.

```
int rio_release_inb_mbox(struct rio_mport * mport, int mbox)
    release inbound mailbox message service
```

**Parameters**

**struct rio\_mport \* mport** RIO master port from which to release the mailbox resource

**int mbox** Mailbox number to release

**Description**

Releases ownership of an inbound mailbox resource. Returns 0 if the request has been satisfied.

```
int rio_request_outb_mbox(struct rio_mport * mport, void * dev_id,
                          int mbox, int entries, void (*moutb)(struct
                          rio_mport * mport, void *dev_id, int mbox, int
                          slot))
    request outbound mailbox service
```

**Parameters**

**struct rio\_mport \* mport** RIO master port from which to allocate the mailbox resource

**void \* dev\_id** Device specific pointer to pass on event

**int mbox** Mailbox number to claim

**int entries** Number of entries in outbound mailbox queue

**void (\*) (struct rio\_mport \* mport, void \*dev\_id, int mbox, int slot) moutb**  
Callback to execute when outbound message is sent

### Description

Requests ownership of an outbound mailbox resource and binds a callback function to the resource. Returns 0 on success.

int **rio\_release\_outb\_mbox**(struct rio\_mport \* mport, int mbox)  
release outbound mailbox message service

### Parameters

**struct rio\_mport \* mport** RIO master port from which to release the mailbox resource

**int mbox** Mailbox number to release

### Description

Releases ownership of an inbound mailbox resource. Returns 0 if the request has been satisfied.

int **rio\_request\_inb\_dbell**(struct rio\_mport \* mport, void \* dev\_id, u16 start, u16 end, void (\*dinb)(struct rio\_mport \* mport, void \*dev\_id, u16 src, u16 dst, u16 info))  
request inbound doorbell message service

### Parameters

**struct rio\_mport \* mport** RIO master port from which to allocate the doorbell resource

**void \* dev\_id** Device specific pointer to pass on event

**u16 start** Doorbell info range start

**u16 end** Doorbell info range end

**void (\*) (struct rio\_mport \* mport, void \*dev\_id, u16 src, u16 dst, u16 info) di**  
Callback to execute when doorbell is received

### Description

Requests ownership of an inbound doorbell resource and binds a callback function to the resource. Returns 0 if the request has been satisfied.

int **rio\_release\_inb\_dbell**(struct rio\_mport \* mport, u16 start, u16 end)  
release inbound doorbell message service

### Parameters

**struct rio\_mport \* mport** RIO master port from which to release the doorbell resource

**u16 start** Doorbell info range start

**u16 end** Doorbell info range end

### Description

Releases ownership of an inbound doorbell resource and removes callback from the doorbell event list. Returns 0 if the request has been satisfied.

```
struct resource * rio_request_outb_dbell(struct rio_dev * rdev, u16 start,  
                                         u16 end)  
    request outbound doorbell message range
```

### Parameters

**struct rio\_dev \* rdev** RIO device from which to allocate the doorbell resource

**u16 start** Doorbell message range start

**u16 end** Doorbell message range end

### Description

Requests ownership of a doorbell message range. Returns a resource if the request has been satisfied or NULL on failure.

```
int rio_release_outb_dbell(struct rio_dev * rdev, struct resource * res)  
    release outbound doorbell message range
```

### Parameters

**struct rio\_dev \* rdev** RIO device from which to release the doorbell resource

**struct resource \* res** Doorbell resource to be freed

### Description

Releases ownership of a doorbell message range. Returns 0 if the request has been satisfied.

```
int rio_add_mport_pw_handler(struct rio_mport * mport, void * context, int  
                             (*pwcbk)(struct rio_mport *mport, void  
                             *context, union rio_pw_msg *msg, int step))  
    add port-write message handler into the list of mport specific pw handlers
```

### Parameters

**struct rio\_mport \* mport** RIO master port to bind the portwrite callback

**void \* context** Handler specific context to pass on event

**int (\*)(struct rio\_mport \*mport, void \*context, union rio\_pw\_msg \*msg, int step)**  
 Callback to execute when portwrite is received

### Description

Returns 0 if the request has been satisfied.

```
int rio_del_mport_pw_handler(struct rio_mport * mport, void * context, int  
                             (*pwcbk)(struct rio_mport *mport, void  
                             *context, union rio_pw_msg *msg, int step))  
    remove port-write message handler from the list of mport specific pw handlers
```

### Parameters

**struct rio\_mport \* mport** RIO master port to bind the portwrite callback

**void \* context** Registered handler specific context to pass on event

**int (\*)(struct rio\_mport \*mport, void \*context, union rio\_pw\_msg \*msg, int step)**  
Registered callback function

### Description

Returns 0 if the request has been satisfied.

**int rio\_request\_inb\_pwrite**(struct rio\_dev \*rdev, int (\*pwcbk)(struct rio\_dev \*rdev, union rio\_pw\_msg \*msg, int step))  
request inbound port-write message service for specific RapidIO device

### Parameters

**struct rio\_dev \* rdev** RIO device to which register inbound port-write callback routine

**int (\*)(struct rio\_dev \*rdev, union rio\_pw\_msg \*msg, int step) pwcbk**  
Callback routine to execute when port-write is received

### Description

Binds a port-write callback function to the RapidIO device. Returns 0 if the request has been satisfied.

**int rio\_release\_inb\_pwrite**(struct rio\_dev \*rdev)  
release inbound port-write message service associated with specific RapidIO device

### Parameters

**struct rio\_dev \* rdev** RIO device which registered for inbound port-write callback

### Description

Removes callback from the rio\_dev structure. Returns 0 if the request has been satisfied.

**void rio\_pw\_enable**(struct rio\_mport \*mport, int enable)  
Enables/disables port-write handling by a master port

### Parameters

**struct rio\_mport \* mport** Master port associated with port-write handling

**int enable** 1=enable, 0=disable

**int rio\_map\_inb\_region**(struct rio\_mport \*mport, dma\_addr\_t local, u64 rbase, u32 size, u32 rflags)

- Map inbound memory region.

### Parameters

**struct rio\_mport \* mport** Master port.

**dma\_addr\_t local** physical address of memory region to be mapped

**u64 rbase** RIO base address assigned to this window

**u32 size** Size of the memory region

**u32 rflags** Flags for mapping.

**Return**

0 - Success.

**Description**

This function will create the mapping from RIO space to local memory.

```
void rio_unmap_inb_region(struct rio_mport * mport, dma_addr_t lstart)
```

- Unmap the inbound memory region

**Parameters**

**struct rio\_mport \* mport** Master port

**dma\_addr\_t lstart** physical address of memory region to be unmapped

```
int rio_map_outb_region(struct rio_mport * mport, u16 destid, u64 rbase,  
                       u32 size, u32 rflags, dma_addr_t * local)
```

- Map outbound memory region.

**Parameters**

**struct rio\_mport \* mport** Master port.

**u16 destid** destination id window points to

**u64 rbase** RIO base address window translates to

**u32 size** Size of the memory region

**u32 rflags** Flags for mapping.

**dma\_addr\_t \* local** physical address of memory region mapped

**Return**

0 - Success.

**Description**

This function will create the mapping from RIO space to local memory.

```
void rio_unmap_outb_region(struct rio_mport * mport, u16 destid,  
                          u64 rstart)
```

- Unmap the inbound memory region

**Parameters**

**struct rio\_mport \* mport** Master port

**u16 destid** destination id mapping points to

**u64 rstart** RIO base address window translates to

```
u32 rio_mport_get_physefb(struct rio_mport * port, int local, u16 destid,  
                        u8 hopcount, u32 * rmap)
```

Helper function that returns register offset for Physical Layer Extended Features Block.

**Parameters**

**struct rio\_mport \* port** Master port to issue transaction

**int local** Indicate a local master port or remote device access

**u16 destid** Destination ID of the device

**u8 hopcount** Number of switch hops to the device

**u32 \* rmap** pointer to location to store register map type info

**struct rio\_dev \* rio\_get\_comptag**(u32 comp\_tag, struct rio\_dev \* from)  
Begin or continue searching for a RIO device by component tag

### Parameters

**u32 comp\_tag** RIO component tag to match

**struct rio\_dev \* from** Previous RIO device found in search, or NULL for new search

### Description

Iterates through the list of known RIO devices. If a RIO device is found with a matching **comp\_tag**, a pointer to its device structure is returned. Otherwise, NULL is returned. A new search is initiated by passing NULL to the **from** argument. Otherwise, if **from** is not NULL, searches continue from next device on the global list.

**int rio\_set\_port\_lockout**(struct rio\_dev \* rdev, u32 pnum, int lock)  
Sets/clears LOCKOUT bit (RIO EM 1.3) for a switch port.

### Parameters

**struct rio\_dev \* rdev** Pointer to RIO device control structure

**u32 pnum** Switch port number to set LOCKOUT bit

**int lock** Operation : set (=1) or clear (=0)

**int rio\_enable\_rx\_tx\_port**(struct rio\_mport \* port, int local, u16 destid,  
u8 hopcount, u8 port\_num)  
enable input receiver and output transmitter of given port

### Parameters

**struct rio\_mport \* port** Master port associated with the RIO network

**int local** local=1 select local port otherwise a far device is reached

**u16 destid** Destination ID of the device to check host bit

**u8 hopcount** Number of hops to reach the target

**u8 port\_num** Port (-number on switch) to enable on a far end device

### Description

Returns 0 or 1 from on General Control Command and Status Register (EXT\_PTR+0x3C)

**int rio\_mport\_chk\_dev\_access**(struct rio\_mport \* mport, u16 destid,  
u8 hopcount)  
Validate access to the specified device.

### Parameters

**struct rio\_mport \* mport** Master port to send transactions  
**u16 destid** Device destination ID in network  
**u8 hopcount** Number of hops into the network  
**int rio\_inb\_pwrite\_handler**(struct rio\_mport \* mport, union rio\_pw\_msg \* pw\_msg)  
inbound port-write message handler

#### Parameters

**struct rio\_mport \* mport** mport device associated with port-write  
**union rio\_pw\_msg \* pw\_msg** pointer to inbound port-write message

#### Description

Processes an inbound port-write message. Returns 0 if the request has been satisfied.

**u32 rio\_mport\_get\_efb**(struct rio\_mport \* port, int local, u16 destid, u8 hopcount, u32 from)  
get pointer to next extended features block

#### Parameters

**struct rio\_mport \* port** Master port to issue transaction  
**int local** Indicate a local master port or remote device access  
**u16 destid** Destination ID of the device  
**u8 hopcount** Number of switch hops to the device  
**u32 from** Offset of current Extended Feature block header (if 0 starts from ExtFeaturePtr)  
**u32 rio\_mport\_get\_feature**(struct rio\_mport \* port, int local, u16 destid, u8 hopcount, int ftr)  
query for devices' extended features

#### Parameters

**struct rio\_mport \* port** Master port to issue transaction  
**int local** Indicate a local master port or remote device access  
**u16 destid** Destination ID of the device  
**u8 hopcount** Number of switch hops to the device  
**int ftr** Extended feature code

#### Description

Tell if a device supports a given RapidIO capability. Returns the offset of the requested extended feature block within the device' s RIO configuration space or 0 in case the device does not support it.

**struct rio\_dev \* rio\_get\_asm**(u16 vid, u16 did, u16 asm\_vid, u16 asm\_did, struct rio\_dev \* from)  
Begin or continue searching for a RIO device by vid/did/asm\_vid/asm\_did

#### Parameters

**u16 vid** RIO vid to match or RIO\_ANY\_ID to match all vids

**u16 did** RIO did to match or RIO\_ANY\_ID to match all dids

**u16 asm\_vid** RIO asm\_vid to match or RIO\_ANY\_ID to match all asm\_vids

**u16 asm\_did** RIO asm\_did to match or RIO\_ANY\_ID to match all asm\_dids

**struct rio\_dev \* from** Previous RIO device found in search, or NULL for new search

### Description

Iterates through the list of known RIO devices. If a RIO device is found with a matching **vid**, **did**, **asm\_vid**, **asm\_did**, the reference count to the device is incremented and a pointer to its device structure is returned. Otherwise, NULL is returned. A new search is initiated by passing NULL to the **from** argument. Otherwise, if **from** is not NULL, searches continue from next device on the global list. The reference count for **from** is always decremented if it is not NULL.

```
struct rio_dev * rio_get_device(u16 vid, u16 did, struct rio_dev * from)
    Begin or continue searching for a RIO device by vid/did
```

### Parameters

**u16 vid** RIO vid to match or RIO\_ANY\_ID to match all vids

**u16 did** RIO did to match or RIO\_ANY\_ID to match all dids

**struct rio\_dev \* from** Previous RIO device found in search, or NULL for new search

### Description

Iterates through the list of known RIO devices. If a RIO device is found with a matching **vid** and **did**, the reference count to the device is incremented and a pointer to its device structure is returned. Otherwise, NULL is returned. A new search is initiated by passing NULL to the **from** argument. Otherwise, if **from** is not NULL, searches continue from next device on the global list. The reference count for **from** is always decremented if it is not NULL.

```
int rio_lock_device(struct rio_mport * port, u16 destid, u8 hopcount,
                    int wait_ms)
    Acquires host device lock for specified device
```

### Parameters

**struct rio\_mport \* port** Master port to send transaction

**u16 destid** Destination ID for device/switch

**u8 hopcount** Hopcount to reach switch

**int wait\_ms** Max wait time in msec (0 = no timeout)

### Description

Attempts to acquire host device lock for specified device Returns 0 if device lock acquired or EINVAL if timeout expires.

```
int rio_unlock_device(struct rio_mport * port, u16 destid, u8 hopcount)
    Releases host device lock for specified device
```

**Parameters**

**struct rio\_mport \* port** Master port to send transaction

**u16 destid** Destination ID for device/switch

**u8 hopcount** Hopcount to reach switch

**Description**

Returns 0 if device lock released or EINVAL if fails.

int **rio\_route\_add\_entry**(struct rio\_dev \* rdev, u16 table, u16 route\_destid,  
u8 route\_port, int lock)

Add a route entry to a switch routing table

**Parameters**

**struct rio\_dev \* rdev** RIO device

**u16 table** Routing table ID

**u16 route\_destid** Destination ID to be routed

**u8 route\_port** Port number to be routed

**int lock** apply a hardware lock on switch device flag (1=lock, 0=no\_lock)

**Description**

If available calls the switch specific add\_entry() method to add a route entry into a switch routing table. Otherwise uses standard RT update method as defined by RapidIO specification. A specific routing table can be selected using the **table** argument if a switch has per port routing tables or the standard (or global) table may be used by passing RIO\_GLOBAL\_TABLE in **table**.

Returns 0 on success or -EINVAL on failure.

int **rio\_route\_get\_entry**(struct rio\_dev \* rdev, u16 table, u16 route\_destid,  
u8 \* route\_port, int lock)

Read an entry from a switch routing table

**Parameters**

**struct rio\_dev \* rdev** RIO device

**u16 table** Routing table ID

**u16 route\_destid** Destination ID to be routed

**u8 \* route\_port** Pointer to read port number into

**int lock** apply a hardware lock on switch device flag (1=lock, 0=no\_lock)

**Description**

If available calls the switch specific get\_entry() method to fetch a route entry from a switch routing table. Otherwise uses standard RT read method as defined by RapidIO specification. A specific routing table can be selected using the **table** argument if a switch has per port routing tables or the standard (or global) table may be used by passing RIO\_GLOBAL\_TABLE in **table**.

Returns 0 on success or -EINVAL on failure.

int **rio\_route\_clr\_table**(struct rio\_dev \* rdev, u16 table, int lock)  
Clear a switch routing table

### Parameters

**struct rio\_dev \* rdev** RIO device

**u16 table** Routing table ID

**int lock** apply a hardware lock on switch device flag (1=lock, 0=no\_lock)

### Description

If available calls the switch specific `clr_table()` method to clear a switch routing table. Otherwise uses standard RT write method as defined by RapidIO specification. A specific routing table can be selected using the **table** argument if a switch has per port routing tables or the standard (or global) table may be used by passing `RIO_GLOBAL_TABLE` in **table**.

Returns 0 on success or -EINVAL on failure.

struct dma\_chan \* **rio\_request\_mport\_dma**(struct rio\_mport \* mport)  
request RapidIO capable DMA channel associated with specified local RapidIO mport device.

### Parameters

**struct rio\_mport \* mport** RIO mport to perform DMA data transfers

### Description

Returns pointer to allocated DMA channel or NULL if failed.

struct dma\_chan \* **rio\_request\_dma**(struct rio\_dev \* rdev)  
request RapidIO capable DMA channel that supports specified target RapidIO device.

### Parameters

**struct rio\_dev \* rdev** RIO device associated with DMA transfer

### Description

Returns pointer to allocated DMA channel or NULL if failed.

void **rio\_release\_dma**(struct dma\_chan \* dchan)  
release specified DMA channel

### Parameters

**struct dma\_chan \* dchan** DMA channel to release

struct dma\_async\_tx\_descriptor \* **rio\_dma\_prep\_xfer**(struct dma\_chan \* dchan, u16 destid, struct rio\_dma\_data \* data, enum dma\_transfer\_direction direction, unsigned long flags)

RapidIO specific wrapper for `device_prep_slave_sg` callback defined by DMAENGINE.

### Parameters

**struct dma\_chan \* dchan** DMA channel to configure

**u16 destid** target RapidIO device destination ID

**struct rio\_dma\_data \* data** RIO specific data descriptor

**enum dma\_transfer\_direction direction** DMA data transfer direction (TO or FROM the device)

**unsigned long flags** dmaengine defined flags

### Description

Initializes RapidIO capable DMA channel for the specified data transfer. Uses DMA channel private extension to pass information related to remote target RIO device.

### Return

**pointer to DMA transaction descriptor if successful**, error-valued pointer or NULL if failed.

```
struct dma_async_tx_descriptor * rio_dma_prep_slave_sg(struct rio_dev
                                                    * rdev, struct
                                                    dma_chan
                                                    * dchan, struct
                                                    rio_dma_data
                                                    * data, enum
                                                    dma_transfer_direction direction,
                                                    unsigned
                                                    long flags)
    RapidIO specific wrapper for device_prep_slave_sg callback defined by
    DMAENGINE.
```

### Parameters

**struct rio\_dev \* rdev** RIO device control structure

**struct dma\_chan \* dchan** DMA channel to configure

**struct rio\_dma\_data \* data** RIO specific data descriptor

**enum dma\_transfer\_direction direction** DMA data transfer direction (TO or FROM the device)

**unsigned long flags** dmaengine defined flags

### Description

Initializes RapidIO capable DMA channel for the specified data transfer. Uses DMA channel private extension to pass information related to remote target RIO device.

### Return

**pointer to DMA transaction descriptor if successful**, error-valued pointer or NULL if failed.

```
int rio_register_scan(int mport_id, struct rio_scan * scan_ops)
    enumeration/discovery method registration interface
```

### Parameters

**int mport\_id** mport device ID for which fabric scan routine has to be set (RIO\_MPORT\_ANY = set for all available mports)

**struct rio\_scan \* scan\_ops** enumeration/discovery operations structure

### Description

Registers enumeration/discovery operations with RapidIO subsystem and attaches it to the specified mport device (or all available mports if RIO\_MPORT\_ANY is specified).

Returns error if the mport already has an enumerator attached to it. In case of RIO\_MPORT\_ANY skips mports with valid scan routines (no error).

**int rio\_unregister\_scan**(int mport\_id, struct rio\_scan \* scan\_ops)  
removes enumeration/discovery method from mport

### Parameters

**int mport\_id** mport device ID for which fabric scan routine has to be unregistered (RIO\_MPORT\_ANY = apply to all mports that use the specified scan\_ops)

**struct rio\_scan \* scan\_ops** enumeration/discovery operations structure

### Description

Removes enumeration or discovery method assigned to the specified mport device. If RIO\_MPORT\_ANY is specified, removes the specified operations from all mports that have them attached.

## 63.4 Internals

This chapter contains the autogenerated documentation of the RapidIO subsystem.

