ECE 471 – Embedded Systems
Lecture 19

Vince Weaver

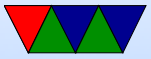
https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

16 October 2023

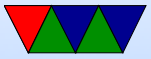
Announcements

- No class Wednesday due to Engineering job fair (will send e-mail)
- Don't forget RT homework HW#6
- HW#6 comment: the 3rd/4th experiment might slow down your system to be unusable if logged in at the GUI. If this happens and you can't complete the question, just mention that and complete things the best you can.

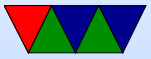


Previous Homework Note – Breaking the Build

- I do take off points if your code won't compile
- In real world if you check in code that breaks the build it will annoy a lot of people (some companies will give out trophies to try to shame you)
- Code that won't compile can be more annoying than buggy code, as no one else can work on the project until it is fixed
- Linux kernel developers don't like it because it implies

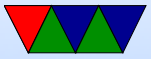


the code you sent them was never tested (because, how could you test it if doesn't build)



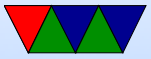
Common OS scheduling strategies

- Event driven – have priorities, highest priority pre-empts lower
  Usually can "yield" rest of your timeslice
- Time sharing – only switch at regular clock time, round-robin



Scheduler Types

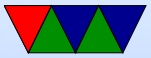
- There is a large body of work on scheduling algorithms.
- Assume you tell it to run tasks, they are put into queue
- How should they be run? A few (not exhaustive) possibilities:
  - Simple: In order the jobs arrive
  - Static: (RMS) Rate Monotonic Scheduling – shortest first
  - Dynamic: (EDF) Earliest deadline first



Deadline Scheduler Example

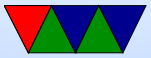
- Three tasks come in
  - A: deadline: finish by 10s, takes 4s to run
  - B: deadline: finish by 3s, takes 2s to run
  - C: deadline: finish by 5s, takes 1s to run
- Can they meet the deadline?

In-order	A	A	A	A	B	B	C	-	-	-
RMS	C	B	B	A	A	A	A	-	-	-
EDF	B	B	C	A	A	A	A	-	-	-



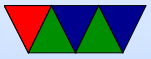
Where do the deadlines come from?

- Often from hardware, or from the spec
- Can you change them? Maybe, but might involve hardware changes
- Examples
  - Write to device, need to do all 4 memory accesses within 100us
  - Car notices brakes locking, start pumping them within 100ms
  - Drop laptop, notice and park hard-drive within 100ms

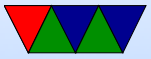


Priority Based Scheduling

- It's actually rare for an OS to let you specify a deadline
- Usually instead they are priority based (like in HW#6)
  - Have multiple tasks running, assign priority
  - In previous example, B highest, then C, then A
  - B can pre-empt C and A
- What can happen if overcommit resources? Starvation

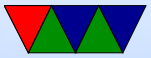


IRQ	-	-	-	-	-	I	-	-	-	-
HIGH	-	-	-	-	B	-	B	-	-	-
MEDIUM	-	-	C	-	-	-	-	-	-	-
LOW	A	A	-	A	-	-	-	A	-	-
OS	!	!	!	!	!	!	!	!	!	!



Priority Inversion

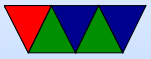
- Task priority 3 takes lock on some piece of hardware (camera for picture)
- Task 2 fires up and pre-empts task 3
- Task 1 fires up and pre-empts task 1, but it needs same HW as task 3. Waits for it. It will never get free. (camera for navigation?)
- Space probes have had issues due to this.



Real Time without an O/S

Often an event loop. All parts have to be written so deadlines can be met. This means all tasks must carefully be written to not take too long, this can be extra work if one of the tasks is low-priority/not important

```
main() {
    while(1) {
        do_task1(); // read sensor
        do_task2(); // react to sensor
        do_task3(); // update GUI (low priority)
    }
}
```



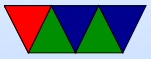
Real Time with an O/S

What if instead you ran all three at once, and let OS switch between them

```
while(1) {
  do_task1();
}
```

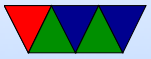
```
while(1) {
  do_task_2();
}
```

```
while(1) {
   do_task3();
}
```



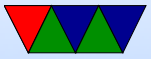
Bare Metal

- What if want priorities?
- Have GUI always run, have the other things happen in timer interrupt handler?
- What if you have multiple hardware all trying to use interrupts (network, serial port, etc)
- At some point it's easier to let an OS handle the hard stuff



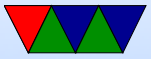
Real Time Operating System

- Can provide multi-tasking/context-switching
- Can provide priority-based scheduling
- Can provide low-overhead interrupts
- Can provide locking primitives



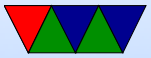
Hard Real Time Operating System

- Can it be hard real time?
- Is it just some switch you can throw? (No)
- Simple ones can be mathematically provable
- Otherwise, it's a best effort



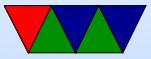
Priority Based, like Vxworks

- 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

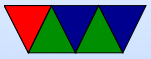


Free RTOS

- Useful article series about this: https://www.digikey.com/en/maker/projects/what-is-a-realtime-operating-system-rtos/28d8087f53844decafa5000d89608016
- Footprint as low as 9K
- Pre-emptive or co-op multitasking
- Regularly scheduled tasks (vTaskDelayUntil(), i.e. blink LED every 10 ticks)
- Task priority
- Semaphores/Mutexes
- Timers

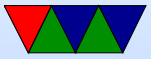


- Stack overflow protection
- Inter-process communication (queues, etc)
- Power management support
- Interrupts (interrupt priority)



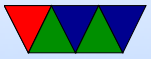
Is Regular Linux a RTOS

- Not really
- Can do priorities ("nice") but the default ones are not RT.
- Aside, "nice" comes from old UNIX multi-user days, when you could be nice and give your long-running jobs a low-priority so they wouldn't interfere with other people doing interactive tasks



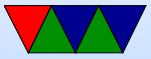
PREEMPT Kernel

- 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

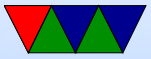


Typical kernel, when can you pre-empt

- 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

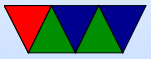


into pre-emptible kernel threads.



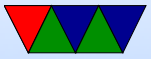
Linux PREEMPT Kernel

- 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



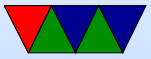
Linux Real Time Priorities

- Linux Nice: -20 to 19 (lowest), use nice command
- Real Time: 0 to 99 (highest)
- Appears in ps as 0 to 139?
- Can set with `chrt` command (see HW#6)



Co-operative real-time Linux

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



Next up is SPI

Start early on it as there's more than one lecture of material