Lab 4 Report | Hao Li | cs425 | 4-18-21

The code is in the Github repository that I shared to you last time.
https://github.com/hl723/cs425

Side Note: Sorry that this assignment is being turned in very late. I know from past labs that each of them will take me multiple consecutive days to complete at a minimum. As a result, I try to start these labs as soon as I get all of my other work out of the way. For this lab, I started 2 days before the original deadline, which was a solid 5-6 days before the first extension. However, out of the 5 days I worked on the lab before the first extension, I was stuck on part 3 for 4 days. At first, I was not stuck, I just simply didn't know enough, however, after a few days, when I knew a lot more about context switching, I was stuck on the implementation. At that time, I knew I couldn't be working on this lab any longer without doing my other assignments, particularly a project that was due in 3 days. So I spent the majority of this week catching up on my classwork for my 4 other classes. Lastly, I was able to complete part 3 with the help of Vickram explaining the knowledge that he got from meeting with Zhiyao.

Part 1

I am completely new to assembly, so it took some time to get familiar with it and apply that to the context of modifying the stack pointer.

Part 2

Once part 1 was working, it was relatively simple (compared to part 3) to get something working that blinks the blue LED every second using a FlashBlue function. Once I was aware of how the exception model worked and knew that there was a wfi() function, this part came together relatively quickly.

When FlashBlue is interrupted by tim2, we enter the interrupt handler. In this case, this would be the tim2 interrupt handler. When we enter the handler, it saves the state of the core registers in the stack inside an exception frame and then when it exits, it pops those registers back and restores the values, including the PC, to its original state, which resumes FlashBlue when the handler exits. Most, if not all of this, is an reiteration of the programming manual in the exception model section. All of this is done for us by the cortex-m4 processor.

Part 3

By far, this part was the bottleneck of this assignment. I completed the first two parts within two days given that I was new to assembly and was figuring out all the implementation details regarding the stack pointer. To start, I was reading the exception model in the programming manual and I soon realized that we had to do some form of context switching. However, I struggled for the longest time to figure out how the pieces came together. For example, I know that we have to make a dummy exception frame so that when the exception returns, we could make the execution go to FlashRed for the first time. However, I was stuck on where I should be pushing this exception frame to in the context of the stack pointers and assembly. Similarly, I know that we have to save the registers (r4-r11, lr) when we do context switching, but I did not know when to do it and the proper way to do it with naked functions.

Part 4

Once I understood how part 3 worked, I was able to relatively quickly refactor the code specific to the kernel into its own interface. I made init() - setting up my data structures, start() - enable systick so that the scheduler can start, create_task() - to add tasks to the kernel, delay() - to give a way to pause execution for a task. There was nothing difficult with this part other than maybe figuring out how to link another source file in Rust, which turned out to be very simple.