ECE 471 – Embedded Systems
Lecture 19

Vince Weaver

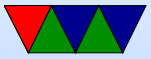
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

23 October 2019

Announcements

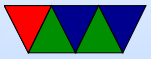
- Project handout posted to website
- Midterms still note quite graded



HW#6 Note

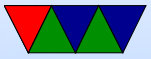
- How to take a byte and break into bits
- With C it's a matter of shifting and masking
- You can do something like the following (there are a lot of ways to do this in C)

```
unsigned char byte=0x5a;
for(x=0;x<8;x++) {
        if (byte&(1<<x)) {
                printf("1");
```



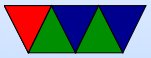
```
                }
                else {
                        printf("0");
                }
        }
```

- Note: for i2c we send the bits MSB (top bit) first so the code for that is going to look a bit different



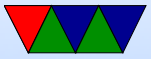
Project Preview

- The handout for this has been posted to the course website.
- Can work in groups
- Embedded system (any type, not just Pi)
- Written in any language (asm, C, python, C++, Java, etc.)
- Do some manner of input and some manner of output using the various capabilities we discussed
- I have a large amount of i2c, spi, and other devices that



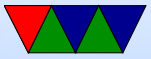
you can borrow if you want to try anything interesting.

- Past projects: games, robots, weather stations, motor controllers, music visualization, etc.
- Will be a final writeup, and then a short presentation and demo in front of the class during last week of classes.
- Can compliment another project, but must have some original code



Can you get Real-Time on Modern Systems?

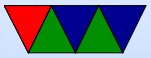
- Modern hardware does make it difficult with potentially unpredictable delay
- Some machines provide special, deterministic co-processors to help (PRUs on the beaglebone)
- You can still attempt to get real-time by coding your OS carefully



Real Time Operating Systems

How do RTOSes differ from regular OSes?

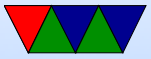
- Low-latency of OS calls (reduced jitter)
- Fast/Advanced Context switching (especially the scheduler used to pick which jobs to run)
- Often some sort of job priority mechanism that allows high-importance tasks to run first



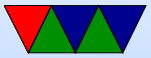
Software Worst Case – Context Switching

- 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.

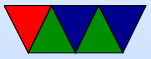


- You need to save register state. Can be slow, especially with lots of registers.
- When does context switch happen? Periodic timer interrupt. Certain syscalls (yield, sleep) when a process gives up its timeslice. When waiting on I/O
- Who decided who gets to run next? The scheduler.
- The scheduler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



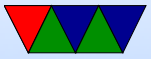
next?

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



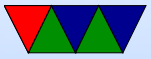
Common OS scheduling strategies

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

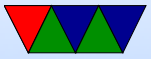


Scheduler example

- 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

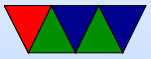


- EDF: bbbcaaaa also works.
- Lots of information on various scheduling algorithms



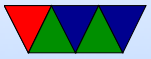
Locking

- 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,



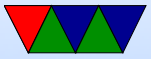
others have to wait.

- 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
- Atomicity



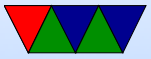
Priority Inversion Example

- Task priority 3 takes lock on some piece of hardware (camera for picture)
- 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. (camera for navigation?)
- Space probes have had issues due to this.



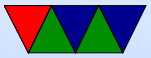
Real Time Operating System

- Can it be hard real time?
- Simple ones can be mathematically provable
- Otherwise, it's a best effort



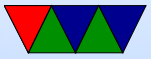
Priority Based, like Vxworks

- Each task has priority 0 (high) to 255 (low)
- When task launched, highest priority gets to run
- Other tasks only get to run when higher is finished or yields
- What if multiple of same priority? Then go round-robin or similar



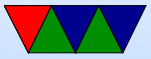
Is Regular Linux a RTOS

- Not really
- Can do priorities ("nice") but the default ones are not RT.



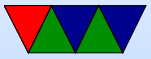
Real Time Linux

- Project to have a small supervisor RTOS and run Linux as a process
- Code that needed a compatible OS interface could call into this process-Linux, but it could always be pre-empted
- Not supported anymore?



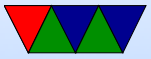
PREEMPT Kernel

- Linux PREEMPT_RT
- Faster response times
- Remove all unbounded latencies
- Change locks and interrupt threads to be pre-emptible
- Have been gradually merging changes upstream

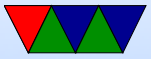


Typical kernel, when can you pre-empt

- When user code running
- When a system call or interrupt happens
- When kernel code blocks on mutex (lock) or voluntarily yields
- If a high priority task wants to run, and the kernel is running, it might be hundreds of milliseconds before you get to run

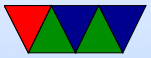


- Pre-empt patch makes it so almost any part of kernel can be stopped (pre-empted). Also moves interrupt routines into pre-emptible kernel threads.

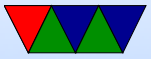


Linux PREEMPT Kernel

- What latencies can you get?
  10-30us on some x86 machines
- Depends on firmware; SMI interrupts (secret system mode, can't be blocked, emulate USB, etc.)
  Slow hardware; CPU frequency scaling; nohz
- Special patches, recompile kernel
- Priorities
  - Linux Nice: -20 to 19 (lowest), use nice command
  - Real Time: 0 to 99 (highest)

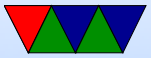


- Appears in ps as 0 to 139?



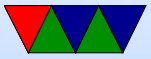
Changes to your code

- What do you do about unknown memory latency?
  - mlockall() memory in, start threads and touch at beginning, avoid all causes of pagefaults.
- What do you do about priority?
  - Use POSIX interfaces, no real changes needed in code, just set higher priority
  - See the `chrt` tool to set priorities.
- What do you do about interrupts?
  - See next



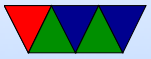
Interrupts

- Why are interrupts slow?
- Shared lines, have to run all handlers
- When can they not be pre-empted? IRQ disabled? If a driver really wanted to pause 1ms for hardware to be ready, would often turn off IRQ and spin rather than sleep
- Higher priority IRQs? FIR on ARM?
- Top Halves / Bottom Halves
- Unrelated, but hi-res timers



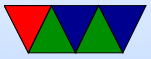
Co-operative real-time Linux

- Xenomai
- Linux run as side process, sort of like hypervisor

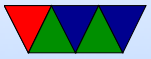


Non-Linux RTOSes

- Interesting reference: `https://rtos.com/rtos/`
- Often are much simpler than Linux
- Some only need a few kilobytes of overhead
- Really, just replacements for an open-coded main loop that did a few tasks sequentially. (Effectively round-robin). Can possibly get better response if you multitask.
- Provide fast context-switching, interrupt handling,



process priority (scheduling), and various locking/mutex libraries



List of some RTOSes

- Vxworks
- Neutrino
- Free RTOS
- Windows CE
- MongooseOS (recent LWN article?)
- ThreadX (in the Pi GPU)