ECE 471 – Embedded Systems
Lecture 16

Vince Weaver

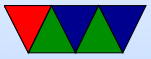
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

6 October 2021

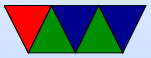
Announcements

- Don't forget HW#5

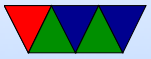


Homework #4 Error Checking

- What do you do if there's an error?
- Ignore it? Why could that be bad?
- Retry until it succeeds?
- Print an error message and continue?
  Can you continue?
  What if continuing with a bad file descriptor breaks things?
  What if printing too many error messages fills up a log, swamps the screen, hides other errors?

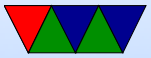


- Good error message
  Can't be confused with valid input (airlock)
  If displayed to user, make it easy to understand
- Print an error message and exit?
  What if it's a critical system?
- Crashing is almost never the right answer.

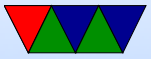


Homework #4 Permissions

- We haven't really discussed Linux permissions
- List file, "user" "group" "all"
- `drwxr-xr-x`
- Often in octal, 777 means everyone access
- Devices under /dev or /sysfs might be set to only root or superuser
- Traditional UNIX /dev you can set with chown (to set user/group) or chmod (to set permissions)
- Group under /etc/group, so gpio group



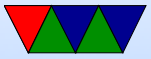
- Having to manually set permissions a pain. Program called udev that does it automatically when a device driver is configured. It might take a few ms to notice
- Why is it better than using "sudo"? Why might I as grader not want to run your code using "sudo" if I can avoid it?
- How to set up sudo? /etc/sudoers file



Homework #4 – LED Blinking

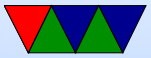
- Blink frequency. Remember, 1Hz is 500ms on / 500ms off

  not 500us, not 1s
- Blink correct GPIO. Does it matter? Want to fire engines, not engage self destruct.



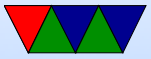
Homework #4 – Switch

- Debouncing
  - 100ms or even 10ms is long time
  - Tricky as we are detecting levels not edges here
  - Reading and only reporting if you say have 3 in a row of save val
  - Reading, sleeping a bit, then report the value after has settled
  - Just sleeping a long time after any change? If a short glitch happens this might misreport.



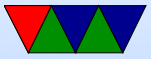
- Sleep too long, might miss events
- Debounce if using interrupt-driven code

  In that case debouncing might be to ignore repeated changes if they happen too close together



Homework #4 Question

- 5.a Why usleep? Less resources (not busy sleeping), cross-platform (not speed-of-machine-dependent), compiler won't remove, other things can run, power saving.
  Be careful saying accuracy! usleep() guarantees a minimum time delay, but it is best effort how long the delay actually is. So if you really need *exact* time delays you probably want some other interface.
- 5.b Layer of abstraction. In this case, not having

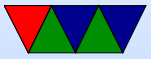


to bitbang the interface or know low-level addresses, portability among machines.
ability to run WiringPi is not a benefit

- 6.a Machines from dmesg: 2021: Pi4 (12) Pi3B+ (5) (alphabetical) dmesg a good place to find error messages, etc. grep
- 6.b Kernel versions. Current Linus kernel (upstream) is 5.14 /5.15-rc4
Uname syscall, what the parts mean

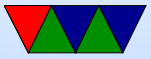
```
Linux linpack-test 4.14.50-v7+ #1122 SMP Tue Jun 19 12:26:26 BST 2018 armv7l GNU/Linux\\
Linux orvavista 4.5.0-2-amd64 #1 SMP Debian 4.5.5-1 (2016-05-29) x86_64 GNU/Linux\\
```

2021: 5.10.11 (5) 5.10.17 (5) 5.10.60 (4) 5.4.51 (1)



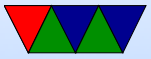
4.19.97 (1) 4.9.80 (1)

- 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.



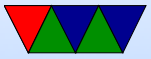
Trusted Firmware

- Firmware can be dangerous: runs below/outside of the Operating System, doesn't matter how secure your OS is if firmware compromised
- Can you trust your firmware to be not-evil?
- Evil Maid problem – what if someone breaks into your hotel room and replaces your firmware – could you tell?
- Best you can do is trust it to be the same firmware released by your vendor (you still have to trust them)
- Use cryptographic signing. Hardware will only run code

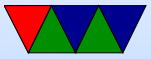


"signed" by a trusted entity.

- A signed firmware can run a signed bootloader which can run a signed operating system which can run signed apps
- Downside: no longer general purpose, average person cannot run code they wrote unless they can get it signed
- Code still has to be well written. "jailbreaks" on phones and video game consoles are due to trusted code having bugs and then jumping into unsigned code.
- Will you still be able to run Linux?
  Trust Microsoft to keep signing bootloader for us?

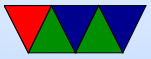


- Also the Apple / EPIC thing



Trusted Firmware

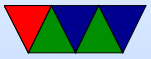
- What is the Pi GPU doing?
- What about the T2 processor on macs?
- New for ARMv8: ARM Trusted Firmware (ATF). Two standards, vendors have possibly made a mess of it already.
- Other platforms have it too. DRM to keep you from copying movies or video games.
- Windows 11 requiring TPM2 module



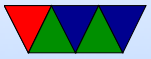
Boot Methods

Firmware can be quite complex.

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

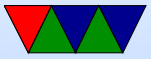


- Flash, SD card
- Tape
- Networked tape
- Paper tape? Front-panel switches?

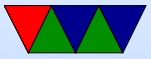


Disk Partitions

- Way to virtually split up disk.
- DOS GPT – old partition type, in MBR. Start/stop sectors, type
- Types: Linux, swap, DOS, etc
- GPT had 4 primary and then more secondary
- Lots of different schemes (each OS has own, Linux supports many). UEFI more flexible, greater than 2TB
- Why partition disks?
  - Different filesystems; bootloader can only read FAT?



- Dual/Triple boot (multiple operating systems)
- Old: filesystems can't handle disk size

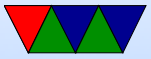


Device Detection

- 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.

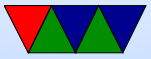
- Old ARM, hard-coded. So a rasp-pi kernel only could boot on Rasp-pi. Lots of pound-defined and hard-coded hw info.

- 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.

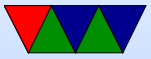
- Does mean that updating to a new kernel can be a pain.



Detecting Devices

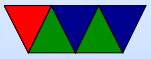
There are many ways to detect devices

- 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



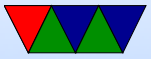
it is located

- 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.