# ECE 471 – Embedded Systems
# Lecture 15

Vince Weaver

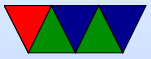
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

2 October 2017

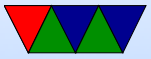
# Announcements

- Midterm is Thursday.



# Homework #4 Review

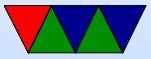
- Blink OK – only issue write after close.
- Questions
  - 5.a Why usleep? Not udelay, how come no one caught this past three years? Less resources (not busy sleeping), cross-platform (not speed-of-machine-dependent), compiler won't remove, other things can run, power saving
  - 5.b Layer of abstraction. In this case, not having to bitbang the interface or know low-level addresses,



portability among machines. Note: You can use high-level languages w/o an OS.

- 5.c Limitations : higher overhead, not all features exposed, uncertain timing.
  superuser permissions? when no OS you run everything as super user, though this depends on HW and is complicated.

- 5.d. Web browser part of OS? Microsoft law suit.
  Interesting comments on google/chrome
- 6.a Machines from dmesg: Pi2 (3) Pi3 (11) dmesg a



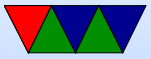
good place to find error messages, etc.

- 6.b Kernel versions. Current Linus kernel (upstream) is 4.13/4.14-rc3
  Uname syscall, what the parts mean

```
Linux rasp-pi 4.1.19+ #858 Tue Mar 15 15:52:03 GMT 2016 armv6l GNU
Linux orvavista 4.5.0-2-amd64 #1 SMP Debian 4.5.5-1 (2016-05-29) x
```

  2017: 4.1.9 (1) 4.9.35 (3) 4.9.41 (6) 4.4.38 (2)

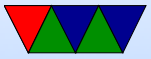
- 6.c. Disk space. Why `-h`? Human readable. what does that mean? Why is it not the default? At least Linux defaults to 1kB blocks (UNIX was 512) Lots of large disks.



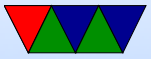
# Midterm Review

You can bring 1 page (8.5"x11") of notes if you want.

- Be sure you know the four characteristics of an embedded system, and can make an argument about whether a system is one or not.
  - Inside of something (embedded)
  - Fixed-purpose
  - Resource constrained
  - Real time constraints
- Benefits/downsides of using an operating system on an

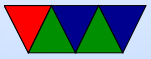


embedded device
  - Cost, time to market, helper libraries, overhead, timing
- ARM assembly language
  - Have you look at some assembly language code and know what it is doing
  - Only really need to know some of the more common instructions (add, cmp, mov, ldr, strb, swi). Also be aware of conditional execution.
- Code Density
  - Why is dense code good in embedded systems?
  - What changes were needed to ARM32 to make it fit

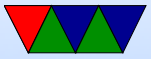


into 16-bit THUMB?

- GPIO & i2c
  - Know some of its limitations (speeds, length of wires, number of wires, etc)
  - Don't need to know the raw protocol
  - Know the Linux interface (open, ioctl, write) and be familiar with how those system calls work

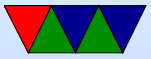


# Booting a System



# Firmware

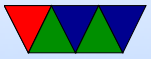
- What is firmware?



# 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.

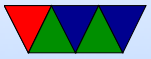
- BIOS – legacy 16-bit interface on x86 machines
- UEFI – Unified Extensible Firmware Interface
  ia64, x86, ARM. From Intel. Replaces BIOS
- OpenFirmware – old macs, SPARC
- LinuxBIOS



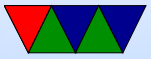
# Boot Methods

Firmware can be quite complex.

- Floppy
- Hard-drive (PATA/SATA/SCSI/RAID)
- CD/DVD
- USB
- Network (PXE/tftp)

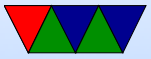


- Flash, SD card
- Tape
- Networked tape
- Paper tape? Front-panel switches?



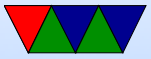
# Bootloaders on ARM

- 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)