Lab Report of Lab 3: Process Scheduling

By: Ian Naeger

12433573

ECE 4220

**Objectives**:

The objective of this lab was to familiarize us further with using kernal modules and to teach us how to use real time tasks and the effects of different priorities on them.

**Lab Description**:

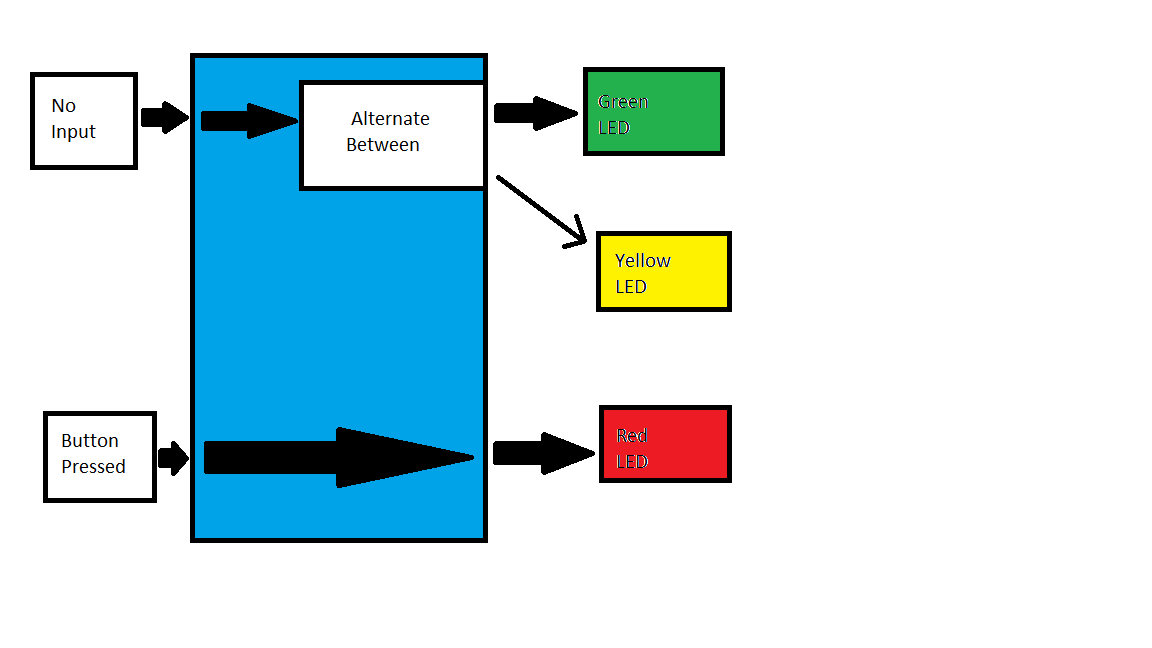
In the first part of the lab, we were tasked with creating a module that would alternate the TS-7250 board’s green and yellow leds until a button was pressed. After the button was pressed, the board would light the red led after the yellow one for one iteration. In the second part of the lab, we had to take our module from the first part and update it to use real time tasks to operate the lights. We then had to find what changing the priority of the different tasks would do.

**Implementation**:

In part one of this lab, I used ioremap to get the locations of the registers of the leds so that I could correctly control them with the module. After getting those, I had to create a simple while loop with an if statement to regulate the operation of the leds.

In part two of the lab, I had to use real time tasks to control the operation of the leds. In order to limit control of the leds I had to implement semaphores to lock control of the leds to certain tasks when appropriate.

Flow Chart:



**Experiments and Results**:

This lab was a breath of relief after lab 2. I found it to be much less difficult, but that isn’t to say it was without it’s difficulty. The initial coding of the first part of the lab did not give me many headaches, but the difficulty came in, oddly enough, compiling the code. After double checking my compiler settings and moving around the location of my libraries, I finally got my code to compile thanks to much help from my TA. Yet, once I got the program actually compiled, it worked quite well!

With the complier issues worked out, the second part of the lab went quite fast! I simply had to adapt my previous code to work as separate tasks for each led. The only problem I had was where to implement the while loop, but through some thinking and a little bit of trial and error, I figured out that each task should be given its own while loop. Once that was sorted, I got the new module to run no problem! The only thing left at that point was to observe what would happen to the module if the priorities on the different tasks were changed, the results of which you can find below in the provided table.

Part 2 Table:

|  |  |  |
| --- | --- | --- |
| Case | Combinations | Observations |
| 1 | PTL1=PTL2=PPL | Green and yellow alternate. If button is pressed, red goes after yellow. |
| 2 | PTL1=PTL2>PPL | Red never goes. |
| 3 | PTL1=PTL2<PPL | If button is pressed, red will always go next. |
| 4 | PTL1>PTL2>PPL | Red never goes. |
| 5 | PTL1<PTL2<PPL | If button is pressed, red will always go next. |
| 6 | PTL1<PTL2=PPL | Green goes once and then never again, from then on yellow stays on, unless the button is pressed. Then red will go before returning to yellow |
| 7 | PTL1>PTL2=PPL | Green goes for twice as long, then yellow goes. If button is pressed while green is on, it will next go red. If pressed while yellow, it will go green and then red. |

**Discussion**:

I found this lab to be a welcome ease up on the difficulty compared to lab 2. I found the further familiarization with the topics of both module creation and board programming to be an incredibly useful opportunity. The knowledge gained from the priority changing is also very helpful in understanding not only how priority changes effect the way a program runs, but also how priority can greatly effect an entire system.

**Post-Lab Questions**:

Part 1:

Question 1: I do not believe this to be real time, because there are no parts of the module that must complete in a certain time frame.

Question 2: You could improve the scheduling of this module by breaking it into real time tasks, like we did in part 2

Part 2:

Question 1 and 2: My only real problem was where to implement my while loop. If I had the code set up like I had part 1 the module didn’t function, but once I placed the loop in each of the tasks, the module worked as intended.

Question 3: It only registers the first press until the button has been cleared again, so nothing.

Question 4: rt\_busy\_sleep holds onto the CPU for its duration, whereas rt\_sleep just goes into the background for its duration. This makes rt\_busy\_sleep typically a worse option and as such I did not use it in my code.

**Code Section**: