

ECE 471 – Embedded Systems

Lecture 12

Vince Weaver

`http://www.eece.maine.edu/~vweaver`

`vincent.weaver@maine.edu`

9 October 2014

Announcements

- Homework grades have been sent out, let me know if you did not receive them.
- A midterm exam is coming up soon.



Device Detection

- x86, well-known standardized platform. What windows needs to boot. Can auto-discover things like PCI bus, USB. Linux kernel on x86 can boot on most.
- Old ARM, hard-coded. So a rasp-pi kernel only could boot on Rasp-pi. Lots of pound-defined and hard-coded hw info.
- New way, device tree. A blob that describes the hardware. Pass it in with boot loader, and kernel can use



it to determine what hardware is available. So instead of Debian needing to provide 100 kernels, instead just 1 kernel and 100 device tree files that one is chosen at install time.

- Does mean that updating to a new kernel can be a pain.



Detecting Devices

There are many ways to detect devices

- Guessing/Probing – can be bad if you guess wrong and the hardware reacts poorly to having unexpected data sent to it
- Standards – always knowing that, say, VGA is at address 0xa0000. PCs get by with defacto standards
- Enumerable hardware – busses like USB and PCI allow you to query hardware to find out what it is and where



it is located

- Hard-coding – have a separate kernel for each possible board, with the locations of devices hard-coded in. Not very maintainable in the long run.
- Device Trees – see next slide



Devicetree

- Traditional Linux ARM support a bit of a copy-paste and `#ifdef` mess
- Each new platform was a compile option. No common code; kernel for pandaboard not run on beagleboard not run on gumstix, etc.
- Work underway to be more like x86 (where until recently due to PC standards a kernel would boot on any x86)
- A “devicetree” passes in enough config info to the kernel



to describe all the hardware available. Thus kernel much more generic

- Still working on issues with this.



Interrupts

- What are interrupts?
- Something happens, signal pulled up, CPU stops executing (possibly flushing the pipeline), PC changes to handler address, may look up in vector table.
- Alternatives (polling)
- Precise vs non-precise
- Hardware issues: (complicate hardware, restarting



instruction stream, when can you be interrupted, flushing pipelines, priority levels)

- Software issues: capturing, restarting code, latency, performance
- When in control can often cli/sti stop start interrupts to avoid code being interrupted. Why is this dangerous?
- Linux NAPI – switch to polling if interrupt rate too high



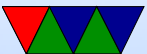
ARM Interrupts

- 7 types. Data Abort, Fast Interrupt Request, Interrupt Request, Prefetch Abort, Software Interrupt, Reset, Undefined Instruction
- ARM designed with fast interrupts in mind
- On interrupt: cpsr saved to specific spsr, pc saved to special lr, cpsr set to exception mode, pc points to address handler
- Vector table, holds instructions branched to on irq.



Usually a branch or move insn to an irq handler

- Priority mechanism, when happen at same time

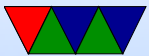


Interrupt Handler

- Disable interrupts
- Save register context
- Figure out which interrupt it was, call right handler
- ISR runs. ACKs the interrupt to hardware. Does minimal possible to handle interrupt. Returns.
- Restore context



- Re-enable interrupts



Interrupt Sources

- Data Abort – missing memory
- prefetch – trying to fetch next instruction
- swi – syscall
- undefined – emulate missing. why same priority swi and undefined? swi insn is never undefined



Register Contents

- r13 (sp) r14 (lr) r15 (pc) special cased, bank switched in
- return address. Either next insn, or current insn if has to be re-executed (a bad memory, redo now paged in)



Interrupt Controllers

- Interrupt controllers, map many interrupts to two irq lines
- irq – low priority, high latency – system timer
- firq – fast, dma transfers?



Interrupt Latency

- fastest – nested – ack right away (quiet the hardware) then re-enable so other interrupts not ignored
- prioritized – ignore interrupts of same or higher while servicing: higher priority end up w lower latency
- (NMI interrupts, x86)



Interrupt Stacks

- irq stacks
- separate from user stacks to avoid buggy code causing problems
- sp is one of banked regs



Vectors

From Table 9.2 of textbook

Exception	Mode	Vector Table Offset	Priority
Reset	SVC	0x00	1
Undefined Instruction	UND	0x04	6
Software Interrupt	SVC	0x08	6
Prefetch Abort	ABT	0x0c	5
Data Abort	ABT	0x10	2
n/a	–	0x14	–
IRQ	IRQ	0x18	4
FIQ	FIQ	0x1c	3

FIQ can immediately follow w no branch



Benefits of an OS

If you have an OS, no need to worry about most of this unless you are coding it up yourself.



DMA

- Direct memory access
- Devices can write directly to memory without going through the CPU.
Saves a load/store loop on the CPU.
- CPU sets up the transfer, then can do other things.
Often notified of completion by an interrupt



Device Firmware

- Devices are their own embedded systems these days. May even have full CPUs, etc.
- Need to run code. Firmware.
- In ROM? Or upgradable? Why might you want to upgrade? (bug fixes, economy, etc.)
- Talk about recent USB firmware malware

