ECE 471 – Embedded Systems
Lecture 12

Vince Weaver

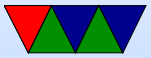
http://www.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

13 October 2016

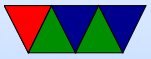
Announcements

- Homework #5 was due
- Homework #6 will be posted today
- Midterms will be graded for next class
- Memsys update.
- UM Talk.
- In computer engineering typically you come up with some



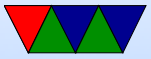
theoretical idea, then come up with contrived theoretical numbers that back it up. Doing *actual* measurements is considered something novel somehow (by some people, others reject it as too boring).

- If you are a Junior and are interested in doing power measurement research over the summer, look into applying CUGR (now) or REU (March)

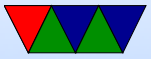


Homework 6 notes

- Handout should cover most of it
- bit-banging i2c
- Use the sysfs gpio interface and driving the SDA and SCL lines manually to talk to the 4x7 LED display
- Still easier than full bitbang, where you'd have to write to various i/o addresses
- A lot of the code is provided for you, follow the directions
- How do you set SDA low?
  Set to output, write a '0'



- How do you set SDA high?
  Do not write a '1'!
  Open collector, need to let it float.
  Set to 'input' works.
- Static in C?
- Why not bitbang everything? A pain. Hardware does it for you. Hardware even does more, can often buffer or DMA, timing more exact.
- Why might you want to bitbang i2c? Only have one i2c bus? Or no i2c bus, only GPIOs? kernel has bitbang driver

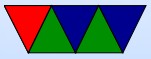


Device Detection

- 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.

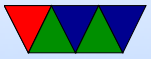
- Old ARM, hard-coded. So a rasp-pi kernel only could boot on Rasp-pi. Lots of pound-defined and hard-coded hw info.

- 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.

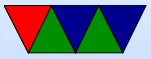
- Does mean that updating to a new kernel can be a pain.



Detecting Devices

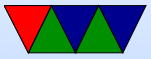
There are many ways to detect devices

- 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



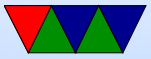
it is located

- 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



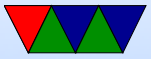
Devicetree

- 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



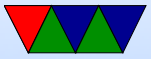
to describe all the hardware available. Thus kernel much more generic

- Still working on issues with this.



Device Firmware

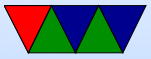
- Devices are their own embedded systems these days. May even have full CPUs, etc.
- Need to run code. Firmware.
- In ROM? Or upgradable? Why might you want to upgrade? (bug fixes, economy, etc.)
- Talk about recent USB firmware malware



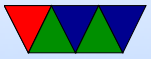
Firmware

Provides booting, configuration/setup, sometimes provides rudimentary hardware access routines.
Kernel developers like to complain about firmware authors.
Often mysterious bugs, only tested under Windows, etc.

Old days things were simple, just external ROM and CPU jump to the address.
Old old days (Altair, etc) no ROM, toggled bootloader via switches

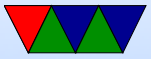


- BIOS – legacy 16-bit interface on x86 machines
- UEFI – Unified Extensible Firmware Interface
  ia64, x86, ARM. From Intel. Replaces BIOS
- OpenFirmware – old macs, SPARC
- LinuxBIOS



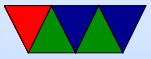
Bootloaders on ARM

- uBoot – Universal Bootloader, for ARM and under embedded systems
- So both BIOS and bootloader like minimal OSes



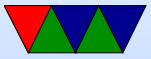
Raspberry Pi Booting

- Unusual
- Small amount of firmware on SoC
- ARM 1176 brought up inactive (in reset)
- Videocore loads first stage from ROM
- This reads `bootcode.bin` from fat partition on SD card into L2 cache. It's actually a RTOS (real time OS in own right "ThreadX")



- This runs on videocard, enables SDRAM, then loads `start.elf`

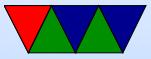
- This initializes things, the loads and boots Linux `kernel.img`. (also reads some config files there first)



More booting

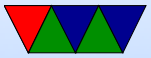
- Most other ARM devices, ARM chip runs first-stage boot loader (often MLO) and second-stage (uboot)

- FAT partition
  Why FAT? (Simple, Low-memory, Works on most machines, In theory no patents despite MS's best attempts (see exfat))
  The boot firmware (burned into the CPU) is smart enough to mount a FAT partition

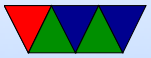


Booting Linux

- 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

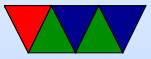


- Run init scripts
- rc boot scripts, /etc/rc.local
  Start servers, or “daemons” as they’re called under Linux.
- fork()/exec(), run login, run shell



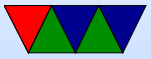
How a Program is Loaded on Linux

- 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.