### 63.4.1 Structures

**struct rio\_switch**  
RIO switch info

#### Definition

```
struct rio_switch {
    struct list_head node;
    u8 *route_table;
    u32 port_ok;
    struct rio_switch_ops *ops;
    spinlock_t lock;
    struct rio_dev *nextdev[];
};
```

#### Members

**node** Node in global list of switches

**route\_table** Copy of switch routing table

**port\_ok** Status of each port (one bit per port) - OK=1 or UNINIT=0

**ops** pointer to switch-specific operations

**lock** lock to serialize operations updates

**nextdev** Array of per-port pointers to the next attached device

struct **rio\_switch\_ops**

Per-switch operations

### Definition

```
struct rio_switch_ops {
    struct module *owner;
    int (*add_entry) (struct rio_mport *mport, u16 destid, u8 hopcount, u16_
↳table, u16 route_destid, u8 route_port);
    int (*get_entry) (struct rio_mport *mport, u16 destid, u8 hopcount, u16_
↳table, u16 route_destid, u8 *route_port);
    int (*clr_table) (struct rio_mport *mport, u16 destid, u8 hopcount, u16_
↳table);
    int (*set_domain) (struct rio_mport *mport, u16 destid, u8 hopcount, u8_
↳sw_domain);
    int (*get_domain) (struct rio_mport *mport, u16 destid, u8 hopcount, u8_
↳*sw_domain);
    int (*em_init) (struct rio_dev *dev);
    int (*em_handle) (struct rio_dev *dev, u8 swport);
};
```

### Members

**owner** The module owner of this structure

**add\_entry** Callback for switch-specific route add function

**get\_entry** Callback for switch-specific route get function

**clr\_table** Callback for switch-specific clear route table function

**set\_domain** Callback for switch-specific domain setting function

**get\_domain** Callback for switch-specific domain get function

**em\_init** Callback for switch-specific error management init function

**em\_handle** Callback for switch-specific error management handler function

### Description

Defines the operations that are necessary to initialize/control a particular RIO switch device.

struct **rio\_dev**

RIO device info

### Definition

```
struct rio_dev {
    struct list_head global_list;
    struct list_head net_list;
    struct rio_net *net;
    bool do_enum;
```

(continues on next page)

(continued from previous page)

```
u16 did;
u16 vid;
u32 device_rev;
u16 asm_did;
u16 asm_vid;
u16 asm_rev;
u16 efptr;
u32 pef;
u32 swpinfo;
u32 src_ops;
u32 dst_ops;
u32 comp_tag;
u32 phys_efptr;
u32 phys_rmap;
u32 em_efptr;
u64 dma_mask;
struct rio_driver *driver;
struct device dev;
struct resource riores[RIO_MAX_DEV_RESOURCES];
int (*pwcback) (struct rio_dev *rdev, union rio_pw_msg *msg, int step);
u16 destid;
u8 hopcount;
struct rio_dev *prev;
atomic_t state;
struct rio_switch rswitch[];
};
```

### Members

**global\_list** Node in list of all RIO devices

**net\_list** Node in list of RIO devices in a network

**net** Network this device is a part of

**do\_enum** Enumeration flag

**did** Device ID

**vid** Vendor ID

**device\_rev** Device revision

**asm\_did** Assembly device ID

**asm\_vid** Assembly vendor ID

**asm\_rev** Assembly revision

**efptr** Extended feature pointer

**pef** Processing element features

**swpinfo** Switch port info

**src\_ops** Source operation capabilities

**dst\_ops** Destination operation capabilities

**comp\_tag** RIO component tag

**phys\_efptr** RIO device extended features pointer

**phys\_rmap** LP-Serial Register Map Type (1 or 2)

**em\_efptr** RIO Error Management features pointer

**dma\_mask** Mask of bits of RIO address this device implements

**driver** Driver claiming this device

**dev** Device model device

**riores** RIO resources this device owns

**pwcbck** port-write callback function for this device

**destid** Network destination ID (or associated destid for switch)

**hopcount** Hopcount to this device

**prev** Previous RIO device connected to the current one

**state** device state

**rswitch** struct `rio_switch` (if valid for this device)

struct **rio\_msg**  
RIO message event

**Definition**

```
struct rio_msg {
    struct resource *res;
    void (*mcbck) (struct rio_mport * mport, void *dev_id, int mbox, int
↳slot);
};
```

**Members**

**res** Mailbox resource

**mcbck** Message event callback

struct **rio\_dbell**  
RIO doorbell event

**Definition**

```
struct rio_dbell {
    struct list_head node;
    struct resource *res;
    void (*dinb) (struct rio_mport *mport, void *dev_id, u16 src, u16 dst,
↳u16 info);
    void *dev_id;
};
```

**Members**

**node** Node in list of doorbell events

**res** Doorbell resource

**dinb** Doorbell event callback

**dev\_id** Device specific pointer to pass on event

struct **rio\_mport**

RIO master port info

### Definition

```
struct rio_mport {
    struct list_head dbells;
    struct list_head pwrites;
    struct list_head node;
    struct list_head nnode;
    struct rio_net *net;
    struct mutex lock;
    struct resource iores;
    struct resource riores[RIO_MAX_MPORT_RESOURCES];
    struct rio_msg inb_msg[RIO_MAX_MBOX];
    struct rio_msg outb_msg[RIO_MAX_MBOX];
    int host_deviceid;
    struct rio_ops *ops;
    unsigned char id;
    unsigned char index;
    unsigned int sys_size;
    u32 phys_efptr;
    u32 phys_rmap;
    unsigned char name[RIO_MAX_MPORT_NAME];
    struct device dev;
    void *priv;
#ifdef CONFIG_RAPIDIO_DMA_ENGINE;
    struct dma_device      dma;
#endif;
    struct rio_scan *nscan;
    atomic_t state;
    unsigned int pwe_refcnt;
};
```

### Members

**dbells** List of doorbell events

**pwrites** List of portwrite events

**node** Node in global list of master ports

**nnode** Node in network list of master ports

**net** RIO net this mport is attached to

**lock** lock to synchronize lists manipulations

**iores** I/O mem resource that this master port interface owns

**riores** RIO resources that this master port interfaces owns

**inb\_msg** RIO inbound message event descriptors

**outb\_msg** RIO outbound message event descriptors

**host\_deviceid** Host device ID associated with this master port

**ops** configuration space functions

**id** Port ID, unique among all ports

**index** Port index, unique among all port interfaces of the same type

**sys\_size** RapidIO common transport system size

**phys\_efptr** RIO port extended features pointer

**phys\_rmap** LP-Serial EFB Register Mapping type (1 or 2).

**name** Port name string

**dev** device structure associated with an mport

**priv** Master port private data

**dma** DMA device associated with mport

**nscan** RapidIO network enumeration/discovery operations

**state** mport device state

**pwe\_refcnt** port-write enable ref counter to track enable/disable requests

struct **rio\_net**  
RIO network info

**Definition**

```
struct rio_net {
    struct list_head node;
    struct list_head devices;
    struct list_head switches;
    struct list_head mports;
    struct rio_mport *hport;
    unsigned char id;
    struct device dev;
    void *enum_data;
    void (*release)(struct rio_net *net);
};
```

**Members**

**node** Node in global list of RIO networks

**devices** List of devices in this network

**switches** List of switches in this network

**mports** List of master ports accessing this network

**hport** Default port for accessing this network

**id** RIO network ID

**dev** Device object

**enum\_data** private data specific to a network enumerator

**release** enumerator-specific release callback

struct **rio\_mport\_attr**  
RIO mport device attributes

**Definition**

```
struct rio_mport_attr {
    int flags;
    int link_speed;
    int link_width;
    int dma_max_sge;
    int dma_max_size;
    int dma_align;
};
```

### Members

**flags** mport device capability flags

**link\_speed** SRIO link speed value (as defined by RapidIO specification)

**link\_width** SRIO link width value (as defined by RapidIO specification)

**dma\_max\_sge** number of SG list entries that can be handled by DMA channel(s)

**dma\_max\_size** max number of bytes in single DMA transfer (SG entry)

**dma\_align** alignment shift for DMA operations (as for other DMA operations)

struct **rio\_ops**

Low-level RIO configuration space operations

### Definition

```
struct rio_ops {
    int (*lcread) (struct rio_mport *mport, int index, u32 offset, int len, u32 *data);
    int (*lcwrite) (struct rio_mport *mport, int index, u32 offset, int len, u32 data);
    int (*cread) (struct rio_mport *mport, int index, u16 destid, u8 hopcount, u32 offset, int len, u32 *data);
    int (*cwrite) (struct rio_mport *mport, int index, u16 destid, u8 hopcount, u32 offset, int len, u32 data);
    int (*dsend) (struct rio_mport *mport, int index, u16 destid, u16 data);
    int (*pwenable) (struct rio_mport *mport, int enable);
    int (*open_outb_mbox)(struct rio_mport *mport, void *dev_id, int mbox, int entries);
    void (*close_outb_mbox)(struct rio_mport *mport, int mbox);
    int (*open_inb_mbox)(struct rio_mport *mport, void *dev_id, int mbox, int entries);
    void (*close_inb_mbox)(struct rio_mport *mport, int mbox);
    int (*add_outb_message)(struct rio_mport *mport, struct rio_dev *rdev, int mbox, void *buffer, size_t len);
    int (*add_inb_buffer)(struct rio_mport *mport, int mbox, void *buf);
    void (*get_inb_message)(struct rio_mport *mport, int mbox);
    int (*map_inb)(struct rio_mport *mport, dma_addr_t lstart, u64 rstart, u64 size, u32 flags);
    void (*unmap_inb)(struct rio_mport *mport, dma_addr_t lstart);
    int (*query_mport)(struct rio_mport *mport, struct rio_mport_attr *attr);
    int (*map_outb)(struct rio_mport *mport, u16 destid, u64 rstart, u32 size, u32 flags, dma_addr_t *laddr);
    void (*unmap_outb)(struct rio_mport *mport, u16 destid, u64 rstart);
};
```

### Members

**lcread** Callback to perform local (master port) read of config space.

**lcwrite** Callback to perform local (master port) write of config space.

**cread** Callback to perform network read of config space.

**cwrite** Callback to perform network write of config space.

**dsend** Callback to send a doorbell message.

**pwenable** Callback to enable/disable port-write message handling.

**open\_outb\_mbox** Callback to initialize outbound mailbox.

**close\_outb\_mbox** Callback to shut down outbound mailbox.

**open\_inb\_mbox** Callback to initialize inbound mailbox.

**close\_inb\_mbox** Callback to shut down inbound mailbox.

**add\_outb\_message** Callback to add a message to an outbound mailbox queue.

**add\_inb\_buffer** Callback to add a buffer to an inbound mailbox queue.

**get\_inb\_message** Callback to get a message from an inbound mailbox queue.

**map\_inb** Callback to map RapidIO address region into local memory space.

**unmap\_inb** Callback to unmap RapidIO address region mapped with `map_inb()`.

**query\_mport** Callback to query mport device attributes.

**map\_outb** Callback to map outbound address region into local memory space.

**unmap\_outb** Callback to unmap outbound RapidIO address region.

struct **rio\_driver**  
     RIO driver info

### Definition

```
struct rio_driver {
    struct list_head node;
    char *name;
    const struct rio_device_id *id_table;
    int (*probe) (struct rio_dev * dev, const struct rio_device_id * id);
    void (*remove) (struct rio_dev * dev);
    void (*shutdown)(struct rio_dev *dev);
    int (*suspend) (struct rio_dev * dev, u32 state);
    int (*resume) (struct rio_dev * dev);
    int (*enable_wake) (struct rio_dev * dev, u32 state, int enable);
    struct device_driver driver;
};
```

### Members

**node** Node in list of drivers

**name** RIO driver name

**id\_table** RIO device ids to be associated with this driver

**probe** RIO device inserted

**remove** RIO device removed

**shutdown** shutdown notification callback

**suspend** RIO device suspended

**resume** RIO device awakened

**enable\_wake** RIO device enable wake event

**driver** LDM driver struct

### Description

Provides info on a RIO device driver for insertion/removal and power management purposes.

struct **rio\_scan**

RIO enumeration and discovery operations

### Definition

```
struct rio_scan {
    struct module *owner;
    int (*enumerate)(struct rio_mport *mport, u32 flags);
    int (*discover)(struct rio_mport *mport, u32 flags);
};
```

### Members

**owner** The module owner of this structure

**enumerate** Callback to perform RapidIO fabric enumeration.

**discover** Callback to perform RapidIO fabric discovery.

struct **rio\_scan\_node**

list node to register RapidIO enumeration and discovery methods with RapidIO core.

### Definition

```
struct rio_scan_node {
    int mport_id;
    struct list_head node;
    struct rio_scan *ops;
};
```

### Members

**mport\_id** ID of an mport (net) serviced by this enumerator

**node** node in global list of registered enumerators

**ops** RIO enumeration and discovery operations

### 63.4.2 Enumeration and Discovery

u16 **rio\_destid\_alloc**(struct rio\_net \* net)  
Allocate next available destID for given network

#### Parameters

**struct rio\_net \* net** RIO network

#### Description

Returns next available device destination ID for the specified RIO network. Marks allocated ID as one in use. Returns RIO\_INVALID\_DESTID if new destID is not available.

int **rio\_destid\_reserve**(struct rio\_net \* net, u16 destid)  
Reserve the specified destID

#### Parameters

**struct rio\_net \* net** RIO network

**u16 destid** destID to reserve

#### Description

Tries to reserve the specified destID. Returns 0 if successful.

void **rio\_destid\_free**(struct rio\_net \* net, u16 destid)  
free a previously allocated destID

#### Parameters

**struct rio\_net \* net** RIO network

**u16 destid** destID to free

#### Description

Makes the specified destID available for use.

u16 **rio\_destid\_first**(struct rio\_net \* net)  
return first destID in use

#### Parameters

**struct rio\_net \* net** RIO network

u16 **rio\_destid\_next**(struct rio\_net \* net, u16 from)  
return next destID in use

#### Parameters

**struct rio\_net \* net** RIO network

**u16 from** destination ID from which search shall continue

u16 **rio\_get\_device\_id**(struct rio\_mport \* port, u16 destid, u8 hopcount)  
Get the base/extended device id for a device

#### Parameters

**struct rio\_mport \* port** RIO master port

**u16 destid** Destination ID of device

**u8 hopcount** Hopcount to device

### Description

Reads the base/extended device id from a device. Returns the 8/16-bit device ID.

```
void rio_set_device_id(struct rio_mport * port, u16 destid, u8 hopcount,  
                    u16 did)
```

Set the base/extended device id for a device

### Parameters

**struct rio\_mport \* port** RIO master port

**u16 destid** Destination ID of device

**u8 hopcount** Hopcount to device

**u16 did** Device ID value to be written

### Description

Writes the base/extended device id from a device.

```
int rio_clear_locks(struct rio_net * net)
```

Release all host locks and signal enumeration complete

### Parameters

**struct rio\_net \* net** RIO network to run on

### Description

Marks the component tag CSR on each device with the enumeration complete flag. When complete, it then release the host locks on each device. Returns 0 on success or -EINVAL on failure.

```
int rio_enum_host(struct rio_mport * port)
```

Set host lock and initialize host destination ID

### Parameters

**struct rio\_mport \* port** Master port to issue transaction

### Description

Sets the local host master port lock and destination ID register with the host device ID value. The host device ID value is provided by the platform. Returns 0 on success or -1 on failure.

```
int rio_device_has_destid(struct rio_mport * port, int src_ops,  
                        int dst_ops)
```

Test if a device contains a destination ID register

### Parameters

**struct rio\_mport \* port** Master port to issue transaction

**int src\_ops** RIO device source operations

**int dst\_ops** RIO device destination operations

### Description

Checks the provided **src\_ops** and **dst\_ops** for the necessary transaction capabilities that indicate whether or not a device will implement a destination ID register. Returns 1 if true or 0 if false.

```
void rio_release_dev(struct device * dev)
    Frees a RIO device struct
```

#### **Parameters**

**struct device \* dev** LDM device associated with a RIO device struct

#### **Description**

Gets the RIO device struct associated a RIO device struct. The RIO device struct is freed.

```
int rio_is_switch(struct rio_dev * rdev)
    Tests if a RIO device has switch capabilities
```

#### **Parameters**

**struct rio\_dev \* rdev** RIO device

#### **Description**

Gets the RIO device Processing Element Features register contents and tests for switch capabilities. Returns 1 if the device is a switch or 0 if it is not a switch. The RIO device struct is freed.

```
struct rio_dev * rio_setup_device(struct rio_net * net, struct rio_mport
                                   * port, u16 destid, u8 hopcount,
                                   int do_enum)
    Allocates and sets up a RIO device
```

#### **Parameters**

**struct rio\_net \* net** RIO network

**struct rio\_mport \* port** Master port to send transactions

**u16 destid** Current destination ID

**u8 hopcount** Current hopcount

**int do\_enum** Enumeration/Discovery mode flag

#### **Description**

Allocates a RIO device and configures fields based on configuration space contents. If device has a destination ID register, a destination ID is either assigned in enumeration mode or read from configuration space in discovery mode. If the device has switch capabilities, then a switch is allocated and configured appropriately. Returns a pointer to a RIO device on success or NULL on failure.

```
int rio_sport_is_active(struct rio_dev * rdev, int sp)
    Tests if a switch port has an active connection.
```

#### **Parameters**

**struct rio\_dev \* rdev** RapidIO device object

**int sp** Switch port number

### Description

Reads the port error status CSR for a particular switch port to determine if the port has an active link. Returns `RIO_PORT_N_ERR_STS_PORT_OK` if the port is active or `0` if it is inactive.

`u16 rio_get_host_deviceid_lock(struct rio_mport * port, u8 hopcount)`  
Reads the Host Device ID Lock CSR on a device

### Parameters

`struct rio_mport * port` Master port to send transaction

`u8 hopcount` Number of hops to the device

### Description

Used during enumeration to read the Host Device ID Lock CSR on a RIO device. Returns the value of the lock register.

`int rio_enum_peer(struct rio_net * net, struct rio_mport * port, u8 hopcount, struct rio_dev * prev, int prev_port)`  
Recursively enumerate a RIO network through a master port

### Parameters

`struct rio_net * net` RIO network being enumerated

`struct rio_mport * port` Master port to send transactions

`u8 hopcount` Number of hops into the network

`struct rio_dev * prev` Previous RIO device connected to the enumerated one

`int prev_port` Port on previous RIO device

### Description

Recursively enumerates a RIO network. Transactions are sent via the master port passed in `port`.

`int rio_enum_complete(struct rio_mport * port)`  
Tests if enumeration of a network is complete

### Parameters

`struct rio_mport * port` Master port to send transaction

### Description

Tests the PGCCSR discovered bit for non-zero value (enumeration complete flag). Return 1 if enumeration is complete or `0` if enumeration is incomplete.

`int rio_disc_peer(struct rio_net * net, struct rio_mport * port, u16 destid, u8 hopcount, struct rio_dev * prev, int prev_port)`  
Recursively discovers a RIO network through a master port

### Parameters

`struct rio_net * net` RIO network being discovered

`struct rio_mport * port` Master port to send transactions

`u16 destid` Current destination ID in network

**u8 hopcount** Number of hops into the network

**struct rio\_dev \* prev** previous rio\_dev

**int prev\_port** previous port number

### Description

Recursively discovers a RIO network. Transactions are sent via the master port passed in **port**.

**int rio\_mport\_is\_active**(struct rio\_mport \* port)  
Tests if master port link is active

### Parameters

**struct rio\_mport \* port** Master port to test

### Description

Reads the port error status CSR for the master port to determine if the port has an active link. Returns `RIO_PORT_N_ERR_STS_PORT_OK` if the master port is active or `0` if it is inactive.

**void rio\_update\_route\_tables**(struct rio\_net \* net)  
Updates route tables in switches

### Parameters

**struct rio\_net \* net** RIO network to run update on

### Description

For each enumerated device, ensure that each switch in a system has correct routing entries. Add routes for devices that were unknown during the first enumeration pass through the switch.

**void rio\_init\_em**(struct rio\_dev \* rdev)  
Initializes RIO Error Management (for switches)

### Parameters

**struct rio\_dev \* rdev** RIO device

### Description

For each enumerated switch, call device-specific error management initialization routine (if supplied by the switch driver).

**int rio\_enum\_mport**(struct rio\_mport \* mport, u32 flags)  
Start enumeration through a master port

### Parameters

**struct rio\_mport \* mport** Master port to send transactions

**u32 flags** Enumeration control flags

### Description

Starts the enumeration process. If somebody has enumerated our master port device, then give up. If not and we have an active link, then start recursive peer enumeration. Returns `0` if enumeration succeeds or `-EBUSY` if enumeration fails.

void **rio\_build\_route\_tables**(struct rio\_net \* net)  
Generate route tables from switch route entries

### Parameters

**struct rio\_net \* net** RIO network to run route tables scan on

### Description

For each switch device, generate a route table by copying existing route entries from the switch.

int **rio\_disc\_mport**(struct rio\_mport \* mport, u32 flags)  
Start discovery through a master port

### Parameters

**struct rio\_mport \* mport** Master port to send transactions

**u32 flags** discovery control flags

### Description

Starts the discovery process. If we have an active link, then wait for the signal that enumeration is complete (if wait is allowed). When enumeration completion is signaled, start recursive peer discovery. Returns 0 if discovery succeeds or -EBUSY on failure.

int **rio\_basic\_attach**(void)

### Parameters

**void** no arguments

### Description

When this enumeration/discovery method is loaded as a module this function registers its specific enumeration and discover routines for all available RapidIO mport devices. The “scan” command line parameter controls ability of the module to start RapidIO enumeration/discovery automatically.

Returns 0 for success or -EIO if unable to register itself.

This enumeration/discovery method cannot be unloaded and therefore does not provide a matching cleanup\_module routine.

### 63.4.3 Driver functionality

int **rio\_setup\_inb\_dbell**(struct rio\_mport \* mport, void \* dev\_id, struct resource \* res, void (\*dinb)(struct rio\_mport \* mport, void \*dev\_id, u16 src, u16 dst, u16 info))  
bind inbound doorbell callback

### Parameters

**struct rio\_mport \* mport** RIO master port to bind the doorbell callback

**void \* dev\_id** Device specific pointer to pass on event

**struct resource \* res** Doorbell message resource

**void (\*) (struct rio\_mport \* mport, void \*dev\_id, u16 src, u16 dst, u16 info) di**  
 Callback to execute when doorbell is received

### Description

Adds a doorbell resource/callback pair into a port' s doorbell event list. Returns 0 if the request has been satisfied.

**int rio\_chk\_dev\_route**(struct rio\_dev \* rdev, struct rio\_dev \*\* nrdev, int \* npnum)  
 Validate route to the specified device.

### Parameters

**struct rio\_dev \* rdev** RIO device failed to respond

**struct rio\_dev \*\* nrdev** Last active device on the route to rdev

**int \* npnum** nrdev' s port number on the route to rdev

### Description

Follows a route to the specified RIO device to determine the last available device (and corresponding RIO port) on the route.

**int rio\_chk\_dev\_access**(struct rio\_dev \* rdev)  
 Validate access to the specified device.

### Parameters

**struct rio\_dev \* rdev** Pointer to RIO device control structure

**int rio\_get\_input\_status**(struct rio\_dev \* rdev, int pnum, u32 \* lnkresp)  
 Sends a Link-Request/Input-Status control symbol and returns link-response (if requested).

### Parameters

**struct rio\_dev \* rdev** RIO devive to issue Input-status command

**int pnum** Device port number to issue the command

**u32 \* lnkresp** Response from a link partner

**int rio\_clr\_err\_stopped**(struct rio\_dev \* rdev, u32 pnum, u32 err\_status)  
 Clears port Error-stopped states.

### Parameters

**struct rio\_dev \* rdev** Pointer to RIO device control structure

**u32 pnum** Switch port number to clear errors

**u32 err\_status** port error status (if 0 reads register from device)

### Description

TODO: Currently this routine is not compatible with recovery process specified for idt\_gen3 RapidIO switch devices. It has to be reviewed to implement universal recovery process that is compatible full range off available devices. IDT gen3 switch driver now implements HW-specific error handler that issues soft port reset to the port to reset ERR\_STOP bits and ackIDs.

```
int rio_std_route_add_entry(struct rio_mport * mport, u16 destid,  
                           u8 hopcount, u16 table, u16 route_destid,  
                           u8 route_port)
```

Add switch route table entry using standard registers defined in RIO specification rev.1.3

### Parameters

**struct rio\_mport \* mport** Master port to issue transaction

**u16 destid** Destination ID of the device

**u8 hopcount** Number of switch hops to the device

**u16 table** routing table ID (global or port-specific)

**u16 route\_destid** destID entry in the RT

**u8 route\_port** destination port for specified destID

```
int rio_std_route_get_entry(struct rio_mport * mport, u16 destid,  
                           u8 hopcount, u16 table, u16 route_destid,  
                           u8 * route_port)
```

Read switch route table entry (port number) associated with specified destID using standard registers defined in RIO specification rev.1.3

### Parameters

**struct rio\_mport \* mport** Master port to issue transaction

**u16 destid** Destination ID of the device

**u8 hopcount** Number of switch hops to the device

**u16 table** routing table ID (global or port-specific)

**u16 route\_destid** destID entry in the RT

**u8 \* route\_port** returned destination port for specified destID

```
int rio_std_route_clr_table(struct rio_mport * mport, u16 destid,  
                           u8 hopcount, u16 table)
```

Clear switch route table using standard registers defined in RIO specification rev.1.3.

### Parameters

**struct rio\_mport \* mport** Master port to issue transaction

**u16 destid** Destination ID of the device

**u8 hopcount** Number of switch hops to the device

**u16 table** routing table ID (global or port-specific)

```
struct rio_mport * rio_find_mport(int mport_id)
```

find RIO mport by its ID

### Parameters

**int mport\_id** number (ID) of mport device

### Description

Given a RIO mport number, the desired mport is located in the global list of mports. If the mport is found, a pointer to its data structure is returned. If no mport is found, NULL is returned.

int **rio\_mport\_scan**(int mport\_id)  
    execute enumeration/discovery on the specified mport

**Parameters**

**int mport\_id** number (ID) of mport device

**RIO\_LOP\_READ**(size, type, len)  
    Generate rio\_local\_read\_config\_\* functions

**Parameters**

**size** Size of configuration space read (8, 16, 32 bits)

**type** C type of value argument

**len** Length of configuration space read (1, 2, 4 bytes)

**Description**

Generates rio\_local\_read\_config\_\* functions used to access configuration space registers on the local device.

**RIO\_LOP\_WRITE**(size, type, len)  
    Generate rio\_local\_write\_config\_\* functions

**Parameters**

**size** Size of configuration space write (8, 16, 32 bits)

**type** C type of value argument

**len** Length of configuration space write (1, 2, 4 bytes)

**Description**

Generates rio\_local\_write\_config\_\* functions used to access configuration space registers on the local device.

**RIO\_OP\_READ**(size, type, len)  
    Generate rio\_mport\_read\_config\_\* functions

**Parameters**

**size** Size of configuration space read (8, 16, 32 bits)

**type** C type of value argument

**len** Length of configuration space read (1, 2, 4 bytes)

**Description**

Generates rio\_mport\_read\_config\_\* functions used to access configuration space registers on the local device.

**RIO\_OP\_WRITE**(size, type, len)  
    Generate rio\_mport\_write\_config\_\* functions

**Parameters**

**size** Size of configuration space write (8, 16, 32 bits)

**type** C type of value argument

**len** Length of configuration space write (1, 2, 4 bytes)

### Description

Generates `rio_mport_write_config_*` functions used to access configuration space registers on the local device.

### 63.4.4 Device model support

```
const struct rio_device_id * rio_match_device(const struct rio_device_id
                                             * id, const struct rio_dev
                                             * rdev)
```

Tell if a RIO device has a matching RIO device id structure

#### Parameters

**const struct rio\_device\_id \* id** the RIO device id structure to match against

**const struct rio\_dev \* rdev** the RIO device structure to match against

Used from driver probe and bus matching to check whether a RIO device matches a device id structure provided by a RIO driver. Returns the matching `struct rio_device_id` or `NULL` if there is no match.

```
int rio_device_probe(struct device * dev)
```

Tell if a RIO device structure has a matching RIO device id structure

#### Parameters

**struct device \* dev** the RIO device structure to match against

### Description

return 0 and set `rio_dev->driver` when drv claims `rio_dev`, else error

```
int rio_device_remove(struct device * dev)
```

Remove a RIO device from the system

#### Parameters

**struct device \* dev** the RIO device structure to match against

### Description

Remove a RIO device from the system. If it has an associated driver, then run the driver `remove()` method. Then update the reference count.

```
int rio_match_bus(struct device * dev, struct device_driver * drv)
```

Tell if a RIO device structure has a matching RIO driver device id structure

#### Parameters

**struct device \* dev** the standard device structure to match against

**struct device\_driver \* drv** the standard driver structure containing the ids to match against

Used by a driver to check whether a RIO device present in the system is in its list of supported devices. Returns 1 if there is a matching `struct rio_device_id` or 0 if there is no match.

int **rio\_bus\_init**(void)

Register the RapidIO bus with the device model

### Parameters

**void** no arguments

### Description

Registers the RIO mport device class and RIO bus type with the Linux device model.

## 63.4.5 PPC32 support

int **fsl\_local\_config\_read**(struct rio\_mport \* mport, int index, u32 offset,  
int len, u32 \* data)

Generate a MPC85xx local config space read

### Parameters

**struct rio\_mport \* mport** RapidIO master port info

**int index** ID of RapdiIO interface

**u32 offset** Offset into configuration space

**int len** Length (in bytes) of the maintenance transaction

**u32 \* data** Value to be read into

### Description

Generates a MPC85xx local configuration space read. Returns 0 on success or -EINVAL on failure.

int **fsl\_local\_config\_write**(struct rio\_mport \* mport, int index, u32 offset,  
int len, u32 data)

Generate a MPC85xx local config space write

### Parameters

**struct rio\_mport \* mport** RapidIO master port info

**int index** ID of RapdiIO interface

**u32 offset** Offset into configuration space

**int len** Length (in bytes) of the maintenance transaction

**u32 data** Value to be written

### Description

Generates a MPC85xx local configuration space write. Returns 0 on success or -EINVAL on failure.

int **fsl\_rio\_config\_read**(struct rio\_mport \* mport, int index, u16 destid,  
u8 hopcount, u32 offset, int len, u32 \* val)

Generate a MPC85xx read maintenance transaction

### Parameters

**struct rio\_mport \* mport** RapidIO master port info

**int index** ID of RapdiIO interface  
**u16 destid** Destination ID of transaction  
**u8 hopcount** Number of hops to target device  
**u32 offset** Offset into configuration space  
**int len** Length (in bytes) of the maintenance transaction  
**u32 \* val** Location to be read into

### Description

Generates a MPC85xx read maintenance transaction. Returns 0 on success or -EINVAL on failure.

```
int fsl_rio_config_write(struct rio_mport * mport, int index, u16 destid,  
                       u8 hopcount, u32 offset, int len, u32 val)  
    Generate a MPC85xx write maintenance transaction
```

### Parameters

**struct rio\_mport \* mport** RapidIO master port info  
**int index** ID of RapdiIO interface  
**u16 destid** Destination ID of transaction  
**u8 hopcount** Number of hops to target device  
**u32 offset** Offset into configuration space  
**int len** Length (in bytes) of the maintenance transaction  
**u32 val** Value to be written

### Description

Generates an MPC85xx write maintenance transaction. Returns 0 on success or -EINVAL on failure.

```
int fsl_rio_setup(struct platform_device * dev)  
    Setup Freescale PowerPC RapidIO interface
```

### Parameters

**struct platform\_device \* dev** platform\_device pointer

### Description

Initializes MPC85xx RapidIO hardware interface, configures master port with system-specific info, and registers the master port with the RapidIO subsystem.

## 63.5 Credits

The following people have contributed to the RapidIO subsystem directly or indirectly:

1. [Matt Porter](mailto:MattPorter@kernel.crashing.org)[porter@kernel.crashing.org](mailto:porter@kernel.crashing.org)
2. [Randy Vinson](mailto:RandyVinson@mvista.com)[rvinson@mvista.com](mailto:rvinson@mvista.com)
3. [Dan Malek](mailto:DanMalekdan@embeddedalley.com)[dan@embeddedalley.com](mailto:dan@embeddedalley.com)

The following people have contributed to this document:

1. [Matt Porter](mailto:MattPorter@kernel.crashing.org)[porter@kernel.crashing.org](mailto:porter@kernel.crashing.org)



## **RELIABILITY, AVAILABILITY AND SERVICEABILITY**

### **64.1 RAS concepts**

Reliability, Availability and Serviceability (RAS) is a concept used on servers meant to measure their robustness.

**Reliability** is the probability that a system will produce correct outputs.

- Generally measured as Mean Time Between Failures (MTBF)
- Enhanced by features that help to avoid, detect and repair hardware faults

**Availability** is the probability that a system is operational at a given time

- Generally measured as a percentage of downtime per a period of time
- Often uses mechanisms to detect and correct hardware faults in runtime;

**Serviceability (or maintainability)** is the simplicity and speed with which a system can be repaired or maintained

- Generally measured on Mean Time Between Repair (MTBR)

#### **64.1.1 Improving RAS**

In order to reduce systems downtime, a system should be capable of detecting hardware errors, and, when possible correcting them in runtime. It should also provide mechanisms to detect hardware degradation, in order to warn the system administrator to take the action of replacing a component before it causes data loss or system downtime.

Among the monitoring measures, the most usual ones include:

- CPU - detect errors at instruction execution and at L1/L2/L3 caches;
- Memory - add error correction logic (ECC) to detect and correct errors;
- I/O - add CRC checksums for transferred data;
- Storage - RAID, journal file systems, checksums, Self-Monitoring, Analysis and Reporting Technology (SMART).

By monitoring the number of occurrences of error detections, it is possible to identify if the probability of hardware errors is increasing, and, on such case, do a

preventive maintenance to replace a degraded component while those errors are correctable.

### 64.1.2 Types of errors

Most mechanisms used on modern systems use technologies like Hamming Codes that allow error correction when the number of errors on a bit packet is below a threshold. If the number of errors is above, those mechanisms can indicate with a high degree of confidence that an error happened, but they can't correct.

Also, sometimes an error occur on a component that it is not used. For example, a part of the memory that it is not currently allocated.

That defines some categories of errors:

- **Correctable Error (CE)** - the error detection mechanism detected and corrected the error. Such errors are usually not fatal, although some Kernel mechanisms allow the system administrator to consider them as fatal.
- **Uncorrected Error (UE)** - the amount of errors happened above the error correction threshold, and the system was unable to auto-correct.
- **Fatal Error** - when an UE error happens on a critical component of the system (for example, a piece of the Kernel got corrupted by an UE), the only reliable way to avoid data corruption is to hang or reboot the machine.
- **Non-fatal Error** - when an UE error happens on an unused component, like a CPU in power down state or an unused memory bank, the system may still run, eventually replacing the affected hardware by a hot spare, if available.

Also, when an error happens on a userspace process, it is also possible to kill such process and let userspace restart it.

The mechanism for handling non-fatal errors is usually complex and may require the help of some userspace application, in order to apply the policy desired by the system administrator.

### 64.1.3 Identifying a bad hardware component

Just detecting a hardware flaw is usually not enough, as the system needs to pinpoint to the minimal replaceable unit (MRU) that should be exchanged to make the hardware reliable again.

So, it requires not only error logging facilities, but also mechanisms that will translate the error message to the silkscreen or component label for the MRU.

Typically, it is very complex for memory, as modern CPUs interlace memory from different memory modules, in order to provide a better performance. The DMI BIOS usually have a list of memory module labels, with can be obtained using the dmidecode tool. For example, on a desktop machine, it shows:

```
Memory Device
  Total Width: 64 bits
  Data Width: 64 bits
  Size: 16384 MB
```

(continues on next page)

(continued from previous page)

```
Form Factor: SODIMM
Set: None
Locator: ChannelA-DIMM0
Bank Locator: BANK 0
Type: DDR4
Type Detail: Synchronous
Speed: 2133 MHz
Rank: 2
Configured Clock Speed: 2133 MHz
```

On the above example, a DDR4 SO-DIMM memory module is located at the system's memory labeled as "BANK 0", as given by the bank locator field. Please notice that, on such system, the total width is equal to the data width. It means that such memory module doesn't have error detection/correction mechanisms.

Unfortunately, not all systems use the same field to specify the memory bank. On this example, from an older server, dmidecode shows:

```
Memory Device
  Array Handle: 0x1000
  Error Information Handle: Not Provided
  Total Width: 72 bits
  Data Width: 64 bits
  Size: 8192 MB
  Form Factor: DIMM
  Set: 1
  Locator: DIMM_A1
  Bank Locator: Not Specified
  Type: DDR3
  Type Detail: Synchronous Registered (Buffered)
  Speed: 1600 MHz
  Rank: 2
  Configured Clock Speed: 1600 MHz
```

There, the DDR3 RDIMM memory module is located at the system's memory labeled as "DIMM\_A1", as given by the locator field. Please notice that this memory module has 64 bits of data width and 72 bits of total width. So, it has 8 extra bits to be used by error detection and correction mechanisms. Such kind of memory is called Error-correcting code memory (ECC memory).

To make things even worse, it is not uncommon that systems with different labels on their system's board to use exactly the same BIOS, meaning that the labels provided by the BIOS won't match the real ones.

### 64.1.4 ECC memory

As mentioned in the previous section, ECC memory has extra bits to be used for error correction. In the above example, a memory module has 64 bits of data width, and 72 bits of total width. The extra 8 bits which are used for the error detection and correction mechanisms are referred to as the syndrome<sup>12</sup>.

So, when the cpu requests the memory controller to write a word with data width, the memory controller calculates the syndrome in real time, using Hamming code, or some other error correction code, like SECDED+, producing a code with total width size. Such code is then written on the memory modules.

At read, the total width bits code is converted back, using the same ECC code used on write, producing a word with data width and a syndrome. The word with data width is sent to the CPU, even when errors happen.

The memory controller also looks at the syndrome in order to check if there was an error, and if the ECC code was able to fix such error. If the error was corrected, a Corrected Error (CE) happened. If not, an Uncorrected Error (UE) happened.

The information about the CE/UE errors is stored on some special registers at the memory controller and can be accessed by reading such registers, either by BIOS, by some special CPUs or by Linux EDAC driver. On x86 64 bit CPUs, such errors can also be retrieved via the Machine Check Architecture (MCA)<sup>3</sup>.

## 64.2 EDAC - Error Detection And Correction

---

**Note:** “bluesmoke” was the name for this device driver subsystem when it was “out-of-tree” and maintained at <http://bluesmoke.sourceforge.net>. That site is mostly archaic now and can be used only for historical purposes.

When the subsystem was pushed upstream for the first time, on Kernel 2.6.16, it was renamed to EDAC.

---

---

<sup>1</sup> Please notice that several memory controllers allow operation on a mode called “Lock-Step” , where it groups two memory modules together, doing 128-bit reads/writes. That gives 16 bits for error correction, with significantly improves the error correction mechanism, at the expense that, when an error happens, there’ s no way to know what memory module is to blame. So, it has to blame both memory modules.

<sup>2</sup> Some memory controllers also allow using memory in mirror mode. On such mode, the same data is written to two memory modules. At read, the system checks both memory modules, in order to check if both provide identical data. On such configuration, when an error happens, there’ s no way to know what memory module is to blame. So, it has to blame both memory modules (or 4 memory modules, if the system is also on Lock-step mode).

<sup>3</sup> For more details about the Machine Check Architecture (MCA), please read Documentation/x86/x86\_64/machinecheck.rst at the Kernel tree.

### 64.2.1 Purpose

The edac kernel module' s goal is to detect and report hardware errors that occur within the computer system running under linux.

### 64.2.2 Memory

Memory Correctable Errors (CE) and Uncorrectable Errors (UE) are the primary errors being harvested. These types of errors are harvested by the edac\_mc device.

Detecting CE events, then harvesting those events and reporting them, **can** but must not necessarily be a predictor of future UE events. With CE events only, the system can and will continue to operate as no data has been damaged yet.

However, preventive maintenance and proactive part replacement of memory modules exhibiting CEs can reduce the likelihood of the dreaded UE events and system panics.

### 64.2.3 Other hardware elements

A new feature for EDAC, the edac\_device class of device, was added in the 2.6.23 version of the kernel.

This new device type allows for non-memory type of ECC hardware detectors to have their states harvested and presented to userspace via the sysfs interface.

Some architectures have ECC detectors for L1, L2 and L3 caches, along with DMA engines, fabric switches, main data path switches, interconnections, and various other hardware data paths. If the hardware reports it, then a edac\_device device probably can be constructed to harvest and present that to userspace.

### 64.2.4 PCI bus scanning

In addition, PCI devices are scanned for PCI Bus Parity and SERR Errors in order to determine if errors are occurring during data transfers.

The presence of PCI Parity errors must be examined with a grain of salt. There are several add-in adapters that do **not** follow the PCI specification with regards to Parity generation and reporting. The specification says the vendor should tie the parity status bits to 0 if they do not intend to generate parity. Some vendors do not do this, and thus the parity bit can “float” giving false positives.

There is a PCI device attribute located in sysfs that is checked by the EDAC PCI scanning code. If that attribute is set, PCI parity/error scanning is skipped for that device. The attribute is:

broken_parity_status
----------------------

and is located in /sys/devices/pci<XXX>/0000:XX:YY.Z directories for PCI devices.

### 64.2.5 Versioning

EDAC is composed of a “core” module (`edac_core.ko`) and several Memory Controller (MC) driver modules. On a given system, the CORE is loaded and one MC driver will be loaded. Both the CORE and the MC driver (or `edac_device` driver) have individual versions that reflect current release level of their respective modules.

Thus, to “report” on what version a system is running, one must report both the CORE’ s and the MC driver’ s versions.

### 64.2.6 Loading

If `edac` was statically linked with the kernel then no loading is necessary. If `edac` was built as modules then simply `modprobe` the `edac` pieces that you need. You should be able to `modprobe` hardware-specific modules and have the dependencies load the necessary core modules.

Example:

```
$ modprobe amd76x_edac
```

loads both the `amd76x_edac.ko` memory controller module and the `edac_mc.ko` core module.

### 64.2.7 Sysfs interface

EDAC presents a `sysfs` interface for control and reporting purposes. It lives in the `/sys/devices/system/edac` directory.

Within this directory there currently reside 2 components:

<code>mc</code>	memory controller(s) system
<code>pci</code>	PCI control and status system

### 64.2.8 Memory Controller (mc) Model

Each `mc` device controls a set of memory modules<sup>4</sup>. These modules are laid out in a Chip-Select Row (`csrowX`) and Channel table (`chX`). There can be multiple `csrows` and multiple channels.

Memory controllers allow for several `csrows`, with 8 `csrows` being a typical value. Yet, the actual number of `csrows` depends on the layout of a given motherboard, memory controller and memory module characteristics.

---

<sup>4</sup> Nowadays, the term DIMM (Dual In-line Memory Module) is widely used to refer to a memory module, although there are other memory packaging alternatives, like SO-DIMM, SIMM, etc. The UEFI specification (Version 2.7) defines a memory module in the Common Platform Error Record (CPEP) section to be an SMBIOS Memory Device (Type 17). Along this document, and inside the EDAC subsystem, the term “`dimmm`” is used for all memory modules, even when they use a different kind of packaging.

Dual channels allow for dual data length (e. g. 128 bits, on 64 bit systems) data transfers to/from the CPU from/to memory. Some newer chipsets allow for more than 2 channels, like Fully Buffered DIMMs (FB-DIMMs) memory controllers. The following example will assume 2 channels:

CS Rows	Channels	
	ch0	ch1
	<b>DIMM_A0</b>	<b>DIMM_B0</b>
csrow0	rank0	rank0
csrow1	rank1	rank1
	<b>DIMM_A1</b>	<b>DIMM_B1</b>
csrow2	rank0	rank0
csrow3	rank1	rank1

In the above example, there are 4 physical slots on the motherboard for memory DIMMs:

DIMM_A0	DIMM_B0
DIMM_A1	DIMM_B1

Labels for these slots are usually silk-screened on the motherboard. Slots labeled A are channel 0 in this example. Slots labeled B are channel 1. Notice that there are two csrows possible on a physical DIMM. These csrows are allocated their csrow assignment based on the slot into which the memory DIMM is placed. Thus, when 1 DIMM is placed in each Channel, the csrows cross both DIMMs.

