# ECE 471 – Embedded Systems
# Lecture 21

Vince Weaver

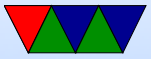
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

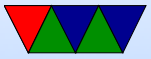
25 October 2021

# Announcements

- HW#7 was posted

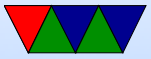


# HW#7 Followup



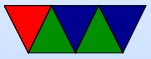
# SPI Sample Code Review

```
#define LENGTH 3
int result;
struct spi_ioc_transfer spi;
unsigned char data_out[LENGTH]={0x1,0x2,0x3};
unsigned char data_in[LENGTH];

/* kernel doesn't like it if stray values, even in padding */
memset(&spi,0,sizeof(struct spi_ioc_transfer));

/* Setup full-duplex transfer of 3 bytes */
spi.tx_buf    = (unsigned long)&data_out;
spi.rx_buf    = (unsigned long)&data_in;
spi.len       = LENGTH;
spi.delay_usecs       = 0 ;
spi.speed_hz          = 100000 ;
spi.bits_per_word     = 8 ; spi.cs_change         = 0 ;

/* Run one full-duplex transaction */
result = ioctl(spi_fd, SPI_IOC_MESSAGE(1), &spi) ;
```

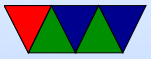


# Zeroed Structs in Kernel ABI

- Why is the kernel erroring out if the empty "pad" bit not zero?
  - Forward compatibility. You want to make sure that any empty bits stay that way.
  - If you want to add new functionality in the future you have to ensure reserved bits are all zero, otherwise old programs will do unexpected things (or break) if they had been accidentally setting those bits.



- So why were the pad bits non-zero? Bad luck. Local struct allocated on the stack, so if there were old values on the stack the pad value could be non-zero.

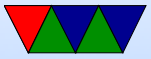


# Floating Point in C

- Converting int to floating point:

```
int value=45;
double temp;

temp=value;                 // works
temp=(float)value;          // casts make the conversion explicit
                            // but can potentially hide bugs
```

- `float` vs `double`
  float is 32-bit, double 64-bit
- Constants 9/5 vs 9.0/5.0
  The first is an integer so just "1". Second is expected

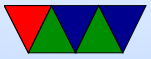


1.8.

- Printing. First prints a double. Second prints a double with only 2 digits after decimal.

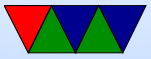
```
printf("%lf\n",temp);
printf("%.2lf\n",temp);
```

- Converting float/double to integer
  - floor()
  - ceil()
  - round()
  - rint()
  - naerbyint()



# Can you get Real-Time on Modern Systems?

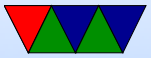
- Modern hardware does make it difficult with potentially unpredictable delay
- Some machines provide special, deterministic co-processors to help (PRUs on the beaglebone)
- You can still attempt to get real-time by coding your OS carefully



# Real Time Operating Systems

How do RTOSes differ from regular OSes?

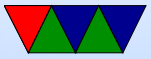
- Low-latency of OS calls (reduced jitter)
- Fast/Advanced Context switching (especially the scheduler used to pick which jobs to run)
- Often some sort of job priority mechanism that allows high-importance tasks to run first



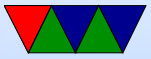
# Software Worst Case – Context Switching

- 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.

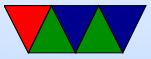


- You need to save register state. Can be slow, especially with lots of registers.
- When does context switch happen? Periodic timer interrupt. Certain syscalls (yield, sleep) when a process gives up its timeslice. When waiting on I/O
- Who decided who gets to run next? The scheduler.
- The scheduler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



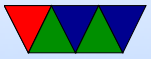
next?

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



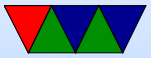
# Common OS scheduling strategies

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



# Scheduler example

- Simple: In order the jobs arrive
- Static: (RMS) Rate Monotonic Scheduling – shortest first
- Dynamic: (EDF) Earliest deadline first
- Three tasks come in
  - A: finish in 10s, 4s long
  - B: finish in 3s, 2s long
  - C: finish in 5s, 1s long
- In order they arrive, **aaaabbccc** bad for everyone



- RMS: **cbbbaaaa** works
- EDF: **bbbcaaaa** also works
- Lots of information on various scheduling algorithms