Linux Parisc Documentation

The kernel development community

Jul 14, 2020

CONTENTS

CHAPTER

ONE

PA-RISC DEBUGGING

okay, here are some hints for debugging the lower-level parts of linux/parisc.

1.1 1. Absolute addresses

A lot of the assembly code currently runs in real mode, which means absolute addresses are used instead of virtual addresses as in the rest of the kernel. To translate an absolute address to a virtual address you can lookup in System.map, add __PAGE_OFFSET (0x10000000 currently).

1.2 2. HPMCs

When real-mode code tries to access non-existent memory, you' ll get an HPMC instead of a kernel oops. To debug an HPMC, try to find the System Responder/Requestor addresses. The System Requestor address should match (one of the) processor HPAs (high addresses in the I/O range); the System Responder address is the address real-mode code tried to access.

Typical values for the System Responder address are addresses larger than __PAGE_OFFSET (0x10000000) which mean a virtual address didn' t get translated to a physical address before real-mode code tried to access it.

1.3 3. Q bit fun

Certain, very critical code has to clear the Q bit in the PSW. What happens when the Q bit is cleared is the CPU does not update the registers interruption handlers read to find out where the machine was interrupted - so if you get an interruption between the instruction that clears the Q bit and the RFI that sets it again you don' t know where exactly it happened. If you' re lucky the IAOQ will point to the instruction that cleared the Q bit, if you' re not it points anywhere at all. Usually Q bit problems will show themselves in unexplainable system hangs or running off the end of physical memory.

CHAPTER

TWO

REGISTER USAGE FOR LINUX/PA-RISC

[an asterisk is used for planned usage which is currently unimplemented]

2.1 General Registers as specified by ABI

2.1.1 Control Registers

CR 0 (Recovery Counter)	used for ptrace
CR 1-CR 7(undefined)	unused
CR 8 (Protection ID)	per-process value*
CR 9, 12, 13 (PIDS)	unused
CR10 (CCR)	lazy FPU saving*
CR11	as specified by ABI (SAR)
CR14 (interruption vector)	initialized to fault_vector
CR15 (EIEM)	initialized to all ones*
CR16 (Interval Timer)	read for cycle count/write starts Interval Tmr
CR17-CR22	interruption parameters
CR19	Interrupt Instruction Register
CR20	Interrupt Space Register
CR21	Interrupt Offset Register
CR22	Interrupt PSW
CR23 (EIRR)	read for pending interrupts/write clears bits
CR24 (TR 0)	Kernel Space Page Directory Pointer
CR25 (TR 1)	User Space Page Directory Pointer
CR26 (TR 2)	not used
CR27 (TR 3)	Thread descriptor pointer
CR28 (TR 4)	not used
CR29 (TR 5)	not used
CR30 (TR 6)	current / 0
CR31 (TR 7)	Temporary register, used in various places

2.1.2 Space Registers (kernel mode)

SR0	temporary space register
SR4-SR7	set to 0
SR1	temporary space register
SR2	kernel should not clobber this
SR3	used for userspace accesses (current process)

2.1.3 Space Registers (user mode)

SR0	temporary space register
SR1	temporary space register
SR2	holds space of linux gateway page
SR3	holds user address space value while in kernel
SR4-SR7	Defines short address space for user/kernel

2.1.4 Processor Status Word

W (64-bit addresses)	0
E (Little-endian)	0
S (Secure Interval Timer)	0
T (Taken Branch Trap)	0
H (Higher-privilege trap)	0
L (Lower-privilege trap)	0
N (Nullify next instruction)	used by C code
X (Data memory break disable)	0
B (Taken Branch)	used by C code
C (code address translation)	1, 0 while executing real-mode code
V (divide step correction)	used by C code
M (HPMC mask)	0, 1 while executing HPMC handler*
C/B (carry/borrow bits)	used by C code
O (ordered references)	1*
F (performance monitor)	0
R (Recovery Counter trap)	0
Q (collect interruption state)	1 (0 in code directly preceding an rfi)
P (Protection Identifiers)	1*
D (Data address translation)	1, 0 while executing real-mode code
I (external interrupt mask)	used by cli()/sti() macros

2.1.5 “Invisible” Registers

PSW default W value	0
PSW default E value	0
Shadow Registers	used by interruption handler code
TOC enable bit	1

The PA-RISC architecture defines 7 registers as “shadow registers”. Those are used in RETURN FROM INTERRUPTION AND RESTORE instruction to reduce the state save and restore time by eliminating the need for general register (GR) saves and restores in interruption handlers. Shadow registers are the GRs 1, 8, 9, 16, 17, 24, and 25.

Register usage notes, originally from John Marvin, with some additional notes from Randolph Chung.

For the general registers:

r1,r2,r19-r26,r28,r29 & r31 can be used without saving them first. And of course, you need to save them if you care about them, before calling another procedure. Some of the above registers do have special meanings that you should be aware of:

r1: The addil instruction is hardwired to place its result in r1, so if you use that instruction be aware of that.

r2: This is the return pointer. In general you don’t want to use this, since you need the pointer to get back to your caller. However, it is grouped with this set of registers since the caller can’t rely on the value being the same when you return, i.e. you can copy r2 to another register and return through that register after trashing r2, and that should not cause a problem for the calling routine.

r19-r22: these are generally regarded as temporary registers. Note that in 64 bit they are arg7-arg4.

r23-r26: these are arg3-arg0, i.e. you can use them if you don’t care about the values that were passed in anymore.

r28,r29: are ret0 and ret1. They are what you pass return values in. r28 is the primary return. When returning small structures r29 may also be used to pass data back to the caller.

r30: stack pointer

r31: the ble instruction puts the return pointer in here.

r3-r18,r27,r30 need to be saved and restored. r3-r18 are just general purpose registers. r27 is the data pointer, and is used to make references to global variables easier. r30 is the stack pointer.