Memory DIMMs come single or dual “ranked”. A rank is a populated csrow. In the example above 2 dual ranked DIMMs are similarly placed. Thus, both csrow0 and csrow1 are populated. On the other hand, when 2 single ranked DIMMs are placed in slots DIMM\_A0 and DIMM\_B0, then they will have just one csrow (csrow0) and csrow1 will be empty. The pattern repeats itself for csrow2 and csrow3. Also note that some memory controllers don’t have any logic to identify the memory module, see rankX directories below.

The representation of the above is reflected in the directory tree in EDAC’s sysfs interface. Starting in directory /sys/devices/system/edac/mc, each memory controller will be represented by its own mcX directory, where X is the index of the MC:

```

.../edac/mc/
    |
    |->mc0
    |->mc1
    |->mc2
    ....

```

Under each mcX directory each cs rowX is again represented by a csrowX, where X is the csrow index:

```

.../mc/mc0/
    |
    |->csrow0

```

(continues on next page)

(continued from previous page)

```
| ->csrow2  
| ->csrow3  
.....
```

Notice that there is no csrow1, which indicates that csrow0 is composed of a single ranked DIMMs. This should also apply in both Channels, in order to have dual-channel mode be operational. Since both csrow2 and csrow3 are populated, this indicates a dual ranked set of DIMMs for channels 0 and 1.

Within each of the mcX and csrowX directories are several EDAC control and attribute files.

### 64.2.9 mcX directories

In mcX directories are EDAC control and attribute files for this X instance of the memory controllers.

For a description of the sysfs API, please see:

[Documentation/ABI/testing/sysfs-devices-edac](#)

### 64.2.10 dimmX or rankX directories

The recommended way to use the EDAC subsystem is to look at the information provided by the dimmX or rankX directories<sup>5</sup>.

A typical EDAC system has the following structure under /sys/devices/system/edac/<sup>6</sup>:

```
/sys/devices/system/edac/  
├── mc  
│   ├── mc0  
│   │   ├── ce_count  
│   │   ├── ce_noinfo_count  
│   │   ├── dimm0  
│   │   │   ├── dimm_ce_count  
│   │   │   ├── dimm_dev_type  
│   │   │   ├── dimm_edac_mode  
│   │   │   ├── dimm_label  
│   │   │   ├── dimm_location  
│   │   │   ├── dimm_mem_type  
│   │   │   ├── dimm_ue_count  
│   │   │   ├── size  
│   │   │   └── uevent  
│   │   ├── max_location  
│   └── mc_name
```

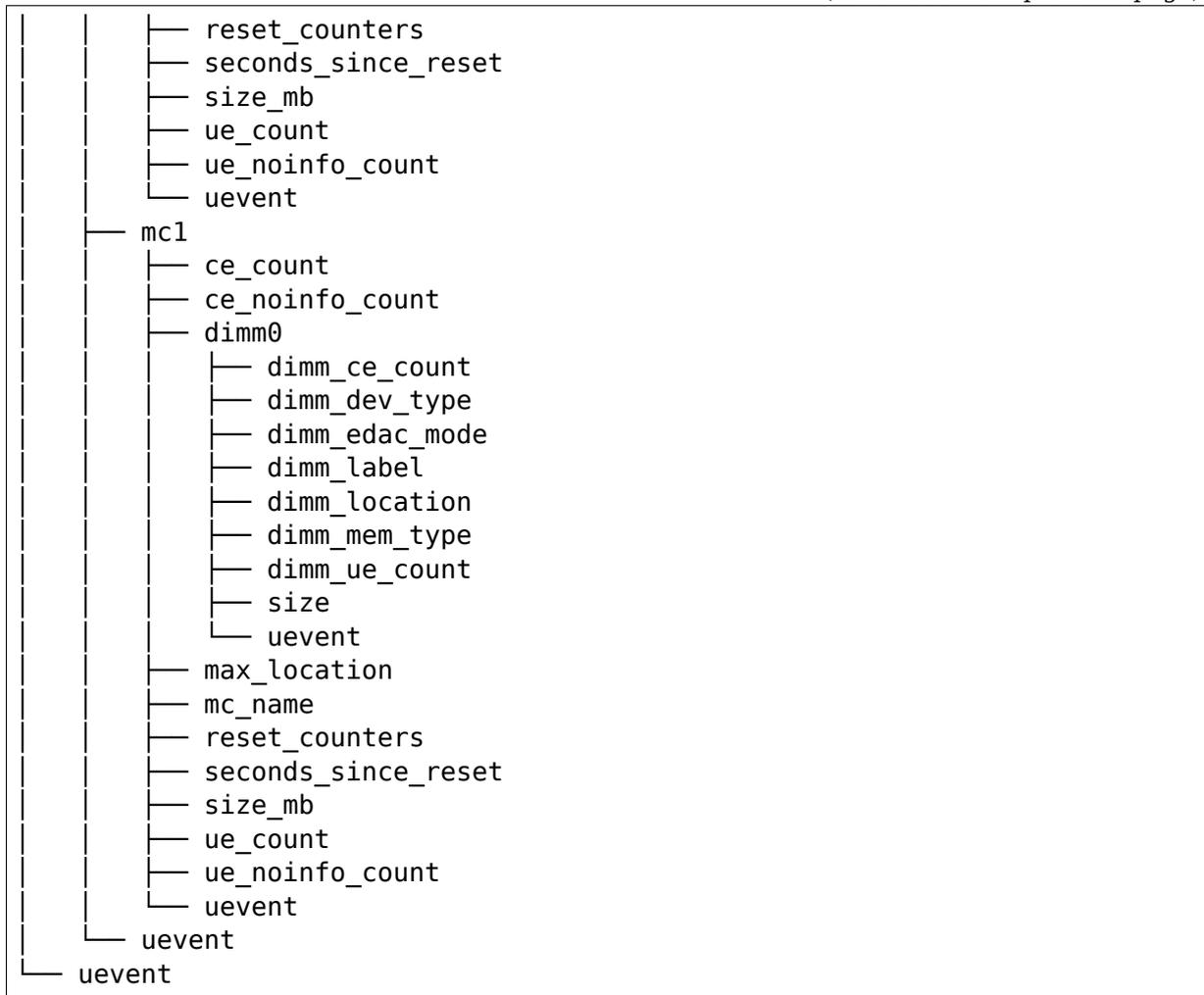
(continues on next page)

---

<sup>5</sup> On some systems, the memory controller doesn't have any logic to identify the memory module. On such systems, the directory is called rankX and works on a similar way as the csrowX directories. On modern Intel memory controllers, the memory controller identifies the memory modules directly. On such systems, the directory is called dimmX.

<sup>6</sup> There are also some power directories and subsystem symlinks inside the sysfs mapping that are automatically created by the sysfs subsystem. Currently, they serve no purpose.

(continued from previous page)



In the `dimmx` directories are EDAC control and attribute files for this `X` memory module:

- `size` - Total memory managed by this csrow attribute file  
This attribute file displays, in count of megabytes, the memory that this csrow contains.
- `dimm_ue_count` - Uncorrectable Errors count attribute file  
This attribute file displays the total count of uncorrectable errors that have occurred on this DIMM. If `panic_on_ue` is set this counter will not have a chance to increment, since EDAC will panic the system.
- `dimm_ce_count` - Correctable Errors count attribute file  
This attribute file displays the total count of correctable errors that have occurred on this DIMM. This count is very important to examine. CEs provide early indications that a DIMM is beginning to fail. This count field should be monitored for non-zero values and report such information to the system administrator.
- `dimm_dev_type` - Device type attribute file

This attribute file will display what type of DRAM device is being utilized on this DIMM. Examples:

- x1
- x2
- x4
- x8

- `dimmedac_mode` - EDAC Mode of operation attribute file

This attribute file will display what type of Error detection and correction is being utilized.

- `dimmlabel` - memory module label control file

This control file allows this DIMM to have a label assigned to it. With this label in the module, when errors occur the output can provide the DIMM label in the system log. This becomes vital for panic events to isolate the cause of the UE event.

DIMM Labels must be assigned after booting, with information that correctly identifies the physical slot with its silk screen label. This information is currently very motherboard specific and determination of this information must occur in userland at this time.

- `dimmlocation` - location of the memory module

The location can have up to 3 levels, and describe how the memory controller identifies the location of a memory module. Depending on the type of memory and memory controller, it can be:

- `csrow` and `channel` - used when the memory controller doesn't identify a single DIMM - e. g. in `rankX` dir;
- `branch`, `channel`, `slot` - typically used on FB-DIMM memory controllers;
- `channel`, `slot` - used on Nehalem and newer Intel drivers.

- `dimmmem_type` - Memory Type attribute file

This attribute file will display what type of memory is currently on this `csrow`. Normally, either buffered or unbuffered memory. Examples:

- Registered-DDR
- Unbuffered-DDR

### 64.2.11 csrowX directories

When CONFIG\_EDAC\_LEGACY\_SYSFS is enabled, sysfs will contain the csrowX directories. As this API doesn't work properly for Rambus, FB-DIMMs and modern Intel Memory Controllers, this is being deprecated in favor of dimmX directories.

In the csrowX directories are EDAC control and attribute files for this X instance of csrow:

- `ue_count` - Total Uncorrectable Errors count attribute file

This attribute file displays the total count of uncorrectable errors that have occurred on this csrow. If `panic_on_ue` is set this counter will not have a chance to increment, since EDAC will panic the system.
- `ce_count` - Total Correctable Errors count attribute file

This attribute file displays the total count of correctable errors that have occurred on this csrow. This count is very important to examine. CEs provide early indications that a DIMM is beginning to fail. This count field should be monitored for non-zero values and report such information to the system administrator.
- `size_mb` - Total memory managed by this csrow attribute file

This attribute file displays, in count of megabytes, the memory that this csrow contains.
- `mem_type` - Memory Type attribute file

This attribute file will display what type of memory is currently on this csrow. Normally, either buffered or unbuffered memory. Examples:

  - Registered-DDR
  - Unbuffered-DDR
- `edac_mode` - EDAC Mode of operation attribute file

This attribute file will display what type of Error detection and correction is being utilized.
- `dev_type` - Device type attribute file

This attribute file will display what type of DRAM device is being utilized on this DIMM. Examples:

  - x1
  - x2
  - x4
  - x8
- `ch0_ce_count` - Channel 0 CE Count attribute file

This attribute file will display the count of CEs on this DIMM located in channel 0.

- `ch0_ue_count` - Channel 0 UE Count attribute file  
This attribute file will display the count of UEs on this DIMM located in channel 0.
- `ch0_dimm_label` - Channel 0 DIMM Label control file  
This control file allows this DIMM to have a label assigned to it. With this label in the module, when errors occur the output can provide the DIMM label in the system log. This becomes vital for panic events to isolate the cause of the UE event.  
  
DIMM Labels must be assigned after booting, with information that correctly identifies the physical slot with its silk screen label. This information is currently very motherboard specific and determination of this information must occur in userland at this time.
- `ch1_ce_count` - Channel 1 CE Count attribute file  
This attribute file will display the count of CEs on this DIMM located in channel 1.
- `ch1_ue_count` - Channel 1 UE Count attribute file  
This attribute file will display the count of UEs on this DIMM located in channel 0.
- `ch1_dimm_label` - Channel 1 DIMM Label control file  
This control file allows this DIMM to have a label assigned to it. With this label in the module, when errors occur the output can provide the DIMM label in the system log. This becomes vital for panic events to isolate the cause of the UE event.  
  
DIMM Labels must be assigned after booting, with information that correctly identifies the physical slot with its silk screen label. This information is currently very motherboard specific and determination of this information must occur in userland at this time.

### 64.2.12 System Logging

If logging for UEs and CEs is enabled, then system logs will contain information indicating that errors have been detected:

```
EDAC MC0: CE page 0x283, offset 0xce0, grain 8, syndrome 0x6ec3, row 0, ↳  
↳channel 1 "DIMM_B1": amd76x_edac  
EDAC MC0: CE page 0x1e5, offset 0xfb0, grain 8, syndrome 0xb741, row 0, ↳  
↳channel 1 "DIMM_B1": amd76x_edac
```

The structure of the message is:

Content	Example
The memory controller	MC0
Error type	CE
Memory page	0x283
Offset in the page	0xce0
The byte granularity or resolution of the error	grain 8
The error syndrome	0xb741
Memory row	row 0
Memory channel	channel 1
DIMM label, if set prior	DIMM B1
And then an optional, driver-specific message that may have additional information.	

Both UEs and CEs with no info will lack all but memory controller, error type, a notice of “no info” and then an optional, driver-specific error message.

### 64.2.13 PCI Bus Parity Detection

On Header Type 00 devices, the primary status is looked at for any parity error regardless of whether parity is enabled on the device or not. (The spec indicates parity is generated in some cases). On Header Type 01 bridges, the secondary status register is also looked at to see if parity occurred on the bus on the other side of the bridge.

### 64.2.14 Sysfs configuration

Under `/sys/devices/system/edac/pci` are control and attribute files as follows:

- `check_pci_parity` - Enable/Disable PCI Parity checking control file

This control file enables or disables the PCI Bus Parity scanning operation. Writing a 1 to this file enables the scanning. Writing a 0 to this file disables the scanning.

Enable:

```
echo "1" >/sys/devices/system/edac/pci/check_pci_parity
```

Disable:

```
echo "0" >/sys/devices/system/edac/pci/check_pci_parity
```

- `pci_parity_count` - Parity Count

This attribute file will display the number of parity errors that have been detected.

### 64.2.15 Module parameters

- edac\_mc\_panic\_on\_ue - Panic on UE control file

An uncorrectable error will cause a machine panic. This is usually desirable. It is a bad idea to continue when an uncorrectable error occurs - it is indeterminate what was uncorrected and the operating system context might be so mangled that continuing will lead to further corruption. If the kernel has MCE configured, then EDAC will never notice the UE.

LOAD TIME:

```
module/kernel parameter: edac_mc_panic_on_ue=[0|1]
```

RUN TIME:

```
echo "1" > /sys/module/edac_core/parameters/edac_mc_panic_on_
↪ue
```

- edac\_mc\_log\_ue - Log UE control file

Generate kernel messages describing uncorrectable errors. These errors are reported through the system message log system. UE statistics will be accumulated even when UE logging is disabled.

LOAD TIME:

```
module/kernel parameter: edac_mc_log_ue=[0|1]
```

RUN TIME:

```
echo "1" > /sys/module/edac_core/parameters/edac_mc_log_ue
```

- edac\_mc\_log\_ce - Log CE control file

Generate kernel messages describing correctable errors. These errors are reported through the system message log system. CE statistics will be accumulated even when CE logging is disabled.

LOAD TIME:

```
module/kernel parameter: edac_mc_log_ce=[0|1]
```

RUN TIME:

```
echo "1" > /sys/module/edac_core/parameters/edac_mc_log_ce
```

- edac\_mc\_poll\_msec - Polling period control file

The time period, in milliseconds, for polling for error information. Too small a value wastes resources. Too large a value might delay necessary handling of errors and might lose valuable information for locating the error. 1000 milliseconds (once each second) is the current default. Systems which require all the bandwidth they can get, may increase this.

LOAD TIME:

```
module/kernel parameter: edac_mc_poll_msec=[0|1]
```

RUN TIME:

```
echo "1000" > /sys/module/edac_core/parameters/edac_mc_poll_
↪msec
```

- `panic_on_pci_parity` - Panic on PCI PARITY Error

This control file enables or disables panicking when a parity error has been detected.

module/kernel parameter:

```
edac_panic_on_pci_pe=[0|1]
```

Enable:

```
echo "1" > /sys/module/edac_core/parameters/edac_panic_on_pci_
↪pe
```

Disable:

```
echo "0" > /sys/module/edac_core/parameters/edac_panic_on_pci_
↪pe
```

### 64.2.16 EDAC device type

In the header file, `edac_pci.h`, there is a series of `edac_device` structures and APIs for the EDAC\_DEVICE.

User space access to an `edac_device` is through the `sysfs` interface.

At the location `/sys/devices/system/edac` (`sysfs`) new `edac_device` devices will appear.

There is a three level tree beneath the above `edac` directory. For example, the `test_device_edac` device (found at the <http://bluesmoke.sourceforge.net> website) installs itself as:

```
/sys/devices/system/edac/test-instance
```

in this directory are various controls, a symlink and one or more instance directories.

The standard default controls are:

<code>log_ce</code>	boolean to log CE events
<code>log_ue</code>	boolean to log UE events
<code>panic_on_</code>	boolean to panic the system if an UE is encountered (default off, can be set true via startup script)
<code>poll_msec</code>	time period between POLL cycles for events

The `test_device_edac` device adds at least one of its own custom control:

<code>test_bits</code>	which in the current test driver does nothing but show how it is installed. A ported driver can add one or more such controls and/or attributes for specific uses. One out-of-tree driver uses controls here to allow for ERROR INJECTION operations to hardware injection registers
------------------------	--

The symlink points to the 'struct dev' that is registered for this `edac_device`.

### 64.2.17 Instances

One or more instance directories are present. For the `test_device_edac` case:

`test-instance0`

In this directory there are two default counter attributes, which are totals of counter in deeper subdirectories.

<code>ce_count</code>	total of CE events of subdirectories
<code>ue_count</code>	total of UE events of subdirectories

### 64.2.18 Blocks

At the lowest directory level is the `block` directory. There can be 0, 1 or more blocks specified in each instance:

`test-block0`

In this directory the default attributes are:

<code>ce_count</code>	which is counter of CE events for this block of hardware being monitored
<code>ue_count</code>	which is counter of UE events for this block of hardware being monitored

The `test_device_edac` device adds 4 attributes and 1 control:

<code>test-block-bits-0</code>	for every POLL cycle this counter is incremented
<code>test-block-bits-1</code>	every 10 cycles, this counter is bumped once, and <code>test-block-bits-0</code> is set to 0
<code>test-block-bits-2</code>	every 100 cycles, this counter is bumped once, and <code>test-block-bits-1</code> is set to 0
<code>test-block-bits-3</code>	every 1000 cycles, this counter is bumped once, and <code>test-block-bits-2</code> is set to 0

reset-counters	writing ANY thing to this control will reset all the above counters.
----------------	--

Use of the `test_device_edac` driver should enable any others to create their own unique drivers for their hardware systems.

The `test_device_edac` sample driver is located at the <http://bluesmoke.sourceforge.net> project site for EDAC.

### 64.2.19 Usage of EDAC APIs on Nehalem and newer Intel CPUs

On older Intel architectures, the memory controller was part of the North Bridge chipset. Nehalem, Sandy Bridge, Ivy Bridge, Haswell, Sky Lake and newer Intel architectures integrated an enhanced version of the memory controller (MC) inside the CPUs.

This chapter will cover the differences of the enhanced memory controllers found on newer Intel CPUs, such as `i7core_edac`, `sb_edac` and `sbx_edac` drivers.

---

**Note:** The Xeon E7 processor families use a separate chip for the memory controller, called Intel Scalable Memory Buffer. This section doesn't apply for such families.

---

- 1) There is one Memory Controller per Quick Patch Interconnect (QPI). At the driver, the term "socket" means one QPI. This is associated with a physical CPU socket.

Each MC have 3 physical read channels, 3 physical write channels and 3 logic channels. The driver currently sees it as just 3 channels. Each channel can have up to 3 DIMMs.

The minimum known unity is DIMMs. There are no information about csrows. As EDAC API maps the minimum unity is csrows, the driver sequentially maps channel/DIMM into different csrows.

For example, supposing the following layout:

```
Ch0 phy rd0, wr0 (0x063f4031): 2 ranks, UDIMMs
  dimm 0 1024 Mb offset: 0, bank: 8, rank: 1, row: 0x4000, col: 0x400
  dimm 1 1024 Mb offset: 4, bank: 8, rank: 1, row: 0x4000, col: 0x400
Ch1 phy rd1, wr1 (0x063f4031): 2 ranks, UDIMMs
  dimm 0 1024 Mb offset: 0, bank: 8, rank: 1, row: 0x4000, col: 0x400
Ch2 phy rd3, wr3 (0x063f4031): 2 ranks, UDIMMs
  dimm 0 1024 Mb offset: 0, bank: 8, rank: 1, row: 0x4000, col: 0x400
```

The driver will map it as:

```
csrow0: channel 0, dimm0
csrow1: channel 0, dimm1
csrow2: channel 1, dimm0
csrow3: channel 2, dimm0
```

exports one DIMM per csrow.

Each QPI is exported as a different memory controller.

- 2) The MC has the ability to inject errors to test drivers. The drivers implement this functionality via some error injection nodes:

For injecting a memory error, there are some sysfs nodes, under `/sys/devices/system/edac/mc/mc?/`:

- **inject\_addrmatch/\***: Controls the error injection mask register. It is possible to specify several characteristics of the address to match an error code:

```
dimmm = the affected dimm. Numbers are relative to a channel;
rank = the memory rank;
channel = the channel that will generate an error;
bank = the affected bank;
page = the page address;
column (or col) = the address column.
```

each of the above values can be set to “any” to match any valid value.

At driver init, all values are set to any.

For example, to generate an error at rank 1 of dimm 2, for any channel, any bank, any page, any column:

```
echo 2 >/sys/devices/system/edac/mc/mc0/inject_
↪addrmatch/dimm
echo 1 >/sys/devices/system/edac/mc/mc0/inject_
↪addrmatch/rank

To return to the default behaviour of matching any, you can
↪do::

echo any >/sys/devices/system/edac/mc/mc0/inject_
↪addrmatch/dimm
echo any >/sys/devices/system/edac/mc/mc0/inject_
↪addrmatch/rank
```

- **inject\_eccmask**: specifies what bits will have troubles,
- **inject\_section**: specifies what ECC cache section will get the error:

```
3 for both
2 for the highest
1 for the lowest
```

- **inject\_type**: specifies the type of error, being a combination of the following bits:

```
bit 0 - repeat
bit 1 - ecc
bit 2 - parity
```

- **inject\_enable**: starts the error generation when something different than 0 is written.

