ECE 471 – Embedded Systems
Lecture 11

Vince Weaver

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

`vincent.weaver@maine.edu`

8 October 2013

Announcements

- Homework #2. Due date extended until Thursday 5pm.
- Does everyone have Gumstix board working?
  Try to start on HW2b as soon as possible.
- Midterm is now scheduled for October 22nd.

Booting Linux

- Bootloader jumps into OS entry point
- Set Up Virtual Memory
- Setup Interrupts
- Detect Hardware / Install Device Drivers
- Mount filesystems
- Pass control to userspace / call init

- Run init scripts
- rc boot scripts, /etc/rc.local
  Start servers, or “daemons” as they’re called under Linux.
- fork()/exec(), run login, run shell

How a Program is Loaded on Linux

- Kernel Boots
- `init` started
- `init` calls `fork()`
- child calls `exec()`
- Kernel checks if valid ELF. Passes to loader
- Loader loads it. Clears out BSS. Sets up stack. Jumps

to entry address (specified by executable)

- Program runs until complete.
- Parent process returned to if waiting. Otherwise, init.

Viewing Processes

- You can use `top` to see what processes are currently running
- Also `ps` but that's a bit harder to use.

Context Switching

- OS provides the illusion of single-user system despite many processes running, by switching between them quickly.

- Switch rate in general 100Hz to 1000Hz, but can vary (and is configurable under Linux). Faster has high overhead but better responsiveness (guis, etc). Slower not good for interactive workloads but better for long-running batch jobs.

- You need to save register state. Can be slow, especially with lots of registers.
- When does context switch happen? Periodic timer interrupt. Certain syscalls (yield, sleep) when a process gives up its timeslice. When waiting on I/O
- Who decided who gets to run next? The scheduler.
- The scheduler is complex.
- Fair scheduling? If two users each have a process, who runs when? If one has 99 and one has 1, which runs

next?

- Various O() ratios for the schedulers

Device Drivers

- Why are device drivers useful?
- Abstraction: no need to know details for each device/machine. For example, you can write code that will talk to an i2c device and it will work on any Linux device with i2c. The various drivers handle setting things up, initialization, detection, etc.
- Permissions – keep users from accidentally (or on purpose) accessing devices they shouldn't

What’s a Device

- Hardware interfaced to a CPU
- Connected, either in address space or over a bus
  I/O ports or memory
- Communicate either by write to address, or in/out

Device Firmware

- Early hardware was just bits hung off a bus
- Hardware has gotten smarter, many devices include CPUs, became embedded systems themselves
- Firmware sometimes has to be uploaded before hardware will work. Why? To fix bugs in software. To make things configurable. Mostly economics (no need to flash chip, can fix bugs w/o re-flashing)

Device Communication

- Interrupts
- I/O
- DMA