ROSE-HULMAN INSTITUTE OF TECHNOLOGY

To: Dr. Mark Yoder

From: Michael McDonald

Subject: Problem Set 2 (8, 10, 10a, 11) Memo

Date: September 16, 2013

---

1 Weekly Summary

This week was devoted to GPIO, both using the Linux file system as well as using C to access the files. Most of the exercises centered around the idea of reading from files to gather input and then writing to files to produce output. There was also a little in exercises 8 and 8a on the kernel and cross compiling for ARMs. All source code is stored on my git repo.

Objective	Exercise	Status	Notes
Compile the 3.8 Kernel	Exercise 8	Completed	I didn't have to deal with the ncurses failed dependency.
Install necessary cross compilation tools	Exercise 8a	Completed	
Read and Control GPIO's via the Linux File System	Exercise 10	Completed	Some I/O pins don't work well, possibly due to the presence of pull up/pull down resistors and those being enabled.
Read analog input values from AIN channels	Exercise 10a	Completed	I actually wish they output a straight numerical value, but I guess I can deal with 1mV per division. It makes the math simpler, and is probably enough resolution for most people.
Control GPIO's via C	Exercise 11	Completed	This part was really cool. It was nice to see the C code mesh with the real world (not just key inputs).

1.1 Exercise Summary

1.1.1 Exercise 8

I feel like I have ascended to a whole new plateau of nerddom after having built the Linux Kernel. It is amazing. That's really all I have to say about that. Everything went surprisingly smoothly. I didn't have to deal with the ncurses library dependency issue, which was nice. Also, downloading everything only took about 12 minutes, and compilation probably took about that long as well. The video went a little fast to follow, I wish we could watch it at about half speed.

1.1.2 Exercise 8a

Cross compilation went off without a hitch. Pretty cool to be able to do that.

1.1.3 Exercise 10

Reading a switch can be accomplished by the following commands

```
host$ sys/class/gpio: echo XX > export # add pin XX
host$ sys/class/gpio: cd gpioXX # go into the gpio directory
host$ sys/class/gpio/gpioXX: echo in > direction # make the pin an input
host$ sys/class/gpio/gpioXX: cat value # read the value in from the port
```

1. What's the min and max voltage? Min: -100mV, Max 3.41V.
2. What period is it? The period varies from 250ms to 260ms, with occasional jumps as high as 350ms or 400ms.
3. How close is it to 200ms? Off by about 50ms
4. Why do they differ? Because the OS schedules it to run and occasionally gives priority to other processes.
5. Run htop and see how much processor you are using. 1-2% of the CPU, and less than 0.25% of the RAM.
6. Try different values for the sleep time (2nd argument). What's the shortest period you can get? Make a table of the values you try and the corresponding period and processor usage. THe shortest period I can get is around 70 to 80ms.
7. How stable is the period? The period is not very stable, it usually varies by 10 to 20ms, and occasionally jumps 100 to 150ms higher.
8. Try launching something like mplayer. How stable is the period? I launched gedit, and it was very unstable when the app was launching, but it stabilized close to what we saw earlier when it was just running in the background. When it was in use, however, the period was much less stable, and longer (300-400ms).
9. Try cleaning up togglegpio.sh and removing unneeded lines. Does it impact the period? Cleaning it up shaved off about 20-30ms, which is 10% or so, so it did a pretty good job in my opinion.
10. togglegpio uses bash (first line in file). Try using sh. Is the period shorter? Running it with sh seems to take about the same amount of time, though it seems to be a little more stable.
11. What's the shortest period you can get? sh could run much faster than bash, and I saw periods as short 25-30ms.

Time (s)	Period (ms)	CPU Load (%)
0.1	250-260	1-2
0.05	150-170	3-4
0.01	70-150	5-6
0.01	70-200	6-8

The controlLED shell script uses 6-8% of the CPU, and less than 0.25% of the memory, according to htop. The delay is about 20ms, and it remains fairly constant.

1.1.4 Exercise 10a

The min and max value that I've been able to read are 0 to 1800, and 12 bits is 4096. Therefore, it is reasonable to assume that AIN0-AIN6 output a 0.000 to 1.800 volt signal, where a difference by one is a 1mV difference.

1.1.5 Exercise 11

The correct waveform does appear on the scope. I set up a 50% duty cycle square wave with a period of 0.2 seconds, and I'm getting a 50% duty cycle square wave with a Frequency of 4.98-5.00 hz (or 200-201ms period), which is fairly close to my 0.2 second desired period. The waveform is remarkably stable, at least for relatively slow waveforms; however, for higher frequency waveforms, they become less and less stable, and require more and more CPU power to get the timing correct. Additionally, there are occasionally stretches where the waveform is interrupted as the OS is servicing other requests. This data is presented in the table below.

Desired Period (ms)	Actual Period (ms)	CPU Load (%)
1000	1000	0
100	100-101	0
10	10.5-12.0	4
1	1.25-2.0	14-16
0.1	0.4-1	30-31

The highest frequency seen in my modified togglegpio was 2-3kHz, however it was very unstable, and I believe the highest stable frequencies are those less than 100Hz.

The output tracks the input fairly well, usually about 0.2 to 0.3ms behind, but sometimes stretches to 1-2ms. Depending on the input frequency, and the other programs running on the bone, the signal gets less stable the higher the frequency and the more programs running. It will track reliably up to 100 or 200 Hz, and will still track somewhat well to 1kHz. Above 1kHz however, it really breaks down and by 2kHz it's almost totally unrecognizable. CPU usage is nearly zero for frequencies below 10Hz, and they rise in the single digits until about 250Hz, and by the time 1kHz rolls around, the program takes up about 25% of the CPU.

1.2 GPIO input and Etch-a-Sketch

The actual homework assignment was really easy to complete once the exercises were completed. The etch-a-sketch took some time, but a majority of that time was due to a bit of proper design, which paid off in the end, where I literally combined my two final products (the etch-a-sketch and the four input/output C program) and they just worked perfectly the first time. Shocking, really. I finally feel like I'm getting a hang of physical computing, since I feel like that was a non-trivial product that was created without serious difficulty. I also really like being able to deal with physical input, such as the switches that I wired up, and see that do something. Like the rest of the exercises in this memo, all the source is available on my github repo.

1.3 Takeaway Points

The main points I took away from these exercises were the following:

- Linux is impressively huge. I'm pretty amazed that people created it and still work on it, and I hope to work on it as well.
- I think I have started to get a better feel for how computers have been developed, interfacing all of the hardware I/O together to get say key input and light output. I think it will be really cool to see the I2C LED screen hooked up next week.
- I wish the exercises were renumbered to deal with dependencies (so that exercise 8a came before exercise 8, for instance). It occasionally gets confusing to be jumping around between exercises that are non-sequential to do dependent tasks.