All inject vars can be read. root permission is needed for write.

Datasheet states that the error will only be generated after a write on an address that matches inject\_addrmatch. It seems, however, that reading will also produce an error.

For example, the following code will generate an error for any write access at socket 0, on any DIMM/address on channel 2:

```
echo 2 >/sys/devices/system/edac/mc/mc0/inject_addrmatch/channel
echo 2 >/sys/devices/system/edac/mc/mc0/inject_type
echo 64 >/sys/devices/system/edac/mc/mc0/inject_eccmask
echo 3 >/sys/devices/system/edac/mc/mc0/inject_section
echo 1 >/sys/devices/system/edac/mc/mc0/inject_enable
dd if=/dev/mem of=/dev/null seek=16k bs=4k count=1 >& /dev/null
```

For socket 1, it is needed to replace “mc0” by “mc1” at the above commands.

The generated error message will look like:

```
EDAC MC0: UE row 0, channel-a= 0 channel-b= 0 labels "-": NON_FATAL_
↳(addr = 0x0075b980, socket=0, Dimm=0, Channel=2,
↳syndrome=0x00000040, count=1, Err=8c0000400001009f:4000080482 (read_
↳error: read ECC error))
```

### 3) Corrected Error memory register counters

Those newer MCs have some registers to count memory errors. The driver uses those registers to report Corrected Errors on devices with Registered DIMMs.

However, those counters don't work with Unregistered DIMM. As the chipset offers some counters that also work with UDIMMs (but with a worse level of granularity than the default ones), the driver exposes those registers for UDIMM memories.

They can be read by looking at the contents of all\_channel\_counts/:

```
$ for i in /sys/devices/system/edac/mc/mc0/all_channel_counts/*; do
↳echo $i; cat $i; done
  /sys/devices/system/edac/mc/mc0/all_channel_counts/udimm0
  0
  /sys/devices/system/edac/mc/mc0/all_channel_counts/udimm1
  0
  /sys/devices/system/edac/mc/mc0/all_channel_counts/udimm2
  0
```

What happens here is that errors on different csrows, but at the same dimm number will increment the same counter. So, in this memory mapping:

```
csrow0: channel 0, dimm0
csrow1: channel 0, dimm1
csrow2: channel 1, dimm0
csrow3: channel 2, dimm0
```

The hardware will increment udimm0 for an error at the first dimm at either csrow0, csrow2 or csrow3;

The hardware will increment `udimm1` for an error at the second dimm at either `csrow0`, `csrow2` or `csrow3`;

The hardware will increment `udimm2` for an error at the third dimm at either `csrow0`, `csrow2` or `csrow3`;

#### 4) Standard error counters

The standard error counters are generated when an `mcelog` error is received by the driver. Since, with UDIMM, this is counted by software, it is possible that some errors could be lost. With RDIMM's, they display the contents of the registers

### 64.2.20 Reference documents used on `amd64_edac`

`amd64_edac` module is based on the following documents (available from <http://support.amd.com/en-us/search/tech-docs>):

- Title** BIOS and Kernel Developer's Guide for AMD Athlon 64 and AMD Opteron Processors

**AMD publication #** 26094

**Revision** 3.26

**Link** <http://support.amd.com/TechDocs/26094.PDF>
- Title** BIOS and Kernel Developer's Guide for AMD NPT Family 0Fh Processors

**AMD publication #** 32559

**Revision** 3.00

**Issue Date** May 2006

**Link** <http://support.amd.com/TechDocs/32559.pdf>
- Title** BIOS and Kernel Developer's Guide (BKDG) For AMD Family 10h Processors

**AMD publication #** 31116

**Revision** 3.00

**Issue Date** September 07, 2007

**Link** <http://support.amd.com/TechDocs/31116.pdf>
- Title** BIOS and Kernel Developer's Guide (BKDG) for AMD Family 15h Models 30h-3Fh Processors

**AMD publication #** 49125

**Revision** 3.06

**Issue Date** 2/12/2015 (latest release)

**Link** [http://support.amd.com/TechDocs/49125\\_15h\\_Models\\_30h-3Fh\\_BKDG.pdf](http://support.amd.com/TechDocs/49125_15h_Models_30h-3Fh_BKDG.pdf)

5.     **Title** BIOS and Kernel Developer' s Guide (BKDG) for AMD Family  
          15h Models 60h-6Fh Processors  
**AMD publication #** 50742  
**Revision** 3.01  
**Issue Date** 7/23/2015 (latest release)  
**Link** [http://support.amd.com/TechDocs/50742\\_15h\\_Models\\_60h-6Fh\\_BKDG.pdf](http://support.amd.com/TechDocs/50742_15h_Models_60h-6Fh_BKDG.pdf)
  
6.     **Title** BIOS and Kernel Developer' s Guide (BKDG) for AMD Family  
          16h Models 00h-0Fh Processors  
**AMD publication #** 48751  
**Revision** 3.03  
**Issue Date** 2/23/2015 (latest release)  
**Link** [http://support.amd.com/TechDocs/48751\\_16h\\_bkdg.pdf](http://support.amd.com/TechDocs/48751_16h_bkdg.pdf)

## Credits

- Written by Doug Thompson <[dougthompson@xmission.com](mailto:dougthompson@xmission.com)>
  - 7 Dec 2005
  - 17 Jul 2007 Updated
- © Mauro Carvalho Chehab
  - 05 Aug 2009 Nehalem interface
  - 26 Oct 2016 Converted to ReST and cleanups at the Nehalem section
- EDAC authors/maintainers:
  - Doug Thompson, Dave Jiang, Dave Peterson et al,
  - Mauro Carvalho Chehab
  - Borislav Petkov
  - original author: Thayne Harbaugh



## **REAL TIME CLOCK (RTC) DRIVERS FOR LINUX**

When Linux developers talk about a “Real Time Clock” , they usually mean something that tracks wall clock time and is battery backed so that it works even with system power off. Such clocks will normally not track the local time zone or daylight savings time - unless they dual boot with MS-Windows - but will instead be set to Coordinated Universal Time (UTC, formerly “Greenwich Mean Time” ).

The newest non-PC hardware tends to just count seconds, like the `time(2)` system call reports, but RTCs also very commonly represent time using the Gregorian calendar and 24 hour time, as reported by `gmtime(3)`.

Linux has two largely-compatible userspace RTC API families you may need to know about:

- `/dev/rtc` ...is the RTC provided by PC compatible systems, so it’ s not very portable to non-x86 systems.
- `/dev/rtc0`, `/dev/rtc1` ...are part of a framework that’ s supported by a wide variety of RTC chips on all systems.

Programmers need to understand that the PC/AT functionality is not always available, and some systems can do much more. That is, the RTCs use the same API to make requests in both RTC frameworks (using different filenames of course), but the hardware may not offer the same functionality. For example, not every RTC is hooked up to an IRQ, so they can’ t all issue alarms; and where standard PC RTCs can only issue an alarm up to 24 hours in the future, other hardware may be able to schedule one any time in the upcoming century.

### **65.1 Old PC/AT-Compatible driver: `/dev/rtc`**

All PCs (even Alpha machines) have a Real Time Clock built into them. Usually they are built into the chipset of the computer, but some may actually have a Motorola MC146818 (or clone) on the board. This is the clock that keeps the date and time while your computer is turned off.

ACPI has standardized that MC146818 functionality, and extended it in a few ways (enabling longer alarm periods, and wake-from-hibernate). That functionality is NOT exposed in the old driver.

However it can also be used to generate signals from a slow 2Hz to a relatively fast 8192Hz, in increments of powers of two. These signals are reported by interrupt number 8. (Oh! So that is what IRQ 8 is for...) It can also function as a 24hr alarm,

raising IRQ 8 when the alarm goes off. The alarm can also be programmed to only check any subset of the three programmable values, meaning that it could be set to ring on the 30th second of the 30th minute of every hour, for example. The clock can also be set to generate an interrupt upon every clock update, thus generating a 1Hz signal.

The interrupts are reported via `/dev/rtc` (major 10, minor 135, read only character device) in the form of an unsigned long. The low byte contains the type of interrupt (update-done, alarm-rang, or periodic) that was raised, and the remaining bytes contain the number of interrupts since the last read. Status information is reported through the pseudo-file `/proc/driver/rtc` if the `/proc` filesystem was enabled. The driver has built in locking so that only one process is allowed to have the `/dev/rtc` interface open at a time.

A user process can monitor these interrupts by doing a `read(2)` or a `select(2)` on `/dev/rtc` - either will block/stop the user process until the next interrupt is received. This is useful for things like reasonably high frequency data acquisition where one doesn't want to burn up 100% CPU by polling `gettimeofday` etc. etc.

At high frequencies, or under high loads, the user process should check the number of interrupts received since the last read to determine if there has been any interrupt "pileup" so to speak. Just for reference, a typical 486-33 running a tight read loop on `/dev/rtc` will start to suffer occasional interrupt pileup (i.e. > 1 IRQ event since last read) for frequencies above 1024Hz. So you really should check the high bytes of the value you read, especially at frequencies above that of the normal timer interrupt, which is 100Hz.

Programming and/or enabling interrupt frequencies greater than 64Hz is only allowed by root. This is perhaps a bit conservative, but we don't want an evil user generating lots of IRQs on a slow 386sx-16, where it might have a negative impact on performance. This 64Hz limit can be changed by writing a different value to `/proc/sys/dev/rtc/max-user-freq`. Note that the interrupt handler is only a few lines of code to minimize any possibility of this effect.

Also, if the kernel time is synchronized with an external source, the kernel will write the time back to the CMOS clock every 11 minutes. In the process of doing this, the kernel briefly turns off RTC periodic interrupts, so be aware of this if you are doing serious work. If you don't synchronize the kernel time with an external source (via `ntp` or whatever) then the kernel will keep its hands off the RTC, allowing you exclusive access to the device for your applications.

The alarm and/or interrupt frequency are programmed into the RTC via various `ioctl(2)` calls as listed in `./include/linux/rtc.h`. Rather than write 50 pages describing the `ioctl()` and so on, it is perhaps more useful to include a small test program that demonstrates how to use them, and demonstrates the features of the driver. This is probably a lot more useful to people interested in writing applications that will be using this driver. See the code at the end of this document.

(The original `/dev/rtc` driver was written by Paul Gortmaker.)

## 65.2 New portable “RTC Class” drivers: /dev/rtcN

Because Linux supports many non-ACPI and non-PC platforms, some of which have more than one RTC style clock, it needed a more portable solution than expecting a single battery-backed MC146818 clone on every system. Accordingly, a new “RTC Class” framework has been defined. It offers three different userspace interfaces:

- /dev/rtcN …much the same as the older /dev/rtc interface
- /sys/class/rtc/rtcN …sysfs attributes support readonly access to some RTC attributes.
- /proc/driver/rtc …the system clock RTC may expose itself using a procfs interface. If there is no RTC for the system clock, rtc0 is used by default. More information is (currently) shown here than through sysfs.

The RTC Class framework supports a wide variety of RTCs, ranging from those integrated into embeddable system-on-chip (SOC) processors to discrete chips using I2C, SPI, or some other bus to communicate with the host CPU. There’s even support for PC-style RTCs …including the features exposed on newer PCs through ACPI.

The new framework also removes the “one RTC per system” restriction. For example, maybe the low-power battery-backed RTC is a discrete I2C chip, but a high functionality RTC is integrated into the SOC. That system might read the system clock from the discrete RTC, but use the integrated one for all other tasks, because of its greater functionality.

Check out `tools/testing/selftests/rtc/rtctest.c` for an example usage of the `ioctl` interface.



## **LINUX SERIAL CONSOLE**

To use a serial port as console you need to compile the support into your kernel - by default it is not compiled in. For PC style serial ports it's the config option next to menu option:

Character devices → Serial drivers → 8250/16550 and compatible serial support  
→ Console on 8250/16550 and compatible serial port

You must compile serial support into the kernel and not as a module.

It is possible to specify multiple devices for console output. You can define a new kernel command line option to select which device(s) to use for console output.

The format of this option is:

```
console=device,options
```

```
device:      tty0 for the foreground virtual console
              ttyX for any other virtual console
              ttySx for a serial port
              lp0 for the first parallel port
              ttyUSB0 for the first USB serial device
```

```
options:     depend on the driver. For the serial port this
              defines the baudrate/parity/bits/flow control of
              the port, in the format BBBBPNF, where BBBB is the
              speed, P is parity (n/o/e), N is number of bits,
              and F is flow control ('r' for RTS). Default is
              9600n8. The maximum baudrate is 115200.
```

You can specify multiple console= options on the kernel command line. Output will appear on all of them. The last device will be used when you open /dev/console. So, for example:

```
console=ttyS1,9600 console=tty0
```

defines that opening /dev/console will get you the current foreground virtual console, and kernel messages will appear on both the VGA console and the 2nd serial port (ttyS1 or COM2) at 9600 baud.

Note that you can only define one console per device type (serial, video).

If no console device is specified, the first device found capable of acting as a system console will be used. At this time, the system first looks for a VGA card and then for a serial port. So if you don't have a VGA card in your system the first serial port will automatically become the console.

You will need to create a new device to use `/dev/console`. The official `/dev/console` is now character device 5,1.

(You can also use a network device as a console. See `Documentation/networking/netconsole.rst` for information on that.)

Here's an example that will use `/dev/ttyS1` (COM2) as the console. Replace the sample values as needed.

1. Create `/dev/console` (real console) and `/dev/tty0` (master virtual console):

```
cd /dev
rm -f console tty0
mknod -m 622 console c 5 1
mknod -m 622 tty0 c 4 0
```

2. LILO can also take input from a serial device. This is a very useful option. To tell LILO to use the serial port: In `lilo.conf` (global section):

```
serial = 1,9600n8 (ttyS1, 9600 bd, no parity, 8 bits)
```

3. Adjust to kernel flags for the new kernel, again in `lilo.conf` (kernel section):

```
append = "console=ttyS1,9600"
```

4. Make sure a `getty` runs on the serial port so that you can login to it once the system is done booting. This is done by adding a line like this to `/etc/inittab` (exact syntax depends on your `getty`):

```
S1:23:respawn:/sbin/getty -L ttyS1 9600 vt100
```

5. `Init` and `/etc/ioctl.save`

`Sysvinit` remembers its `stty` settings in a file in `/etc`, called `/etc/ioctl.save`. REMOVE THIS FILE before using the serial console for the first time, because otherwise `init` will probably set the baudrate to 38400 (baudrate of the virtual console).

6. `/dev/console` and X Programs that want to do something with the virtual console usually open `/dev/console`. If you have created the new `/dev/console` device, and your console is NOT the virtual console some programs will fail. Those are programs that want to access the VT interface, and use `/dev/console` instead of `/dev/tty0`. Some of those programs are:

```
Xfree86, svgalib, gpm, SVGATextMode
```

It should be fixed in modern versions of these programs though.

Note that if you boot without a `console=` option (or with `console=/dev/tty0`), `/dev/console` is the same as `/dev/tty0`. In that case everything will still work.

7. Thanks

Thanks to Geert Uytterhoeven <[geert@linux-m68k.org](mailto:geert@linux-m68k.org)> for porting the patches from 2.1.4x to 2.1.6x for taking care of the integration of these patches into m68k, ppc and alpha.

Miquel van Smoorenburg <[miquels@cistron.nl](mailto:miquels@cistron.nl)>, 11-Jun-2000



## **VIDEO MODE SELECTION SUPPORT 2.13**

**Copyright** © 1995–1999 Martin Mares, <mj@ucw.cz>

### **67.1 Intro**

This small document describes the “Video Mode Selection” feature which allows the use of various special video modes supported by the video BIOS. Due to usage of the BIOS, the selection is limited to boot time (before the kernel decompression starts) and works only on 80X86 machines.

---

**Note:** Short intro for the impatient: Just use `vga=ask` for the first time, enter scan on the video mode prompt, pick the mode you want to use, remember its mode ID (the four-digit hexadecimal number) and then set the `vga` parameter to this number (converted to decimal first).

---

The video mode to be used is selected by a kernel parameter which can be specified in the kernel Makefile (the `SVGA_MODE=...` line) or by the “`vga=...`” option of LILO (or some other boot loader you use) or by the “`vidmode`” utility (present in standard Linux utility packages). You can use the following values of this parameter:

<p><code>NORMAL_VGA</code> - Standard 80x25 mode available on all display adapters.</p> <p><code>EXTENDED_VGA</code> - Standard 8-pixel font mode: 80x43 on EGA, 80x50 on VGA.</p> <p><code>ASK_VGA</code> - Display a video mode menu upon startup (see below).</p> <p><code>0..35</code> - Menu item number (when you have used the menu to view the list of modes available on your adapter, you can specify the menu item you want to use). <code>0..9</code> correspond to “0”..“9”, <code>10..35</code> to “a”..“z”. Warning: the mode list displayed may vary as the kernel version changes, because the modes are listed in a “first detected -- first displayed” manner. It's better to use absolute mode numbers instead.</p> <p><code>0x....</code> - Hexadecimal video mode ID (also displayed on the menu, see below for exact meaning of the ID). Warning: <code>rdev</code> and LILO don't support hexadecimal numbers -- you have to convert it to decimal manually.</p>
--

## 67.2 Menu

The `ASK_VGA` mode causes the kernel to offer a video mode menu upon bootup. It displays a “Press <RETURN> to see video modes available, <SPACE> to continue or wait 30 secs” message. If you press <RETURN>, you enter the menu, if you press <SPACE> or wait 30 seconds, the kernel will boot up in the standard 80x25 mode.

The menu looks like:

```
Video adapter: <name-of-detected-video-adapter>
Mode:         COLSxROWS:
0  0F00  80x25
1  0F01  80x50
2  0F02  80x43
3  0F03  80x26
....
Enter mode number or ``scan``: <flashing-cursor-here>
```

<name-of-detected-video-adapter> tells what video adapter did Linux detect - it's either a generic adapter name (MDA, CGA, HGC, EGA, VGA, VESA VGA [a VGA with VESA-compliant BIOS]) or a chipset name (e.g., Trident). Direct detection of chipsets is turned off by default as it's inherently unreliable due to absolutely insane PC design.

“0 0F00 80x25” means that the first menu item (the menu items are numbered from “0” to “9” and from “a” to “z” ) is a 80x25 mode with ID=0x0f00 (see the next section for a description of mode IDs).

<flashing-cursor-here> encourages you to enter the item number or mode ID you wish to set and press <RETURN>. If the computer complains something about “Unknown mode ID”, it is trying to tell you that it isn't possible to set such a mode. It's also possible to press only <RETURN> which leaves the current mode.

The mode list usually contains a few basic modes and some VESA modes. In case your chipset has been detected, some chipset-specific modes are shown as well (some of these might be missing or unusable on your machine as different BIOSes are often shipped with the same card and the mode numbers depend purely on the VGA BIOS).

The modes displayed on the menu are partially sorted: The list starts with the standard modes (80x25 and 80x50) followed by “special” modes (80x28 and 80x43), local modes (if the local modes feature is enabled), VESA modes and finally SVGA modes for the auto-detected adapter.

If you are not happy with the mode list offered (e.g., if you think your card is able to do more), you can enter “scan” instead of item number / mode ID. The program will try to ask the BIOS for all possible video mode numbers and test what happens then. The screen will be probably flashing wildly for some time and strange noises will be heard from inside the monitor and so on and then, really all consistent video modes supported by your BIOS will appear (plus maybe some ghost modes). If you are afraid this could damage your monitor, don't use this function.

After scanning, the mode ordering is a bit different: the auto-detected SVGA modes are not listed at all and the modes revealed by scan are shown before all VESA

modes.

## 67.3 Mode IDs

Because of the complexity of all the video stuff, the video mode IDs used here are also a bit complex. A video mode ID is a 16-bit number usually expressed in a hexadecimal notation (starting with "0x"). You can set a mode by entering its mode directly if you know it even if it isn't shown on the menu.

The ID numbers can be divided to those regions:

```
0x0000 to 0x00ff - menu item references. 0x0000 is the first item. Don't
→use
    outside the menu as this can change from boot to boot (especially if
→you
    have used the ``scan`` feature).

0x0100 to 0x017f - standard BIOS modes. The ID is a BIOS video mode number
    (as presented to INT 10, function 00) increased by 0x0100.

0x0200 to 0x08ff - VESA BIOS modes. The ID is a VESA mode ID increased by
    0x0100. All VESA modes should be autodetected and shown on the menu.

0x0900 to 0x09ff - Video7 special modes. Set by calling INT 0x10,
→AX=0x6f05.
    (Usually 940=80x43, 941=132x25, 942=132x44, 943=80x60, 944=100x60,
    945=132x28 for the standard Video7 BIOS)

0x0f00 to 0x0fff - special modes (they are set by various tricks -- usually
    by modifying one of the standard modes). Currently available:
    0x0f00 standard 80x25, don't reset mode if already set (=FFFF)
    0x0f01 standard with 8-point font: 80x43 on EGA, 80x50 on VGA
    0x0f02 VGA 80x43 (VGA switched to 350 scanlines with a 8-point font)
    0x0f03 VGA 80x28 (standard VGA scans, but 14-point font)
    0x0f04 leave current video mode
    0x0f05 VGA 80x30 (480 scans, 16-point font)
    0x0f06 VGA 80x34 (480 scans, 14-point font)
    0x0f07 VGA 80x60 (480 scans, 8-point font)
    0x0f08 Graphics hack (see the VIDEO_GFX_HACK paragraph below)

0x1000 to 0x7fff - modes specified by resolution. The code has a "0xRRCC"
    form where RR is a number of rows and CC is a number of columns.
    E.g., 0x1950 corresponds to a 80x25 mode, 0x2b84 to 132x43 etc.
    This is the only fully portable way to refer to a non-standard mode,
    but it relies on the mode being found and displayed on the menu
    (remember that mode scanning is not done automatically).

0xff00 to 0xffff - aliases for backward compatibility:
    0xffff equivalent to 0x0f00 (standard 80x25)
    0xfffe equivalent to 0x0f01 (EGA 80x43 or VGA 80x50)
```

If you add 0x8000 to the mode ID, the program will try to recalculate vertical display timing according to mode parameters, which can be used to eliminate some annoying bugs of certain VGA BIOSes (usually those used for cards with S3 chipsets and old Cirrus Logic BIOSes) - mainly extra lines at the end of the

display.

