

ECE 471 – Embedded Systems

Lecture 16

Vince Weaver

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

`vincent.weaver@maine.edu`

9 October 2026

Announcements

- Midterm on Friday, the 16th, in class
let me know if you'll be using extra-time accommodation in the classroom, as there is a class immediately following so I need to make sure I have a plan for that
- Am trying to grade all homeworks by then
- HW#6 will be posted, have two weeks
- Hang on to displays, needed for HW#9

Midterm Notes

- The midterm will be in-person during class time
- Closed book/notes but you are allowed one page (8.5"x11") full of notes if you want

Midterm Content – Characteristics

- Be sure you know the characteristics of an embedded system, and can make an argument about whether a system is one or not.
 - Inside of something (embedded)
 - Fixed-purpose
 - Resource constrained
 - Sensor I/O
 - Real time constraints (if you use this, be sure you can explain)

Midterm Content – Operating Systems

- Benefits/downsides of using an operating system on an embedded device
 - Benefits: “Layer of Abstraction”
 - Downsides: overhead, timing

Midterm Content – C Code

- C code
 - Have you look at some code and know what it is doing
 - Fill in missing comments
 - Look at code and find bugs
 - Mostly know what file I/O, loops, usleep, open/ioctl (things we've done in the homeworks)

Midterm Content – Code Density

- Why is dense code good in embedded systems?
- Know why ARM introduced THUMB/THUMB2

Midterm Content – GPIO & i2c

- Know some of its limitations (speeds, length of wires, number of wires, etc)
- Don't need to know the raw protocol
- Know the Linux interface (open, ioctl, write) and be familiar with how those system calls work

Midterm Content – Realtime

- Realtime won't be on this midterm
- You should still know about it if you plan on using it as one of the characteristics of an embedded system

Project Preview

- Posted a PDF with full details to the website
- Can work in groups
- Embedded system (any type, not just Pi)
Pi Pico, Beagleboard, Orange Pi, 271 STM boards, TS-7600, etc.
- Written in any language (asm, C, python, C++, Java, Rust, etc.)
- Do some manner of input and some manner of output using the various capabilities we discussed

Project Parts / Topics

- 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.
- Note: this will have to be distinct in some way from senior project or work from other classes

Project: Writeup/Presentation

- Will be a final writeup, and then a short minute presentation and demo in front of the class during last week of classes.
- Deadlines:
 - November 6: pick topic (send e-mail with group members and preliminary topic)
 - November 24: progress report (e-mail with summary, how it's going, and which day you'd prefer to present)
 - Last 4 days of classes: project presentations
 - December 18: Writeup due

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.

More Context Switching

- 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.

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 $O(N^3)$
- BPF scheduler? LAVD? EEVDF?
<https://lwn.net/Articles/1097209/>

Operating System Scheduling

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

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	-	-	-

Bonus Material – Real time on 8-bit Platforms

Showed a Real-Time Example

- Apple II computer, has 1MHz 6502 processor
- Two common graphics modes
 - 40x48 in 15 colors (1k)
 - 280x192 in 6 colors (8k)
- Each mode has two “pages” you can select between by writing a soft-switch

Mid-screen Mode Switch

- The screen updates at 60Hz
- Unlike other 8-bit machines could not easily find out when VBLANK is
- There's a hack using the "floating bus" called "vaporlock" by drawing patterns on the screen and reading an unused memory address can find out where you are
- You can then use cycle-counting to change graphics modes mid-screen

Relevance to Class

- The timing on this is exact, the whole effect only works if you are cycle-accurate
- If you are 1us off in your code timing the whole effect breaks
- Because interrupts are disabled and the 6502 is a deterministic chip, by carefully selecting assembly code you can have exact timings for when things happen down to that 1us deadline

Video

- The example in class didn't work well because I forgot the version of the MAME emulator in Debian Trixie is broken when switching page1 to page2 mid-screen
- You can see a video of what it should have looked like here:

<https://www.youtube.com/watch?v=Ifc1fCQrGSU>