ECE 471 – Embedded Systems
Lecture 16

Vince Weaver

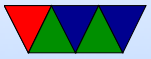
https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

4 October 2023

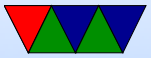
Announcements

- Midterm on 13th, more on that Friday (and Wednesday)
- Don't forget HW#5
- Don't rush to hand back in the i2c displays, you'll need them for HW#9

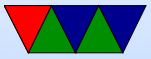


Homework #4 Error Checking

- 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?

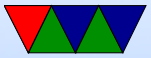


- Good error message
  Can't be confused with valid input (airlock)
  If displayed to user, make it easy to understand
- Print an error message and exit?
  What if it's a critical system?
- Crashing is almost never the right answer.
- Can get more info on error with errno / strerror()

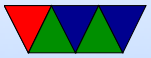


Homework #4 Permissions

- 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



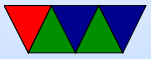
- Why is it better than using "sudo"? Why might I as grader not want to run your code using "sudo" if I can avoid it?
- How to set up sudo? /etc/sudoers file



Homework #4 – LED Blinking

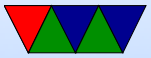
- Blink frequency. Remember, 1Hz is 500ms on / 500ms off

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



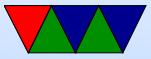
Homework #4 – Switch

- 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.



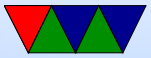
- Sleep too long, might miss events
- Debounce if using interrupt-driven code

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



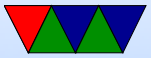
Homework #4 – Something Cool

- How can you read/write at same time (say to let switch activate LED)
- Need to make copy of data structures
- If you do re-use, make sure you close(), especially if you open multiple times. Either will get EBUSY or else fd leak



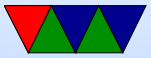
Homework #4 Question – usleep()

- 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.



Homework #4 Question – OS

- provides layer of abstraction
- In this case, not having to bitbang the interface or know low-level addresses, portability among machines.

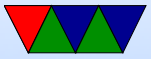


Homework #4 Question – Linux stuff

- 6.a Machines from dmesg: 2022: Pi4 (20) Pi3B+ (2) Pi3 (8) dmesg a good place to find error messages, etc. grep
- 6.b Kernel versions. Current Linus kernel (upstream) is 6.0
  Uname syscall, what the parts mean

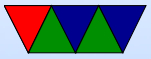
```
Linux linpack-test 4.14.50-v7+ #1122 SMP Tue Jun 19 12:26:26 BST 2018 armv7l GNU/Linux\\
Linux orvavista 4.5.0-2-amd64 #1 SMP Debian 4.5.5-1 (2016-05-29) x86_64 GNU/Linux\\
```

2022: 5.15.61 (12) 5.15.56 (2) 5.15.30 (1) 5.15.32 (3) 5.15.0 (1) (ubuntu?) 5.10.103 (4) 5.10.92 (1) 5.10.63



(1) 5.10.60 (1) 5.4.51 (3) 4.9.80 (1)

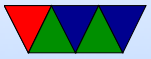
- 6.c. Disk space. Why `-h`? Human readable. what does that mean? Why is it not the default? At least Linux defaults to 1kB blocks (UNIX was 512) Lots of large disks.



Real Time Constraints

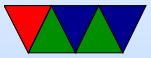
What are real time constraints?

- Time deadlines that hardware needs to respond in.
- Goal not performance, but response time
- Deadlines are often short (order of milliseconds or microseconds)



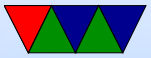
Real-time example

- Self-driving car driving 65mph
  - That's roughly 95 feet/s
  - That's roughly 10 feet / 100ms
  - Stopping distance is 180 feet
- Equipped with image processor: GPU and camera.
  - Camera 60fps, 16ms.
  - GPU code recognize a deer within 100ms.
- Specification: if something jumps out, can stop within 200 feet.



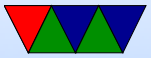
Can we meet that deadline? Yes. What if an interrupt happens for 100ms in the middle? We miss deadline?

- Another example, turn in the road. How long does it take to notice, make turn? What if there's a delay?
- What if wiggly road, and you consistently miss by 100ms? Over-compensate?



Types of Real Time Constraints

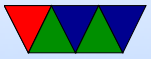
- Hard – miss deadline, total failure of system. Minor or major disaster (people may die?)
  Antilock brakes?
- Firm – result no longer useful after deadline missed
  small number of misses might be acceptable
  lost frames in video, missed frames in video game
- Soft – results gradually less useful as deadline passes.
  Caps lock LED coming on?



Uses of Real Time

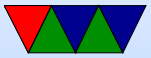
Who uses realtime?

- Timing critical situations. Cars, medical equipment, space probes, etc.
- Industrial automation. SCADA. Stuxnet.
- Musicians, important to have low-latency when recording
- High-speed trading



Why isn't everything Real-time?

- It's hard to do right
- It's expensive to do right
- It might take a lot of testing
- It's usually not necessary

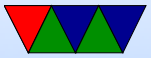


Constraints depend on the Application

Try not to over-think things.

Can almost always come up with a scenario where a soft constraint could become hard.

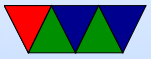
For example: Unlocking a car door taking an extra second? Not hard real-time, except maybe if your car is about to crash and you need to escape quickly.



Deviations from Real Time

Sometimes referred to as “Jitter”

- On an ideal system the same code would take the same, predictable amount of time to run each time
- In real life (and moreso on high-end systems) the hardware and operating systems cannot do this
- Deviation (often some sort of random distribution) is jitter

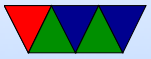


Hardware Sources of Jitter – Historical

- Typically Less jitter on older and simple hardware
- Old chips like 6502 – fixed clock, each instruction takes an exact number of cycles. Deterministic. With interrupts disabled you can perfectly predict how long code will take.

  Steve Wozniak famously wrote disk firmware on 6502 that more or less cycle-accurate bit-banged stepper motors.

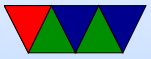
  Also video games, racing the beam.



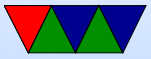
Hardware Sources of Jitter – Modern

Modern hardware more complex.
Tradeoff: systems faster on average but with hard to predict jitter

- Branch prediction / Speculative Execution
- Memory accesses unpredictable with caches – may take 2 cycles or 1000+ cycles (Memory Wall)
- Virtual Memory / Page faults
- Interrupts can take unknown amount of time
- Power-save may change clock frequency

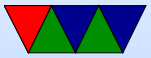


- Even in manuals instructions can take a range of cycles
- Slow/unpredictable hardware (hard disks, network access)
- Memory refresh (LPDDR burst refresh can avoid this a bit)



Software Sources of Jitter

- Interrupts. Taking too long to run; being disabled (cli)
- Operating system. Scheduler. Context-switching.
- Dynamic memory allocation, garbage collection.



Video game keyboard latency example

See Dan Luu's Paper "Computer Latency: 1977-2017"
`https://danluu.com/input-lag/`

- 1977 computers can have less latency to getting keypress on screen than fastest 2010s computers
- Having a fast processor only helps so much
- Slow hardware (keyboards, LCD displays), layers of abstraction in the way
- Apple II (1977) 30ms, modern machines 60-100+ms