ECE 471 – Embedded Systems
Lecture 18

Vince Weaver

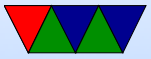
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

13 October 2017

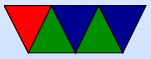
Announcements

- Any problems with HW#5
- HW#6 will be posted

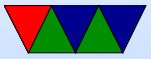


Homework 6 notes

- 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'



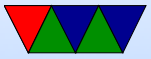
- How do you set SDA high?
  Do not write a '1'!
  Open collector, need to let it float.
  Set to 'input' works.
- Static in C?
- Why not bitbang everything? A pain. Hardware does it for you. Hardware even does more, can often buffer or DMA, timing more exact.
- Why might you want to bitbang i2c? Only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



Real Time OS

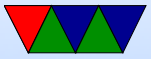
Who uses realtime?

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



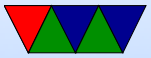
Real-time example

- 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?

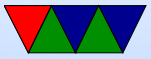


Worst Case Behavior – Hardware

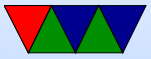
- Easier on older and simple hardware
- Old chips like 6502 – fixed clock, each instruction takes an exact number of cycles. Deterministic. With interrupts disabled you can perfectly predict how long code will take.

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

  Also video games, racing the beam.



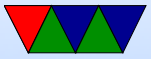
- Modern hardware more complex:
  - Memory accesses unpredictable with caches – may take 2 cycles or 1000 cycles
  - Interrupts can take unknown amount of time
  - Branch prediction
  - Power-save may change clock frequency
  - Even in manuals instructions can take a range of cycles



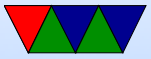
Software Worst Case – Context Switching

- OS provides the illusion of single-user system despite many processes running, by switching between them quickly.

- Switch rate in general 100Hz to 1000Hz, but can vary (and is configurable under Linux). Faster has high overhead but better responsiveness (guis, etc). Slower not good for interactive workloads but better for long-running batch jobs.

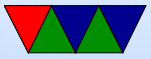


- You need to save register state. Can be slow, especially with lots of registers.
- When does context switch happen? Periodic timer interrupt. Certain syscalls (yield, sleep) when a process gives up its timeslice. When waiting on I/O
- Who decided who gets to run next? The scheduler.
- The scheduler is complex.
- Fair scheduling? If two users each have a process, who runs when? If one has 99 and one has 1, which runs



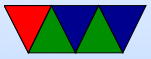
next?

- Linux scheduler was O(N). Then O(1). Now O(log N). Why not O(N^3)



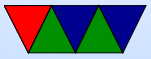
Common OS strategies

- Event driven – have priorities, highest priority pre-empts lower
- Time sharing – only switch at regular clock time, round-robin

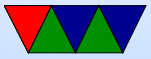


Scheduler example

- Static: Rate Monotonic Scheduling – shortest job goes first
- Dynamic: Earliest deadline first
- Three tasks come in. a. finish in 10s, 4 long. b. finish in 3, 2 long, c. finish in 5, 1 long
- In order they arrive, aaaabbccc bad for everyone
- RMS: cbbbaaaa works

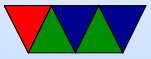


- EDF: bbbcaaaa also works.
- Lots of information on various scheduling algorithms



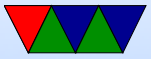
Priority Inversion / Locking

- When shared hardware/software and more than one thing might access at once
- Multicore: thread 1 read temperature, write to temperature variable
  thread 2 read temperature variable to write to display
  let's say it's writing 3 digit ASCII. Goes from 79 to 80. Will you always get 79 or 80? Can you get 70 or 89?
- How do you protect this? With a lock. Special data structure, allows only one access to piece of memory,



others have to wait.

- Can this happen on single core? Yes, what about interrupts.
- Implemented with special instructions, in assembly language
- Usually you will use a library, like pthreads
- mutex/spinlock



Priority Inversion Example

- Task priority 3 takes lock on some piece of hardware
- Task 2 fires up and pre-empts task 3
- Task 1 fires up and pre-empts task 1, but it needs same HW as task 3. Waits for it. It will never get free.
- Space probes have had issues due to this.