### 67.4 Options

Build options for arch/x86/boot/\* are selected by the kernel kconfig utility and the kernel .config file.

VIDEO\_GFX\_HACK - includes special hack for setting of graphics modes to be used later by special drivers. Allows to set *\_any\_* BIOS mode including graphic ones and forcing specific text screen resolution instead of peeking it from BIOS variables. Don't use unless you think you know what you're doing. To activate this setup, use mode number 0x0f08 (see the Mode IDs section above).

### 67.5 Still doesn't work?

When the mode detection doesn't work (e.g., the mode list is incorrect or the machine hangs instead of displaying the menu), try to switch off some of the configuration options listed under "Options". If it fails, you can still use your kernel with the video mode set directly via the kernel parameter.

In either case, please send me a bug report containing what *\_exactly\_* happens and how do the configuration switches affect the behaviour of the bug.

If you start Linux from M\$-DOS, you might also use some DOS tools for video mode setting. In this case, you must specify the 0x0f04 mode ( "leave current settings" ) to Linux, because if you don't and you use any non-standard mode, Linux will switch to 80x25 automatically.

If you set some extended mode and there's one or more extra lines on the bottom of the display containing already scrolled-out text, your VGA BIOS contains the most common video BIOS bug called "incorrect vertical display end setting". Adding 0x8000 to the mode ID might fix the problem. Unfortunately, this must be done manually - no autodetection mechanisms are available.



## 67.6 History

1.0 (??-Nov-95)	First version supporting all adapters supported by the old setup.S + Cirrus Logic 54XX. Present in some 1.3.4? kernels and then removed due to instability on some machines.
2.0 (28-Jan-96)	Rewritten from scratch. Cirrus Logic 64XX support added, almost everything is configurable, the VESA support should be much more stable, explicit mode numbering allowed, "scan" implemented etc.
2.1 (30-Jan-96)	VESA modes moved to 0x200-0x3ff. Mode selection by resolution supported. Few bugs fixed. VESA modes are listed prior to modes supplied by SVGA autodetection as they are more reliable. CLGD autodetect works better. Doesn't depend on 80x25 being active when started. Scanning fixed. 80x43 (any VGA) added. Code cleaned up.
2.2 (01-Feb-96)	EGA 80x43 fixed. VESA extended to 0x200-0x4ff (non-standard 02XX VESA modes work now). Display end bug workaround supported. Special modes renumbered to allow adding of the "recalculate" flag, 0xffff and 0xfffe became aliases instead of real IDs. Screen contents retained during mode changes.
2.3 (15-Mar-96)	Changed to work with 1.3.74 kernel.
2.4 (18-Mar-96)	Added patches by Hans Lermen fixing a memory overwrite problem with some boot loaders. Memory management rewritten to reflect these changes. Unfortunately, screen contents retaining works only with some loaders now. Added a Tseng 132x60 mode.
2.5 (19-Mar-96)	Fixed a VESA mode scanning bug introduced in 2.4.
2.6 (25-Mar-96)	Some VESA BIOS errors not reported - it fixes error reports on several cards with broken VESA code (e.g., ATI VGA).
2.7 (09-Apr-96)	<ul style="list-style-type: none"> <li>Accepted all VESA modes in range 0x100 to 0x7ff, because some cards use very strange mode numbers.</li> <li>Added Realtek VGA modes</li> </ul>
<b>1408</b>	<p><b>Chapter 67. Video Mode Selection Support 2.13</b></p> <ul style="list-style-type: none"> <li>Hardware testing order slightly changed, tests based on ROM contents done as first.</li> </ul>

## **LINUX MAGIC SYSTEM REQUEST KEY HACKS**

Documentation for sysrq.c

### **68.1 What is the magic SysRq key?**

It is a ‘magical’ key combo you can hit which the kernel will respond to regardless of whatever else it is doing, unless it is completely locked up.

### **68.2 How do I enable the magic SysRq key?**

You need to say “yes” to ‘Magic SysRq key (CONFIG\_MAGIC\_SYSRQ)’ when configuring the kernel. When running a kernel with SysRq compiled in, `/proc/sys/kernel/sysrq` controls the functions allowed to be invoked via the SysRq key. The default value in this file is set by the `CONFIG_MAGIC_SYSRQ_DEFAULT_ENABLE` config symbol, which itself defaults to 1. Here is the list of possible values in `/proc/sys/kernel/sysrq`:

- 0 - disable sysrq completely
- 1 - enable all functions of sysrq
- >1 - bitmask of allowed sysrq functions (see below for detailed function description):

2 = 0x2 - enable control of console logging level
4 = 0x4 - enable control of keyboard (SAK, unraw)
8 = 0x8 - enable debugging dumps of processes etc.
16 = 0x10 - enable sync command
32 = 0x20 - enable remount read-only
64 = 0x40 - enable signalling of processes (term, kill, oom-kill)
128 = 0x80 - allow reboot/poweroff
256 = 0x100 - allow nicing of all RT tasks

You can set the value in the file by the following command:

```
echo "number" >/proc/sys/kernel/sysrq
```

The number may be written here either as decimal or as hexadecimal with the 0x prefix. `CONFIG_MAGIC_SYSRQ_DEFAULT_ENABLE` must always be written in hexadecimal.

Note that the value of `/proc/sys/kernel/sysrq` influences only the invocation via a keyboard. Invocation of any operation via `/proc/sysrq-trigger` is always allowed (by a user with admin privileges).

### 68.3 How do I use the magic SysRq key?

**On x86** You press the key combo ALT-SysRq-<command key>.

---

**Note:** Some keyboards may not have a key labeled 'SysRq'. The 'SysRq' key is also known as the 'Print Screen' key. Also some keyboards cannot handle so many keys being pressed at the same time, so you might have better luck with press Alt, press SysRq, release SysRq, press <command key>, release everything.

---

**On SPARC** You press ALT-STOP-<command key>, I believe.

**On the serial console (PC style standard serial ports only)** You send a BREAK, then within 5 seconds a command key. Sending BREAK twice is interpreted as a normal BREAK.

**On PowerPC** Press ALT - Print Screen (or F13) - <command key>. Print Screen (or F13) - <command key> may suffice.

**On other** If you know of the key combos for other architectures, please let me know so I can add them to this section.

**On all** Write a character to `/proc/sysrq-trigger`. e.g.:

```
echo t > /proc/sysrq-trigger
```

## 68.4 What are the ‘command’ keys?

Command	Function
b	Will immediately reboot the system without syncing or unmounting your disks.
c	Will perform a system crash by a NULL pointer dereference. A crashdump will be taken if configured.
d	Shows all locks that are held.
e	Send a SIGTERM to all processes, except for init.
f	Will call the oom killer to kill a memory hog process, but do not panic if nothing can be killed.
g	Used by kgdb (kernel debugger)
h	Will display help (actually any other key than those listed here will display help. but h is easy to remember :-)
i	Send a SIGKILL to all processes, except for init.
j	Forcibly “Just thaw it” - filesystems frozen by the FIFREEZE ioctl.
k	Secure Access Key (SAK) Kills all programs on the current virtual console. NOTE: See important comments below in SAK section.
l	Shows a stack backtrace for all active CPUs.
m	Will dump current memory info to your console.
n	Used to make RT tasks nice-able
o	Will shut your system off (if configured and supported).
p	Will dump the current registers and flags to your console.
q	Will dump per CPU lists of all armed hrtimers (but NOT regular timer_list timers) and detailed information about all clockevent devices.
r	Turns off keyboard raw mode and sets it to XLATE.
s	Will attempt to sync all mounted filesystems.
t	Will dump a list of current tasks and their information to your console.
u	Will attempt to remount all mounted filesystems read-only.
v	Forcefully restores framebuffer console
v	Causes ETM buffer dump [ARM-specific]
w	Dumps tasks that are in uninterruptable (blocked) state.
x	Used by xmon interface on ppc/powerpc platforms. Show global PMU Registers on sparc64. Dump all TLB entries on MIPS.
y	Show global CPU Registers [SPARC-64 specific]
z	Dump the ftrace buffer
0-9	Sets the console log level, controlling which kernel messages will be printed to your console. (0, for example would make it so that only emergency messages like PANICs or OOPSes would make it to your console.)

## 68.5 Okay, so what can I use them for?

Well, `unraw(r)` is very handy when your X server or a `svgalib` program crashes.

`sak(k)` (Secure Access Key) is useful when you want to be sure there is no trojan program running at console which could grab your password when you would try to login. It will kill all programs on given console, thus letting you make sure that the login prompt you see is actually the one from `init`, not some trojan program.

---

**Important:** In its true form it is not a true SAK like the one in a c2 compliant system, and it should not be mistaken as such.

---

It seems others find it useful as (System Attention Key) which is useful when you want to exit a program that will not let you switch consoles. (For example, X or a `svgalib` program.)

`reboot(b)` is good when you're unable to shut down, it is an equivalent of pressing the "reset" button.

`crash(c)` can be used to manually trigger a `crashdump` when the system is hung. Note that this just triggers a crash if there is no dump mechanism available.

`sync(s)` is handy before yanking removable medium or after using a rescue shell that provides no graceful shutdown - it will ensure your data is safely written to the disk. Note that the `sync` hasn't taken place until you see the "OK" and "Done" appear on the screen.

`umount(u)` can be used to mark filesystems as properly unmounted. From the running system's point of view, they will be remounted read-only. The remount isn't complete until you see the "OK" and "Done" message appear on the screen.

The `loglevels 0-9` are useful when your console is being flooded with kernel messages you do not want to see. Selecting `0` will prevent all but the most urgent kernel messages from reaching your console. (They will still be logged if `syslogd/klogd` are alive, though.)

`term(e)` and `kill(i)` are useful if you have some sort of runaway process you are unable to kill any other way, especially if it's spawning other processes.

"just thaw it(j)" is useful if your system becomes unresponsive due to a frozen (probably `root`) filesystem via the `FIFREEZE` ioctl.

## 68.6 Sometimes SysRq seems to get 'stuck' after using it, what can I do?

That happens to me, also. I've found that tapping `shift`, `alt`, and `control` on both sides of the keyboard, and hitting an invalid `sysrq` sequence again will fix the problem. (i.e., something like `alt-sysrq-z`). Switching to another virtual console (`ALT+Fn`) and then back again should also help.

## 68.7 I hit SysRq, but nothing seems to happen, what's wrong?

There are some keyboards that produce a different keycode for SysRq than the pre-defined value of 99 (see `KEY_SYSRQ` in `include/uapi/linux/input-event-codes.h`), or which don't have a SysRq key at all. In these cases, run `showkey -s` to find an appropriate scancode sequence, and use `setkeycodes <sequence> 99` to map this sequence to the usual SysRq code (e.g., `setkeycodes e05b 99`). It's probably best to put this command in a boot script. Oh, and by the way, you exit `showkey` by not typing anything for ten seconds.

## 68.8 I want to add SysRQ key events to a module, how does it work?

In order to register a basic function with the table, you must first include the header `include/linux/sysrq.h`, this will define everything else you need. Next, you must create a `sysrq_key_op` struct, and populate it with A) the key handler function you will use, B) a `help_msg` string, that will print when SysRQ prints help, and C) an `action_msg` string, that will print right before your handler is called. Your handler must conform to the prototype in `'sysrq.h'`.

After the `sysrq_key_op` is created, you can call the kernel function `register_sysrq_key(int key, const struct sysrq_key_op *op_p)`; this will register the operation pointed to by `op_p` at table key `'key'`, if that slot in the table is blank. At module unload time, you must call the function `unregister_sysrq_key(int key, const struct sysrq_key_op *op_p)`, which will remove the key `op` pointed to by `'op_p'` from the key `'key'`, if and only if it is currently registered in that slot. This is in case the slot has been overwritten since you registered it.

The Magic SysRQ system works by registering key operations against a key op lookup table, which is defined in `'drivers/tty/sysrq.c'`. This key table has a number of operations registered into it at compile time, but is mutable, and 2 functions are exported for interface to it:

<code>register_sysrq_key</code> and <code>unregister_sysrq_key</code> .
---

Of course, never ever leave an invalid pointer in the table. I.e., when your module that called `register_sysrq_key()` exits, it must call `unregister_sysrq_key()` to clean up the `sysrq` key table entry that it used. Null pointers in the table are always safe. :)

If for some reason you feel the need to call the `handle_sysrq` function from within a function called by `handle_sysrq`, you must be aware that you are in a lock (you are also in an interrupt handler, which means don't sleep!), so you must call `__handle_sysrq_nolock` instead.

## 68.9 When I hit a SysRq key combination only the header appears on the console?

Sysrq output is subject to the same console loglevel control as all other console output. This means that if the kernel was booted 'quiet' as is common on distro kernels the output may not appear on the actual console, even though it will appear in the dmesg buffer, and be accessible via the dmesg command and to the consumers of /proc/kmsg. As a specific exception the header line from the sysrq command is passed to all console consumers as if the current loglevel was maximum. If only the header is emitted it is almost certain that the kernel loglevel is too low. Should you require the output on the console channel then you will need to temporarily up the console loglevel using alt-sysrq-8 or:

```
echo 8 > /proc/sysrq-trigger
```

Remember to return the loglevel to normal after triggering the sysrq command you are interested in.

## 68.10 I have more questions, who can I ask?

**Just ask them on the linux-kernel mailing list:** [linux-kernel@vger.kernel.org](mailto:linux-kernel@vger.kernel.org)

## 68.11 Credits

- Written by Mydraal <[vulpyne@vulpyne.net](mailto:vulpyne@vulpyne.net)>
- Updated by Adam Sulmicki <[adam@cfar.umd.edu](mailto:adam@cfar.umd.edu)>
- Updated by Jeremy M. Dolan <[jmd@turbogeek.org](mailto:jmd@turbogeek.org)> 2001/01/28 10:15:59
- Added to by Crutcher Dunnavant <[crutcher+kernel@datastacks.com](mailto:crutcher+kernel@datastacks.com)>

## **USB4 AND THUNDERBOLT**

USB4 is the public specification based on Thunderbolt 3 protocol with some differences at the register level among other things. Connection manager is an entity running on the host router (host controller) responsible for enumerating routers and establishing tunnels. A connection manager can be implemented either in firmware or software. Typically PCs come with a firmware connection manager for Thunderbolt 3 and early USB4 capable systems. Apple systems on the other hand use software connection manager and the later USB4 compliant devices follow the suit.

The Linux Thunderbolt driver supports both and can detect at runtime which connection manager implementation is to be used. To be on the safe side the software connection manager in Linux also advertises security level user which means PCIe tunneling is disabled by default. The documentation below applies to both implementations with the exception that the software connection manager only supports user security level and is expected to be accompanied with an IOMMU based DMA protection.

### **69.1 Security levels and how to use them**

The interface presented here is not meant for end users. Instead there should be a userspace tool that handles all the low-level details, keeps a database of the authorized devices and prompts users for new connections.

More details about the sysfs interface for Thunderbolt devices can be found in [Documentation/ABI/testing/sysfs-bus-thunderbolt](#).

Those users who just want to connect any device without any sort of manual work can add following line to `/etc/udev/rules.d/99-local.rules`:

```
ACTION=="add", SUBSYSTEM=="thunderbolt", ATTR{authorized}=="0", ATTR  
->{authorized}="1"
```

This will authorize all devices automatically when they appear. However, keep in mind that this bypasses the security levels and makes the system vulnerable to DMA attacks.

Starting with Intel Falcon Ridge Thunderbolt controller there are 4 security levels available. Intel Titan Ridge added one more security level (usonly). The reason for these is the fact that the connected devices can be DMA masters and thus read contents of the host memory without CPU and OS knowing about it. There are

ways to prevent this by setting up an IOMMU but it is not always available for various reasons.

The security levels are as follows:

- none** All devices are automatically connected by the firmware. No user approval is needed. In BIOS settings this is typically called Legacy mode.
- user** User is asked whether the device is allowed to be connected. Based on the device identification information available through `/sys/bus/thunderbolt/devices`, the user then can make the decision. In BIOS settings this is typically called Unique ID.
- secure** User is asked whether the device is allowed to be connected. In addition to UUID the device (if it supports secure connect) is sent a challenge that should match the expected one based on a random key written to the key sysfs attribute. In BIOS settings this is typically called One time saved key.
- dponly** The firmware automatically creates tunnels for Display Port and USB. No PCIe tunneling is done. In BIOS settings this is typically called Display Port Only.
- usbonly** The firmware automatically creates tunnels for the USB controller and Display Port in a dock. All PCIe links downstream of the dock are removed.

The current security level can be read from `/sys/bus/thunderbolt/devices/domainX/security` where `domainX` is the Thunderbolt domain the host controller manages. There is typically one domain per Thunderbolt host controller.

If the security level reads as `user` or `secure` the connected device must be authorized by the user before PCIe tunnels are created (e.g the PCIe device appears).

Each Thunderbolt device plugged in will appear in sysfs under `/sys/bus/thunderbolt/devices`. The device directory carries information that can be used to identify the particular device, including its name and UUID.

## 69.2 Authorizing devices when security level is user or secure

When a device is plugged in it will appear in sysfs as follows:

```
/sys/bus/thunderbolt/devices/0-1/authorized - 0
/sys/bus/thunderbolt/devices/0-1/device    - 0x8004
/sys/bus/thunderbolt/devices/0-1/device_name - Thunderbolt to FireWire_
↳Adapter
/sys/bus/thunderbolt/devices/0-1/vendor    - 0x1
/sys/bus/thunderbolt/devices/0-1/vendor_name - Apple, Inc.
/sys/bus/thunderbolt/devices/0-1/unique_id - e0376f00-0300-0100-ffff-
↳ffffffffffff
```

The `authorized` attribute reads 0 which means no PCIe tunnels are created yet. The user can authorize the device by simply entering:

```
# echo 1 > /sys/bus/thunderbolt/devices/0-1/authorized
```

This will create the PCIe tunnels and the device is now connected.

If the device supports secure connect, and the domain security level is set to secure, it has an additional attribute key which can hold a random 32-byte value used for authorization and challenging the device in future connects:

```
/sys/bus/thunderbolt/devices/0-3/authorized - 0
/sys/bus/thunderbolt/devices/0-3/device - 0x305
/sys/bus/thunderbolt/devices/0-3/device_name - AKiTi0 Thunder3 PCIe Box
/sys/bus/thunderbolt/devices/0-3/key -
/sys/bus/thunderbolt/devices/0-3/vendor - 0x41
/sys/bus/thunderbolt/devices/0-3/vendor_name - inXtron
/sys/bus/thunderbolt/devices/0-3/unique_id - dc010000-0000-8508-a22d-
↪32ca6421cb16
```

Notice the key is empty by default.

If the user does not want to use secure connect they can just echo 1 to the authorized attribute and the PCIe tunnels will be created in the same way as in the user security level.

If the user wants to use secure connect, the first time the device is plugged a key needs to be created and sent to the device:

```
# key=$(openssl rand -hex 32)
# echo $key > /sys/bus/thunderbolt/devices/0-3/key
# echo 1 > /sys/bus/thunderbolt/devices/0-3/authorized
```

Now the device is connected (PCIe tunnels are created) and in addition the key is stored on the device NVM.

Next time the device is plugged in the user can verify (challenge) the device using the same key:

```
# echo $key > /sys/bus/thunderbolt/devices/0-3/key
# echo 2 > /sys/bus/thunderbolt/devices/0-3/authorized
```

If the challenge the device returns back matches the one we expect based on the key, the device is connected and the PCIe tunnels are created. However, if the challenge fails no tunnels are created and error is returned to the user.

If the user still wants to connect the device they can either approve the device without a key or write a new key and write 1 to the authorized file to get the new key stored on the device NVM.

## 69.3 DMA protection utilizing IOMMU

Recent systems from 2018 and forward with Thunderbolt ports may natively support IOMMU. This means that Thunderbolt security is handled by an IOMMU so connected devices cannot access memory regions outside of what is allocated for them by drivers. When Linux is running on such system it automatically enables IOMMU if not enabled by the user already. These systems can be identified by reading 1 from `/sys/bus/thunderbolt/devices/domainX/iommu_dma_protection` attribute.

The driver does not do anything special in this case but because DMA protection is handled by the IOMMU, security levels (if set) are redundant. For this reason some systems ship with security level set to none. Other systems have security level set to user in order to support downgrade to older OS, so users who want to automatically authorize devices when IOMMU DMA protection is enabled can use the following udev rule:

```
ACTION=="add", SUBSYSTEM=="thunderbolt", ATTRS{iommu_dma_protection}=="1",  
↳ATTR{authorized}=="0", ATTR{authorized}="1"
```

## 69.4 Upgrading NVM on Thunderbolt device or host

Since most of the functionality is handled in firmware running on a host controller or a device, it is important that the firmware can be upgraded to the latest where possible bugs in it have been fixed. Typically OEMs provide this firmware from their support site.

There is also a central site which has links where to download firmware for some machines:

### [Thunderbolt Updates](#)

Before you upgrade firmware on a device or host, please make sure it is a suitable upgrade. Failing to do that may render the device (or host) in a state where it cannot be used properly anymore without special tools!

Host NVM upgrade on Apple Macs is not supported.

Once the NVM image has been downloaded, you need to plug in a Thunderbolt device so that the host controller appears. It does not matter which device is connected (unless you are upgrading NVM on a device - then you need to connect that particular device).

Note an OEM-specific method to power the controller up ( “force power” ) may be available for your system in which case there is no need to plug in a Thunderbolt device.

After that we can write the firmware to the non-active parts of the NVM of the host or device. As an example here is how Intel NUC6i7KYK (Skull Canyon) Thunderbolt controller NVM is upgraded:

```
# dd if=KYK_TBT_FW_0018.bin of=/sys/bus/thunderbolt/devices/0-0/nvm_non_  
↳active0/nvmem
```

Once the operation completes we can trigger NVM authentication and upgrade process as follows:

```
# echo 1 > /sys/bus/thunderbolt/devices/0-0/nvm_authenticate
```

If no errors are returned, the host controller shortly disappears. Once it comes back the driver notices it and initiates a full power cycle. After a while the host controller appears again and this time it should be fully functional.

We can verify that the new NVM firmware is active by running the following commands:

```
# cat /sys/bus/thunderbolt/devices/0-0/nvm_authenticate
0x0
# cat /sys/bus/thunderbolt/devices/0-0/nvm_version
18.0
```

If `nvm_authenticate` contains anything other than `0x0` it is the error code from the last authentication cycle, which means the authentication of the NVM image failed.

Note names of the NVMe devices `nvm_activeN` and `nvm_non_activeN` depend on the order they are registered in the NVMe subsystem. N in the name is the identifier added by the NVMe subsystem.

## 69.5 Upgrading NVM when host controller is in safe mode

If the existing NVM is not properly authenticated (or is missing) the host controller goes into safe mode which means that the only available functionality is flashing a new NVM image. When in this mode, reading `nvm_version` fails with `ENODATA` and the device identification information is missing.

To recover from this mode, one needs to flash a valid NVM image to the host controller in the same way it is done in the previous chapter.

## 69.6 Networking over Thunderbolt cable

Thunderbolt technology allows software communication between two hosts connected by a Thunderbolt cable.

It is possible to tunnel any kind of traffic over a Thunderbolt link but currently we only support Apple ThunderboltIP protocol.

If the other host is running Windows or macOS, the only thing you need to do is to connect a Thunderbolt cable between the two hosts; the `thunderbolt-net` driver is loaded automatically. If the other host is also Linux you should load `thunderbolt-net` manually on one host (it does not matter which one):

```
# modprobe thunderbolt-net
```

This triggers module load on the other host automatically. If the driver is built-in to the kernel image, there is no need to do anything.

The driver will create one virtual ethernet interface per Thunderbolt port which are named like `thunderbolt0` and so on. From this point you can either use standard userspace tools like `ifconfig` to configure the interface or let your GUI handle it automatically.

### 69.7 Forcing power

Many OEMs include a method that can be used to force the power of a Thunderbolt controller to an “On” state even if nothing is connected. If supported by your machine this will be exposed by the WMI bus with a sysfs attribute called “`force_power`” .

**For example the intel-wmi-thunderbolt driver exposes this attribute in:**

```
/sys/bus/wmi/devices/86CCFD48-205E-4A77-9C48-2021CBEDE341/force_power
```

To force the power to on, write 1 to this attribute file. To disable force power, write 0 to this attribute file.

Note: it's currently not possible to query the force power state of a platform.

## USING UFS

```
mount -t ufs -o ufstype=type_of_ufs device dir
```

### 70.1 UFS Options

**ufstype=type\_of\_ufs** UFS is a file system widely used in different operating systems. The problem are differences among implementations. Features of some implementations are undocumented, so its hard to recognize type of ufs automatically. That' s why user must specify type of ufs manually by mount option ufstype. Possible values are:

**old** old format of ufs default value, supported as read-only

**44bsd** used in FreeBSD, NetBSD, OpenBSD supported as read-write

**ufs2** used in FreeBSD 5.x supported as read-write

**5xbsd** synonym for ufs2

**sun** used in SunOS (Solaris) supported as read-write

**sunx86** used in SunOS for Intel (Solarisx86) supported as read-write

**hp** used in HP-UX supported as read-only

**nextstep** used in NextStep supported as read-only

**nextstep-cd** used for NextStep CDROMs (block\_size == 2048) supported as read-only

**openstep** used in OpenStep supported as read-only

#### 70.1.1 Possible Problems

See next section, if you have any.

### **70.1.2 Bug Reports**

Any ufs bug report you can send to [daniel.pirkl@email.cz](mailto:daniel.pirkl@email.cz) or to [dushistov@mail.ru](mailto:dushistov@mail.ru) (do not send partition tables bug reports).

## **UNICODE SUPPORT**

Last update: 2005-01-17, version 1.4

This file is maintained by H. Peter Anvin <[unicode@lanana.org](mailto:unicode@lanana.org)> as part of the Linux Assigned Names And Numbers Authority (LANANA) project. The current version can be found at:

<http://www.lanana.org/docs/unicode/admin-guide/unicode.rst>

### **71.1 Introduction**

The Linux kernel code has been rewritten to use Unicode to map characters to fonts. By downloading a single Unicode-to-font table, both the eight-bit character sets and UTF-8 mode are changed to use the font as indicated.

This changes the semantics of the eight-bit character tables subtly. The four character tables are now:

Map symbol	Map name	Escape code (G0)
LAT1_MAP	Latin-1 (ISO 8859-1)	ESC ( B
GRAF_MAP	DEC VT100 pseudographics	ESC ( 0
IBMPC_MAP	IBM code page 437	ESC ( U
USER_MAP	User defined	ESC ( K

In particular, ESC ( U is no longer “straight to font” , since the font might be completely different than the IBM character set. This permits for example the use of block graphics even with a Latin-1 font loaded.

Note that although these codes are similar to ISO 2022, neither the codes nor their uses match ISO 2022; Linux has two 8-bit codes (G0 and G1), whereas ISO 2022 has four 7-bit codes (G0-G3).

In accordance with the Unicode standard/ISO 10646 the range U+F000 to U+F8FF has been reserved for OS-wide allocation (the Unicode Standard refers to this as a “Corporate Zone” , since this is inaccurate for Linux we call it the “Linux Zone” ). U+F000 was picked as the starting point since it lets the direct-mapping area start on a large power of two (in case 1024- or 2048-character fonts ever become necessary). This leaves U+E000 to U+EFFF as End User Zone.

[v1.2]: The Unicodes range from U+F000 and up to U+F7FF have been hard-coded to map directly to the loaded font, bypassing the translation table. The

user-defined map now defaults to U+F000 to U+F0FF, emulating the previous behaviour. In practice, this range might be shorter; for example, vgacon can only handle 256-character (U+F000..U+F0FF) or 512-character (U+F000..U+F1FF) fonts.

### 71.2 Actual characters assigned in the Linux Zone

In addition, the following characters not present in Unicode 1.1.4 have been defined; these are used by the DEC VT graphics map. [v1.2] THIS USE IS OBSOLETE AND SHOULD NO LONGER BE USED; PLEASE SEE BELOW.

U+F800	DEC VT GRAPHICS HORIZONTAL LINE SCAN 1
U+F801	DEC VT GRAPHICS HORIZONTAL LINE SCAN 3
U+F803	DEC VT GRAPHICS HORIZONTAL LINE SCAN 7
U+F804	DEC VT GRAPHICS HORIZONTAL LINE SCAN 9

The DEC VT220 uses a 6x10 character matrix, and these characters form a smooth progression in the DEC VT graphics character set. I have omitted the scan 5 line, since it is also used as a block-graphics character, and hence has been coded as U+2500 FORMS LIGHT HORIZONTAL.

[v1.3]: These characters have been officially added to Unicode 3.2.0; they are added at U+23BA, U+23BB, U+23BC, U+23BD. Linux now uses the new values.

[v1.2]: The following characters have been added to represent common keyboard symbols that are unlikely to ever be added to Unicode proper since they are horribly vendor-specific. This, of course, is an excellent example of horrible design.

U+F810	KEYBOARD SYMBOL FLYING FLAG
U+F811	KEYBOARD SYMBOL PULLDOWN MENU
U+F812	KEYBOARD SYMBOL OPEN APPLE
U+F813	KEYBOARD SYMBOL SOLID APPLE

### 71.3 Klingon language support

In 1996, Linux was the first operating system in the world to add support for the artificial language Klingon, created by Marc Okrand for the “Star Trek” television series. This encoding was later adopted by the ConScript Unicode Registry and proposed (but ultimately rejected) for inclusion in Unicode Plane 1. Thus, it remains as a Linux/CSUR private assignment in the Linux Zone.

This encoding has been endorsed by the Klingon Language Institute. For more information, contact them at:

<http://www.kli.org/>

Since the characters in the beginning of the Linux CZ have been more of the dingbats/symbols/forms type and this is a language, I have located it at the end, on a 16-cell boundary in keeping with standard Unicode practice.

**Note:** This range is now officially managed by the ConScript Unicode Registry. The normative reference is at:

<https://www.evertype.com/standards/csur/klington.html>

Klinton has an alphabet of 26 characters, a positional numeric writing system with 10 digits, and is written left-to-right, top-to-bottom.

Several glyph forms for the Klinton alphabet have been proposed. However, since the set of symbols appear to be consistent throughout, with only the actual shapes being different, in keeping with standard Unicode practice these differences are considered font variants.

U+F8D0	KLINGTON LETTER A
U+F8D1	KLINGTON LETTER B
U+F8D2	KLINGTON LETTER CH
U+F8D3	KLINGTON LETTER D
U+F8D4	KLINGTON LETTER E
U+F8D5	KLINGTON LETTER GH
U+F8D6	KLINGTON LETTER H
U+F8D7	KLINGTON LETTER I
U+F8D8	KLINGTON LETTER J
U+F8D9	KLINGTON LETTER L
U+F8DA	KLINGTON LETTER M
U+F8DB	KLINGTON LETTER N
U+F8DC	KLINGTON LETTER NG
U+F8DD	KLINGTON LETTER O
U+F8DE	KLINGTON LETTER P
U+F8DF	KLINGTON LETTER Q - Written <q> in standard Okrand Latin transliteration
U+F8E0	KLINGTON LETTER QH - Written <Q> in standard Okrand Latin transliteration
U+F8E1	KLINGTON LETTER R
U+F8E2	KLINGTON LETTER S
U+F8E3	KLINGTON LETTER T
U+F8E4	KLINGTON LETTER TLH
U+F8E5	KLINGTON LETTER U
U+F8E6	KLINGTON LETTER V
U+F8E7	KLINGTON LETTER W
U+F8E8	KLINGTON LETTER Y
U+F8E9	KLINGTON LETTER GLOTTAL STOP
U+F8F0	KLINGTON DIGIT ZERO
U+F8F1	KLINGTON DIGIT ONE
U+F8F2	KLINGTON DIGIT TWO
U+F8F3	KLINGTON DIGIT THREE
U+F8F4	KLINGTON DIGIT FOUR
U+F8F5	KLINGTON DIGIT FIVE
U+F8F6	KLINGTON DIGIT SIX
U+F8F7	KLINGTON DIGIT SEVEN
U+F8F8	KLINGTON DIGIT EIGHT

Continued on next page

Table 1 - continued from previous page

U+F8F9	KLINGON DIGIT NINE
U+F8FD	KLINGON COMMA
U+F8FE	KLINGON FULL STOP
U+F8FF	KLINGON SYMBOL FOR EMPIRE

## 71.4 Other Fictional and Artificial Scripts

Since the assignment of the Klingon Linux Unicode block, a registry of fictional and artificial scripts has been established by John Cowan <[jcowan@reutershealth.com](mailto:jcowan@reutershealth.com)> and Michael Everson <[everson@evertype.com](mailto:everson@evertype.com)>. The ConScript Unicode Registry is accessible at:

<https://www.evertype.com/standards/csur/>

The ranges used fall at the low end of the End User Zone and can hence not be normatively assigned, but it is recommended that people who wish to encode fictional scripts use these codes, in the interest of interoperability. For Klingon, CSUR has adopted the Linux encoding. The CSUR people are driving adding Tengwar and Cirth into Unicode Plane 1; the addition of Klingon to Unicode Plane 1 has been rejected and so the above encoding remains official.

## SOFTWARE CURSOR FOR VGA

by Pavel Machek <pavel@atrey.karlin.mff.cuni.cz> and Martin Mares <mj@atrey.karlin.mff.cuni.cz>

Linux now has some ability to manipulate cursor appearance. Normally, you can set the size of hardware cursor. You can now play a few new tricks: you can make your cursor look like a non-blinking red block, make it inverse background of the character it's over or to highlight that character and still choose whether the original hardware cursor should remain visible or not. There may be other things I have never thought of.

The cursor appearance is controlled by a <ESC>[?1;2;3c escape sequence where 1, 2 and 3 are parameters described below. If you omit any of them, they will default to zeroes.

**first Parameter** specifies cursor size:

```
0=default
1=invisible
2=underline,
...
8=full block
+ 16 if you want the software cursor to be applied
+ 32 if you want to always change the background color
+ 64 if you dislike having the background the same as the
    foreground.
```

Highlights are ignored for the last two flags.

**second parameter** selects character attribute bits you want to change (by simply XORing them with the value of this parameter). On standard VGA, the high four bits specify background and the low four the foreground. In both groups, low three bits set color (as in normal color codes used by the console) and the most significant one turns on highlight (or sometimes blinking - it depends on the configuration of your VGA).

**third parameter** consists of character attribute bits you want to set.

Bit setting takes place before bit toggling, so you can simply clear a bit by including it in both the set mask and the toggle mask.

## 72.1 Examples

To get normal blinking underline, use:

```
echo -e '\033[?2c'
```

To get blinking block, use:

```
echo -e '\033[?6c'
```

To get red non-blinking block, use:

```
echo -e '\033[?17;0;64c'
```

## **VIDEO OUTPUT SWITCHER CONTROL**

2006 [luming.yu@intel.com](mailto:luming.yu@intel.com)

The output sysfs class driver provides an abstract video output layer that can be used to hook platform specific methods to enable/disable video output device through common sysfs interface. For example, on my IBM ThinkPad T42 laptop, The ACPI video driver registered its output devices and read/write method for 'state' with output sysfs class. The user interface under sysfs is:

```
linux:/sys/class/video_output # tree .
.
|-- CRT0
|   |-- device -> ../../../../devices/pci0000:00/0000:00:01.0
|   |-- state
|   |-- subsystem -> ../../../../class/video_output
|   `-- uevent
|-- DVI0
|   |-- device -> ../../../../devices/pci0000:00/0000:00:01.0
|   |-- state
|   |-- subsystem -> ../../../../class/video_output
|   `-- uevent
|-- LCD0
|   |-- device -> ../../../../devices/pci0000:00/0000:00:01.0
|   |-- state
|   |-- subsystem -> ../../../../class/video_output
|   `-- uevent
`-- TV0
    |-- device -> ../../../../devices/pci0000:00/0000:00:01.0
    |-- state
    |-- subsystem -> ../../../../class/video_output
    `-- uevent
```



## **WIMAX SUBSYSTEM**

### **74.1 Linux kernel WiMAX stack**

**Copyright** © 2008 Intel Corporation < [linux-wimax@intel.com](mailto:linux-wimax@intel.com) >

This provides a basic Linux kernel WiMAX stack to provide a common control API for WiMAX devices, usable from kernel and user space.

#### **74.1.1 1. Design**

The WiMAX stack is designed to provide for common WiMAX control services to current and future WiMAX devices from any vendor.

Because currently there is only one and we don't know what would be the common services, the APIs it currently provides are very minimal. However, it is done in such a way that it is easily extensible to accommodate future requirements.

The stack works by embedding a struct `wimax_dev` in your device's control structures. This provides a set of callbacks that the WiMAX stack will call in order to implement control operations requested by the user. As well, the stack provides API functions that the driver calls to notify about changes of state in the device.

The stack exports the API calls needed to control the device to user space using generic netlink as a marshalling mechanism. You can access them using your own code or use the wrappers provided for your convenience in `libwimax` (in the `wimax-tools` package).

For detailed information on the stack, please see `include/linux/wimax.h`.

### 74.1.2 2. Usage

For usage in a driver (registration, API, etc) please refer to the instructions in the header file `include/linux/wimax.h`.

When a device is registered with the WiMAX stack, a set of debugfs files will appear in `/sys/kernel/debug/wimax:wmxX` can tweak for control.

#### 2.1. Obtaining debug information: debugfs entries

The WiMAX stack is compiled, by default, with debug messages that can be used to diagnose issues. By default, said messages are disabled.

The drivers will register debugfs entries that allow the user to tweak debug settings.

Each driver, when registering with the stack, will cause a debugfs directory named `wimax:DEVICENAME` to be created; optionally, it might create more subentries below it.

##### 2.1.1. Increasing debug output

The files named `dl_` indicate knobs for controlling the debug output of different submodules of the WiMAX stack:

```
# find /sys/kernel/debug/wimax\:wmx0 -name \*dl_\*
/sys/kernel/debug/wimax:wmx0/wimax_dl_stack
/sys/kernel/debug/wimax:wmx0/wimax_dl_op_rfkill
/sys/kernel/debug/wimax:wmx0/wimax_dl_op_reset
/sys/kernel/debug/wimax:wmx0/wimax_dl_op_msg
/sys/kernel/debug/wimax:wmx0/wimax_dl_id_table
/sys/kernel/debug/wimax:wmx0/wimax_dl_debugfs
/sys/kernel/debug/wimax:wmx0/.... # other driver specific files
```

**NOTE:** Of course, if debugfs is mounted in a directory other than `/sys/kernel/debug`, those paths will change.

By reading the file you can obtain the current value of said debug level; by writing to it, you can set it.

To increase the debug level of, for example, the `id-table` submodule, just write:

```
$ echo 3 > /sys/kernel/debug/wimax:wmx0/wimax_dl_id_table
```

Increasing numbers yield increasing debug information; for details of what is printed and the available levels, check the source. The code uses 0 for disabled and increasing values until 8.

## 74.2 Driver for the Intel Wireless Wimax Connection 2400m

**Copyright** © 2008 Intel Corporation < [linux-wimax@intel.com](mailto:linux-wimax@intel.com) >

This provides a driver for the Intel Wireless WiMAX Connection 2400m and a basic Linux kernel WiMAX stack.

### 74.2.1 1. Requirements

- Linux installation with Linux kernel 2.6.22 or newer (if building from a separate tree)
- Intel i2400m Echo Peak or Baxter Peak; this includes the Intel Wireless WiMAX/WiFi Link 5x50 series.
- build tools:
  - Linux kernel development package for the target kernel; to build against your currently running kernel, you need to have the kernel development package corresponding to the running image installed (usually if your kernel is named linux-VERSION, the development package is called linux-dev-VERSION or linux-headers-VERSION).
  - GNU C Compiler, make

### 74.2.2 2. Compilation and installation

#### 2.1. Compilation of the drivers included in the kernel

Configure the kernel; to enable the WiMAX drivers select Drivers > Networking Drivers > WiMAX device support. Enable all of them as modules (easier).

If USB or SDIO are not enabled in the kernel configuration, the options to build the i2400m USB or SDIO drivers will not show. Enable said subsystems and go back to the WiMAX menu to enable the drivers.

Compile and install your kernel as usual.

#### 2.2. Compilation of the drivers distributed as an standalone module

To compile:

```
$ cd source/directory
$ make
```

Once built you can load and unload using the provided load.sh script; load.sh will load the modules, load.sh u will unload them.

To install in the default kernel directories (and enable auto loading when the device is plugged):

```
$ make install
$ depmod -a
```

If your kernel development files are located in a non standard directory or if you want to build for a kernel that is not the currently running one, set KDIR to the right location:

```
$ make KDIR=/path/to/kernel/dev/tree
```

For more information, please contact [linux-wimax@intel.com](mailto:linux-wimax@intel.com).

### 3. Installing the firmware

The firmware can be obtained from <http://linuxwimax.org> or might have been supplied with your hardware.

It has to be installed in the target system:

```
$ cp FIRMWAREFILE.sbcf /lib/firmware/i2400m-fw-BUSTYPE-1.3.sbcf
* NOTE: if your firmware came in an .rpm or .deb file, just
↳ install
  it as normal, with the rpm (rpm -i FIRMWARE.rpm) or dpkg
  (dpkg -i FIRMWARE.deb) commands. No further action is needed.
* BUSTYPE will be usb or sdio, depending on the hardware you have.
  Each hardware type comes with its own firmware and will not work
  with other types.
```

#### 74.2.3 4. Design

This package contains two major parts: a WiMAX kernel stack and a driver for the Intel i2400m.

The WiMAX stack is designed to provide for common WiMAX control services to current and future WiMAX devices from any vendor; please see README.wimax for details.

The i2400m kernel driver is broken up in two main parts: the bus generic driver and the bus-specific drivers. The bus generic driver forms the drivercore and contain no knowledge of the actual method we use to connect to the device. The bus specific drivers are just the glue to connect the bus-generic driver and the device. Currently only USB and SDIO are supported. See `drivers/net/wimax/i2400m/i2400m.h` for more information.

The bus generic driver is logically broken up in two parts: OS-glue and hardware-glue. The OS-glue interfaces with Linux. The hardware-glue interfaces with the device on using an interface provided by the bus-specific driver. The reason for this breakup is to be able to easily reuse the hardware-glue to write drivers for other OSes; note the hardware glue part is written as a native Linux driver; no abstraction layers are used, so to port to another OS, the Linux kernel API calls should be replaced with the target OS' s.

