# ECE 471 – Embedded Systems
# Lecture 15

Vince Weaver

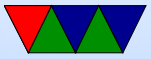
https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

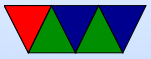
7 October 2024

# Announcements

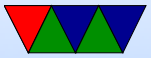
- Next week Fall Break, also cancel class for Career Fair on Wednesday 16th
- Midterm on 18th, more on that Friday
- Don't forget HW#5
- Don't rush to hand back in the i2c displays, you'll need them for HW#9
- Will post project document soon (advance note: project



needs to be different from ECE498 project)

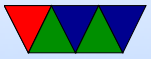


# Finish up some OS notes leftover from last week



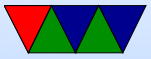
# What's included with an OS

- kernel / drivers (syscall barrier) – Linux definition
- also system libraries – Solaris definition
- low-level utils / software / GUI – Windows definition
  Web Browser included?
- Linux usually makes distinction between the OS Kernel and distribution. OSX/Windows usually doesn't.



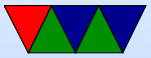
# Operating Systems Types

- Monolithic kernel – everything in one big address space. Something goes wrong, lose it all. Faster
- Microkernel – separate parts that communicate by message passing. can restart independently. Slower.
- Microkernels were supposed to take over the world. Didn't happen. (GNU Hurd?)
- Famous Torvalds (Linux) vs Tannenbaum (Minix) flamewar



# Common Desktop/Server Operating Systems

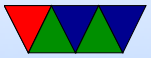
- UNIX derived
  - Linux (clone implemented from scratch)
  - FreeBSD / NetBSD / OpenBSD
  - MacOS (FreeBSD/Nextstep heritage)
  - Legacy (Irix/Solaris/AIX/etc.)
- Windows
- Obscure (BeOS/Haiku)



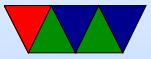
# Embedded Operating Systems

- Cellphone/Tablet
  - Android (Linux)
  - ChromeOS (Linux)
  - Apple iOS
  - Microsoft (WinCE/Mobile/Phone/RT/S/IoT (all these have been discontinued))

    In theory can install Windows 11 on a Raspberry Pi
- Networking
  - OpenWRT (Linux)



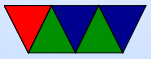
  - Cisco iOS
- Real Time OS
  - VXworks – realtime OS, used on many space probes
  - QNX – realtime microkernel UNIX-like OS, owned by Blackberry now
  - ThreadX – found in Pi GPU (owned by Microsoft now?) https://www.theregister.com/2023/11/28/microsoft_opens_sources_threadx/
  - FreeRTOS



# Embedded Linux Distributions

This list is horribly out of date.

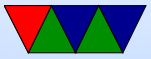
- linaro – consortium that work on ARM software
- openwrt – initially designed for wireless routers
- yocto – Linux Foundation sponsored embedded distro
- maemo – embedded distro originally by Nokia (obsolete)
- MeeGo – continuation of maemo, also obsolete
- Tizen – Follow up on MeeGo, by Samsung and Intel
- Ängstrom – Merger of various projects



# Real Time Constraints

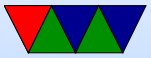
What are real time constraints?

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



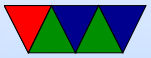
# Real-time example

- 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.



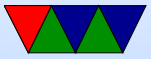
Can we meet that deadline? Yes. What if an interrupt happens for 100ms in the middle? We miss deadline?

- 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?



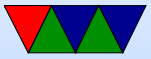
# Types of Real Time Constraints

- 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?

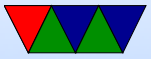


# Note on Soft Realtime

- Some people worry, what if it is something like "press a button, but takes minutes to run"
- If that's unacceptable, does that mean essentially all things are hard-real time?
- Part of it comes down to how likely the failure is. If with minimal programming effort the response is reasonable 99.9% of the time, and any consequence of missing is minor, than soft real time
- If something consistently goes wrong and it takes minutes



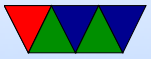
for each button press and that's unacceptable, then yes, maybe what you have is more than soft... but maybe also you need to rethink your hardware/software design



# 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

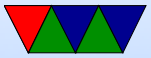


# Constraints depend on the Application

Try not to over-think things.

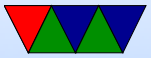
Can almost always come up with a scenario where a soft constraint could become hard.

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.



# Why isn't everything Real-time?

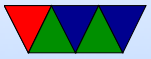
- It's hard to do right
- It's expensive to do right
- It might take a lot of testing
- It's usually not necessary



# 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

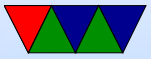


# Hardware Sources of Jitter – Historical

- 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.

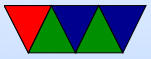
  Also video games, racing the beam.



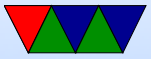
# Hardware Sources of Jitter – Modern

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.