

ECE 471 – Embedded Systems

Lecture 15

Vince Weaver

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

`vincent.weaver@maine.edu`

7 October 2026

Announcements

- Don't forget HW#5 (i2c)
Pick up a display if you haven't already
- Project document will be posted to website soon
- Midterm on the 16th, will review briefly Friday

Last Note on HW#3 – Code Comments

- Was mostly looking that you had them at all
- A lot of people the comment was something like “changed value” which is what you did, but that’s not what the code is actually doing and if you come back to the code in 5 years not very helpful
- In theory it might almost be OK if it’s in git and you have a better commit message about why it changed, but it’s best to have good comments too so you don’t have to poke around commit messages

Homework #3 Review – Exit in Assembly

- Exit – value is an integer which goes into r0 (x0 on 64-bit)
- Note it is an integer, not ASCII
- Be sure to comment your code
- Note: ABI says value going into r0 is first argument, (it's also the return value but that only matters after)
- 95% of students seem to be on 64-bit OS

Homework 3 – ARM32 vs THUMB2

- Be sure to specify base!
- Note it's bytes not bits. Also no need to divide by 4. Also each hex digit is a nibble
- ARM32 – $0x104a8 - 0x1043c = 0x6c = 108$ bytes
- Thumb2 – $0x10490 - 0x10438 = 0x58 = 88$ bytes
- Differences?
 - Thumb2 some instructions are 16-bit rather than 32-bit
 - Thumb2 different instructions (like movt/movw)
 - Thumb2 short instructions can be 2-argument

Homework 3 – Code Density

- You need to run `strip` on this to see it. Why?
Debug info, including extra thumb debug as well as the longer filename.
- You can use `readelf -a` and `readelf -s` to see the space the various segments take up.
Look at executables, **not** the C source code.

Homework 3 – Code Sizes

arch	unstripped	stripped
C arm32	9,896	5564
C thumb2	9,900	5564
asm arm32	1,308	536
C static	573,296	485,656
C ARM64	72,912	67,608

- You would think THUMB2 would be much smaller, but the assembler makes some poor decisions about wide/narrow instructions.
- Reference my LL work

Homework 3 – C Library Size

- C code is larger, but also remember to include the C library:

```
ls -lart /usr/lib/arm-linux-gnueabi/libc.a  
-rw-r--r-- 1 root root 3783784 Jun 17 09:00 /usr/lib/arm-linux-gnueabi/libc.a
```

- There are embedded C libraries, musl, newlib, uclibc, which are much smaller and often used in embedded systems.
- Smallest possible executable? I have written 128 Byte ones for competitions but you have to do sketchy things to the ELF file to be that small

Homework 3 – Linux Kernel Size

- The Linux Kernel is also pretty big these days:

```
ls -la /boot/vmlinuz-6.18.50+rpt-rpi-v8
-rw-r--r-- 1 root root 10192324 Sep 11 09:21 /boot/vmlinuz-6.18.50+rpt-rpi-v8
\item This is it while compressed, and it also isn't counting all
the device drivers in /lib/modules
```

Homework 3 – gdb

- crashes!
- have to use awful gdb interface
- line 9 is the crash
- the assembly is

```
ldrb r3,[r3]    // or ldrb w0,[x0]
```

load byte from the 32-bit address pointed to by r3,
store the resulting zero-extended byte into r3
(replacing the old value)

Homework 3 – gdb continued

- if you look at src code or info asm you can see it's dereferencing (following) a NULL (uninitialized) pointer, which is always a segfault on Linux
- Note in this particular case it's not an “off the end of the array” issue, but rather the array doesn't exist at all problem `array[0]=0` where array is NULL.
- Don't confuse NUL terminated strings with invalid NULL pointers

Homework 3 – Linux Tools – cal 9 1752

- Debian Linux dropped “cal” from the default install, it’s now in the “ncal” package
- cal missing days
- Julian to Gregorian calendar.
- People sad who paid weekly but paid rent monthly.
- George Washington’s b-day / Hunt for Red October
- Beware believing any page you google. Some urban legends / joke sites about this. If it were some sort of programmer bug it would have been fixed years ago.

Homework 3 – Calendar/Time Math Aside

- Calculating time/date is one of the hardest computer tasks you can do
- Calendar itself is tricky (calculating leap year)
- Time tricky too. Timezones? Leap-seconds?
- Things like daylight savings and timezones can change and require updates
- Bugs turn up any time something unusual happens with the time
- Ideally use a library when possibly to avoid having to open-code all of this

Brief Review of Jitter

Latency in Modern Systems

- Modern software stack has sources of latency
- Is this part of jitter?
 - Jitter is random latency variations
 - You can also have deterministic latency which can also get in the way of hitting deadlines

Video game keyboard latency example

See Dan Luu's Paper "Computer Latency: 1977-2017"
<https://danluu.com/input-lag/>

- 1977 computers can have less latency to getting keypress on screen than fastest 2010s computers
- Having a fast processor only helps so much
- Slow hardware (keyboards, LCD displays), layers of abstraction in the way
- Apple II (1977) 30ms, modern machines 60-100+ms

Latency of Apple II

- CPU running code reading memory access
Each CPU instruction handful of 1MHz cycles (few usec)
- Keypress happens, high bit set along with ASCII code, CPU reads in
- CPU writes out ASCII value to memory
- Video generator hardware running in parallel at 60 fps
- Electron beam scanning, reads out RAM, runs through decode ROM to get 7-bit pattern, writes to screen within one frame worst case

Latency of Modern System (part1)

- Press key, keyboard is own embedded system with CPU, scans keyboard, gets value, encodes it up as USB packet
- Sends out over USB bus (complex and with latency)
- USB controller gets packet, sends interrupt to CPU
- CPU gets interrupt, takes packet, notes it, returns from interrupt
- Later bottom half runs, decodes, to input subsystem,
- Operating system sees if anything is waiting for the input, if so it wakes it up (may take a bit if anything else running)

Latency of Modern System (part2)

- If it's a GUI, might have to run and see which window has focus, etc
- Program itself finally gets notified of keypress. `scanf()`.
- Immediately `printf()`, which calls back into the kernel with a syscall
- Kernel either prints it directly or else passes this to the program running, which then displays it
- If in GUI, Terminal emulator, update the graphics for the window (colors, font processing)

Latency of Modern System (part3)

- GUI compositor puts together screen, tells OS
- OS sends out over PCIe bus to GPU
- GPU runs shaders/whatever outputs to display via HDMI
- LCD display gets the data, decodes it to display it
- Display might buffer a few frames to do extra processing (turn this off with “gaming” mode)

Can you get Real-Time on Modern Systems?

- Small embedded systems w/o operating system easier
- Some will have small PIO (programmable I/O), essentially smaller embedded system you can offload important tasks to (Beaglebone, Pi Pico, Pi5)
- Code directly to hardware
- Turn off interrupts
- Turn off/avoid caches/speculation
- Load all of code into memory

What about on higher end systems?

- Modern hardware does make it difficult with potentially unpredictable delay
- Hard to program such machines w/o an operating system
- 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 and interrupts (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 – IRQ overhead

- OS like Linux will split interrupt handlers into top/bottom halves
- Top half will do the bare minimum: ACK the interrupt, make a note for the OS to handle the rest later, then immediately return. Tries to keep IRQ latency as small as possible.
- Bottom half at some later time when nothing else is going on the OS will carry out the work needed by the IRQ (handle a keypress, or a network packet, etc)