## 74.2.4 5. Usage

To load the driver, follow the instructions in the install section; once the driver is loaded, plug in the device (unless it is permanently plugged in). The driver will enumerate the device, upload the firmware and output messages in the kernel log (dmesg, /var/log/messages or /var/log/kern.log) such as:

```
...
i2400m_usb 5-4:1.0: firmware interface version 8.0.0
i2400m_usb 5-4:1.0: WiMAX interface wmx0 (00:1d:e1:01:94:2c) ready
```

At this point the device is ready to work.

Current versions require the Intel WiMAX Network Service in userspace to make things work. See the network service's README for instructions on how to scan, connect and disconnect.

### 5.1. Module parameters

Module parameters can be set at kernel or module load time or by echoing values:

```
$ echo VALUE > /sys/module/MODULENAME/parameters/PARAMETERNAME
```

To make changes permanent, for example, for the i2400m module, you can also create a file named /etc/modprobe.d/i2400m containing:

```
options i2400m idle_mode_disabled=1
```

To find which parameters are supported by a module, run:

```
$ modinfo path/to/module.ko
```

During kernel bootup (if the driver is linked in the kernel), specify the following to the kernel command line:

```
i2400m.PARAMETER=VALUE
```

#### 5.1.1. i2400m: idle\_mode\_disabled

The i2400m module supports a parameter to disable idle mode. This parameter, once set, will take effect only when the device is reinitialized by the driver (eg: following a reset or a reconnect).

## 5.2. Debug operations: debugfs entries

The driver will register debugfs entries that allow the user to tweak debug settings. There are three main container directories where entries are placed, which correspond to the three blocks a i2400m WiMAX driver has:

- /sys/kernel/debug/wimax:DEVNAME/ for the generic WiMAX stack controls
- /sys/kernel/debug/wimax:DEVNAME/i2400m for the i2400m generic driver controls
- /sys/kernel/debug/wimax:DEVNAME/i2400m-usb (or -sdio) for the bus-specific i2400m-usb or i2400m-sdio controls).

Of course, if debugfs is mounted in a directory other than /sys/kernel/debug, those paths will change.

### 5.2.1. Increasing debug output

The files named dl\_ indicate knobs for controlling the debug output of different submodules:

```
# find /sys/kernel/debug/wimax\:wmx0 -name \*dl_\*
/sys/kernel/debug/wimax:wmx0/i2400m-usb/dl_tx
/sys/kernel/debug/wimax:wmx0/i2400m-usb/dl_rx
/sys/kernel/debug/wimax:wmx0/i2400m-usb/dl_notif
/sys/kernel/debug/wimax:wmx0/i2400m-usb/dl_fw
/sys/kernel/debug/wimax:wmx0/i2400m-usb/dl_usb
/sys/kernel/debug/wimax:wmx0/i2400m/dl_tx
/sys/kernel/debug/wimax:wmx0/i2400m/dl_rx
/sys/kernel/debug/wimax:wmx0/i2400m/dl_rfkill
/sys/kernel/debug/wimax:wmx0/i2400m/dl_netdev
/sys/kernel/debug/wimax:wmx0/i2400m/dl_fw
/sys/kernel/debug/wimax:wmx0/i2400m/dl_debugfs
/sys/kernel/debug/wimax:wmx0/i2400m/dl_driver
/sys/kernel/debug/wimax:wmx0/i2400m/dl_control
/sys/kernel/debug/wimax:wmx0/wimax_dl_stack
/sys/kernel/debug/wimax:wmx0/wimax_dl_op_rfkill
/sys/kernel/debug/wimax:wmx0/wimax_dl_op_reset
/sys/kernel/debug/wimax:wmx0/wimax_dl_op_msg
/sys/kernel/debug/wimax:wmx0/wimax_dl_id_table
/sys/kernel/debug/wimax:wmx0/wimax_dl_debugfs
```

By reading the file you can obtain the current value of said debug level; by writing to it, you can set it.

To increase the debug level of, for example, the i2400m' s generic TX engine, just write:

```
$ echo 3 > /sys/kernel/debug/wimax:wmx0/i2400m/dl_tx
```

Increasing numbers yield increasing debug information; for details of what is printed and the available levels, check the source. The code uses 0 for disabled and increasing values until 8.

### 5.2.2. RX and TX statistics

The `i2400m/rx_stats` and `i2400m/tx_stats` provide statistics about the data reception/delivery from the device:

```
$ cat /sys/kernel/debug/wimax:wmx0/i2400m/rx_stats
45 1 3 34 3104 48 480
```

The numbers reported are:

- packets/RX-buffer: total, min, max
- RX-buffers: total RX buffers received, accumulated RX buffer size in bytes, min size received, max size received

Thus, to find the average buffer size received, divide accumulated RX-buffer / total RX-buffers.

To clear the statistics back to 0, write anything to the `rx_stats` file:

```
$ echo 1 > /sys/kernel/debug/wimax:wmx0/i2400m_rx_stats
```

Likewise for TX.

Note the packets this debug file refers to are not network packet, but packets in the sense of the device-specific protocol for communication to the host. See `drivers/net/wimax/i2400m/tx.c`.

### 5.2.3. Tracing messages received from user space

To echo messages received from user space into the trace pipe that the `i2400m` driver creates, set the debug file `i2400m/trace_msg_from_user` to 1:

```
$ echo 1 > /sys/kernel/debug/wimax:wmx0/i2400m/trace_msg_from_user
```

### 5.2.4. Performing a device reset

By writing a 0, a 1 or a 2 to the file `/sys/kernel/debug/wimax:wmx0/reset`, the driver performs a warm (without disconnecting from the bus), cold (disconnecting from the bus) or bus (bus specific) reset on the device.

### 5.2.5. Asking the device to enter power saving mode

By writing any value to the `/sys/kernel/debug/wimax:wmx0` file, the device will attempt to enter power saving mode.

### 74.2.5 6. Troubleshooting

#### 6.1. Driver complains about i2400m-fw-usb-1.2.sbcf: request failed

If upon connecting the device, the following is output in the kernel log:

```
i2400m_usb 5-4:1.0: fw i2400m-fw-usb-1.3.sbcf: request failed: -2
```

This means that the driver cannot locate the firmware file named `/lib/firmware/i2400m-fw-usb-1.2.sbcf`. Check that the file is present in the right location.

## THE SGI XFS FILESYSTEM

XFS is a high performance journaling filesystem which originated on the SGI IRIX platform. It is completely multi-threaded, can support large files and large filesystems, extended attributes, variable block sizes, is extent based, and makes extensive use of Btrees (directories, extents, free space) to aid both performance and scalability.

Refer to the documentation at <https://xfs.wiki.kernel.org/> for further details. This implementation is on-disk compatible with the IRIX version of XFS.

### 75.1 Mount Options

When mounting an XFS filesystem, the following options are accepted.

**allocsize=size** Sets the buffered I/O end-of-file preallocation size when doing delayed allocation writeout (default size is 64KiB). Valid values for this option are page size (typically 4KiB) through to 1GiB, inclusive, in power-of-2 increments.

The default behaviour is for dynamic end-of-file preallocation size, which uses a set of heuristics to optimise the preallocation size based on the current allocation patterns within the file and the access patterns to the file. Specifying a fixed `allocsize` value turns off the dynamic behaviour.

**attr2 or noattr2** The options enable/disable an “opportunistic” improvement to be made in the way inline extended attributes are stored on-disk. When the new form is used for the first time when `attr2` is selected (either when setting or removing extended attributes) the on-disk superblock feature bit field will be updated to reflect this format being in use.

The default behaviour is determined by the on-disk feature bit indicating that `attr2` behaviour is active. If either mount option is set, then that becomes the new default used by the filesystem.

CRC enabled filesystems always use the `attr2` format, and so will reject the `noattr2` mount option if it is set.

**discard or nodiscard (default)** Enable/disable the issuing of commands to let the block device reclaim space freed by the filesystem.

This is useful for SSD devices, thinly provisioned LUNs and virtual machine images, but may have a performance impact.

Note: It is currently recommended that you use the `fstrim` application to discard unused blocks rather than the `discard` mount option because the performance impact of this option is quite severe.

**grpuid/bsdgroups or nogrpuid/sysvgroups (default)** These options define what group ID a newly created file gets. When `grpuid` is set, it takes the group ID of the directory in which it is created; otherwise it takes the `fsgid` of the current process, unless the directory has the `setgid` bit set, in which case it takes the `gid` from the parent directory, and also gets the `setgid` bit set if it is a directory itself.

**filestreams** Make the data allocator use the filestreams allocation mode across the entire filesystem rather than just on directories configured to use it.

**ikeep or noikeep (default)** When `ikeep` is specified, XFS does not delete empty inode clusters and keeps them around on disk. When `noikeep` is specified, empty inode clusters are returned to the free space pool.

**inode32 or inode64 (default)** When `inode32` is specified, it indicates that XFS limits inode creation to locations which will not result in inode numbers with more than 32 bits of significance.

When `inode64` is specified, it indicates that XFS is allowed to create inodes at any location in the filesystem, including those which will result in inode numbers occupying more than 32 bits of significance.

`inode32` is provided for backwards compatibility with older systems and applications, since 64 bits inode numbers might cause problems for some applications that cannot handle large inode numbers. If applications are in use which do not handle inode numbers bigger than 32 bits, the `inode32` option should be specified.

**largeio or nolargeio (default)** If `nolargeio` is specified, the optimal I/O reported in `st_blksize` by **stat(2)** will be as small as possible to allow user applications to avoid inefficient read/modify/write I/O. This is typically the page size of the machine, as this is the granularity of the page cache.

If `largeio` is specified, a filesystem that was created with a `swidth` specified will return the `swidth` value (in bytes) in `st_blksize`. If the filesystem does not have a `swidth` specified but does specify an `allocsize` then `allocsize` (in bytes) will be returned instead. Otherwise the behaviour is the same as if `nolargeio` was specified.

**logbufs=value** Set the number of in-memory log buffers. Valid numbers range from 2-8 inclusive.

The default value is 8 buffers.

If the memory cost of 8 log buffers is too high on small systems, then it may be reduced at some cost to performance on metadata

intensive workloads. The `logbsize` option below controls the size of each buffer and so is also relevant to this case.

**logbsize=value** Set the size of each in-memory log buffer. The size may be specified in bytes, or in kilobytes with a “k” suffix. Valid sizes for version 1 and version 2 logs are 16384 (16k) and 32768 (32k). Valid sizes for version 2 logs also include 65536 (64k), 131072 (128k) and 262144 (256k). The `logbsize` must be an integer multiple of the log stripe unit configured at **mkfs(8)** time.

The default value for for version 1 logs is 32768, while the default value for version 2 logs is `MAX(32768, log_sunit)`.

**logdev=device and rtdev=device** Use an external log (metadata journal) and/or real-time device. An XFS filesystem has up to three parts: a data section, a log section, and a real-time section. The real-time section is optional, and the log section can be separate from the data section or contained within it.

**noalign** Data allocations will not be aligned at stripe unit boundaries. This is only relevant to filesystems created with non-zero data alignment parameters (`sunit`, `swidth`) by **mkfs(8)**.

**norecovery** The filesystem will be mounted without running log recovery. If the filesystem was not cleanly unmounted, it is likely to be inconsistent when mounted in `norecovery` mode. Some files or directories may not be accessible because of this. Filesystems mounted `norecovery` must be mounted read-only or the mount will fail.

**nouuid** Don't check for double mounted file systems using the file system `uuid`. This is useful to mount LVM snapshot volumes, and often used in combination with `norecovery` for mounting read-only snapshots.

**noquota** Forcibly turns off all quota accounting and enforcement within the filesystem.

**uquota/usrquota/uqnoenforce/quota** User disk quota accounting enabled, and limits (optionally) enforced. Refer to **xfs\_quota(8)** for further details.

**gquota/grpquota/gqnoenforce** Group disk quota accounting enabled and limits (optionally) enforced. Refer to **xfs\_quota(8)** for further details.

**pquota/prjquota/pqnoenforce** Project disk quota accounting enabled and limits (optionally) enforced. Refer to **xfs\_quota(8)** for further details.

**sunit=value and swidth=value** Used to specify the stripe unit and width for a RAID device or a stripe volume. “value” must be specified in 512-byte block units. These options are only relevant to filesystems that were created with non-zero data alignment parameters.

The `sunit` and `swidth` parameters specified must be compatible with the existing filesystem alignment characteristics. In general, that means the only valid changes to `sunit` are increasing it by a

power-of-2 multiple. Valid swidth values are any integer multiple of a valid sunit value.

Typically the only time these mount options are necessary is after an underlying RAID device has had its geometry modified, such as adding a new disk to a RAID5 lun and reshaping it.

**swalloc** Data allocations will be rounded up to stripe width boundaries when the current end of file is being extended and the file size is larger than the stripe width size.

**wsync** When specified, all filesystem namespace operations are executed synchronously. This ensures that when the namespace operation (create, unlink, etc) completes, the change to the namespace is on stable storage. This is useful in HA setups where failover must not result in clients seeing inconsistent namespace presentation during or after a failover event.

## 75.2 Deprecated Mount Options

Name	Removal Schedule

## 75.3 Removed Mount Options

Name	Removed
delaylog/nodelaylog	v4.0
ihashsize	v4.0
irixsgid	v4.0
osyncisdsync/osyncisosync	v4.0
barrier	v4.19
nobarrier	v4.19

## 75.4 sysctls

The following sysctls are available for the XFS filesystem:

**fs.xfs.stats\_clear (Min: 0 Default: 0 Max: 1)** Setting this to “1” clears accumulated XFS statistics in /proc/fs/xfs/stat. It then immediately resets to “0” .

**fs.xfs.xfssyncd\_centisecs (Min: 100 Default: 3000 Max: 720000)**  
The interval at which the filesystem flushes metadata out to disk and runs internal cache cleanup routines.

**fs.xfs.filestream\_centisecs (Min: 1 Default: 3000 Max: 360000)**  
The interval at which the filesystem ages filestreams cache references and returns timed-out AGs back to the free stream pool.

**fs.xfs.speculative\_prealloc\_lifetime** (Units: seconds Min: 1 Default: 300 Max: 86400) The interval at which the background scanning for inodes with unused speculative preallocation runs. The scan removes unused preallocation from clean inodes and releases the unused space back to the free pool.

**fs.xfs.error\_level** (Min: 0 Default: 3 Max: 11) A volume knob for error reporting when internal errors occur. This will generate detailed messages & backtraces for filesystem shutdowns, for example. Current threshold values are:

```
XFS_ERRLEVEL_OFF:    0    XFS_ERRLEVEL_LOW:    1
XFS_ERRLEVEL_HIGH:  5
```

**fs.xfs.panic\_mask** (Min: 0 Default: 0 Max: 256) Causes certain error conditions to call BUG(). Value is a bitmask; OR together the tags which represent errors which should cause panics:

```
XFS_NO_PTAG      0    XFS_PTAG_IFLUSH      0x00000001
XFS_PTAG_LOGRES  0x00000002 XFS_PTAG_AILDELETE
0x00000004          XFS_PTAG_ERROR_REPORT
0x00000008          XFS_PTAG_SHUTDOWN_CORRUPT
0x00000010          XFS_PTAG_SHUTDOWN_IOERROR
0x00000020          XFS_PTAG_SHUTDOWN_LOGERROR
0x00000040 XFS_PTAG_FSBLOCK_ZERO 0x00000080
XFS_PTAG_VERIFIER_ERROR 0x00000100
```

This option is intended for debugging only.

**fs.xfs.irix\_symlink\_mode** (Min: 0 Default: 0 Max: 1) Controls whether symlinks are created with mode 0777 (default) or whether their mode is affected by the umask (irix mode).

**fs.xfs.irix\_sgid\_inherit** (Min: 0 Default: 0 Max: 1) Controls files created in SGID directories. If the group ID of the new file does not match the effective group ID or one of the supplementary group IDs of the parent dir, the ISGID bit is cleared if the `irix_sgid_inherit` compatibility sysctl is set.

**fs.xfs.inherit\_sync** (Min: 0 Default: 1 Max: 1) Setting this to “1” will cause the “sync” flag set by the `xfs_io(8)` chattr command on a directory to be inherited by files in that directory.

**fs.xfs.inherit\_nodump** (Min: 0 Default: 1 Max: 1) Setting this to “1” will cause the “nodump” flag set by the `xfs_io(8)` chattr command on a directory to be inherited by files in that directory.

**fs.xfs.inherit\_noatime** (Min: 0 Default: 1 Max: 1) Setting this to “1” will cause the “noatime” flag set by the `xfs_io(8)` chattr command on a directory to be inherited by files in that directory.

**fs.xfs.inherit\_nosymlinks** (Min: 0 Default: 1 Max: 1) Setting this to “1” will cause the “nosymlinks” flag set by the `xfs_io(8)` chattr command on a directory to be inherited by files in that directory.

**fs.xfs.inherit\_nodfrag** (Min: 0 Default: 1 Max: 1) Setting this to

“1” will cause the “nodefrag” flag set by the **xfs\_io(8)** `chattr` command on a directory to be inherited by files in that directory.

**fs.xfs.rotorstep (Min: 1 Default: 1 Max: 256)** In “inode32” allocation mode, this option determines how many files the allocator attempts to allocate in the same allocation group before moving to the next allocation group. The intent is to control the rate at which the allocator moves between allocation groups when allocating extents for new files.

## 75.5 Deprecated Sysctls

None at present.

## 75.6 Removed Sysctls

Name	Removed
fs.xfs.xfsbufd_centisec	v4.0
fs.xfs.age_buffer_centisecs	v4.0

## 75.7 Error handling

XFS can act differently according to the type of error found during its operation. The implementation introduces the following concepts to the error handler:

- failure speed:** Defines how fast XFS should propagate an error upwards when a specific error is found during the filesystem operation. It can propagate immediately, after a defined number of retries, after a set time period, or simply retry forever.
- error classes:** Specifies the subsystem the error configuration will apply to, such as metadata IO or memory allocation. Different subsystems will have different error handlers for which behaviour can be configured.
- error handlers:** Defines the behavior for a specific error.

The filesystem behavior during an error can be set via `sysfs` files. Each error handler works independently - the first condition met by an error handler for a specific class will cause the error to be propagated rather than reset and retried.

The action taken by the filesystem when the error is propagated is context dependent - it may cause a shut down in the case of an unrecoverable error, it may be reported back to userspace, or it may even be ignored because there's nothing useful we can with the error or anyone we can report it to (e.g. during unmount).

The configuration files are organized into the following hierarchy for each mounted filesystem:

```
/sys/fs/xfs/<dev>/error/<class>/<error>/
```

**Where:**

**<dev>** The short device name of the mounted filesystem. This is the same device name that shows up in XFS kernel error messages as “XFS(<dev>): ...”

**<class>** The subsystem the error configuration belongs to. As of 4.9, the defined classes are:

- “metadata” : applies metadata buffer write IO

**<error>** The individual error handler configurations.

Each filesystem has “global” error configuration options defined in their top level directory:

```
/sys/fs/xfs/<dev>/error/
```

**fail\_at\_unmount (Min: 0 Default: 1 Max: 1)** Defines the filesystem error behavior at unmount time.

If set to a value of 1, XFS will override all other error configurations during unmount and replace them with “immediate fail” characteristics. i.e. no retries, no retry timeout. This will always allow unmount to succeed when there are persistent errors present.

If set to 0, the configured retry behaviour will continue until all retries and/or timeouts have been exhausted. This will delay unmount completion when there are persistent errors, and it may prevent the filesystem from ever unmounting fully in the case of “retry forever” handler configurations.

Note: there is no guarantee that fail\_at\_unmount can be set while an unmount is in progress. It is possible that the sys fs entries are removed by the unmounting filesystem before a “retry forever” error handler configuration causes unmount to hang, and hence the filesystem must be configured appropriately before unmount begins to prevent unmount hangs.

Each filesystem has specific error class handlers that define the error propagation behaviour for specific errors. There is also a “default” error handler defined, which defines the behaviour for all errors that don’ t have specific handlers defined. Where multiple retry constraints are configured for a single error, the first retry configuration that expires will cause the error to be propagated. The handler configurations are found in the directory:

```
/sys/fs/xfs/<dev>/error/<class>/<error>/
```

**max\_retries (Min: -1 Default: Varies Max: INTMAX)** Defines the allowed number of retries of a specific error before the filesystem will propagate the error. The retry count for a given error context (e.g. a specific metadata buffer) is reset every time there is a successful completion of the operation.

Setting the value to “-1” will cause XFS to retry forever for this specific error.

Setting the value to “0” will cause XFS to fail immediately when the specific error is reported.

Setting the value to “N” (where  $0 < N < \text{Max}$ ) will make XFS retry the operation “N” times before propagating the error.

**retry\_timeout\_seconds (Min: -1 Default: Varies Max: 1 day)**

Define the amount of time (in seconds) that the filesystem is allowed to retry its operations when the specific error is found.

Setting the value to “-1” will allow XFS to retry forever for this specific error.

Setting the value to “0” will cause XFS to fail immediately when the specific error is reported.

Setting the value to “N” (where  $0 < N < \text{Max}$ ) will allow XFS to retry the operation for up to “N” seconds before propagating the error.

**Note:** The default behaviour for a specific error handler is dependent on both the class and error context. For example, the default values for “metadata/ENODEV” are “0” rather than “-1” so that this error handler defaults to “fail immediately” behaviour. This is done because ENODEV is a fatal, unrecoverable error no matter how many times the metadata IO is retried.