Lab 3

Process Scheduling

Objective

In this experiment, students will be exposed to:

- Different scheduling algorithms
- Problems that can arise from different scheduling algorithms
- Learn about process/task starvation, priority inversion, etc

Prelab

In the prelab, you should include the definitions for the following terms: shortest job first, first come first served, foreground-background, round robin, polled scheduling, priority based scheduling, priority inversion, and starvation. Investigate the function *sched_setscheduler*, and provide a short description of the input parameters and output of the function. Read about the following functions to initialize and signal semaphores: *sem_init*, *sem_wait*, *sem_post*. You do not need to write about those functions in the prelab. We will discuss the concept of semaphores and mutexes in class. It's okay if you don't understand everything about those functions, but you should have an idea before coming to the lab. Don't forget to include pseudo code/flow-charts describing the tasks you need to accomplish.

Lab Procedure

The overall purpose of this lab is to implement a traffic light using the Raspberry Pi and the auxiliary board. You will use three of the lights, which represent the signals for one direction, the other direction, and pedestrians (figure-1). When a light is turned on it represents a "green light" or "WALK sign" and when turned off, it represents a "red light". The far left button on the auxiliary board will represents the push button for the pedestrians to use when they wish to cross the street. The lights on the auxiliary board are connected to GPIO ports 2, 3, 4 and 5 on the RPi. Be sure to configure those ports as outputs. The left push button is connected to GPIO port 16. This port should be configured as input. You will need to enable the **pull down** resistor for this port. Make sure that your configuration is correct by reading the port value through the command line gpio utility. When the button is pressed, the port value should read '1'. Otherwise, the port value should read '0'. **Note**: this configuration test is independent of the button functions described below. It's just to make sure that your port 16 is properly configured. Otherwise, the functions below will not work.

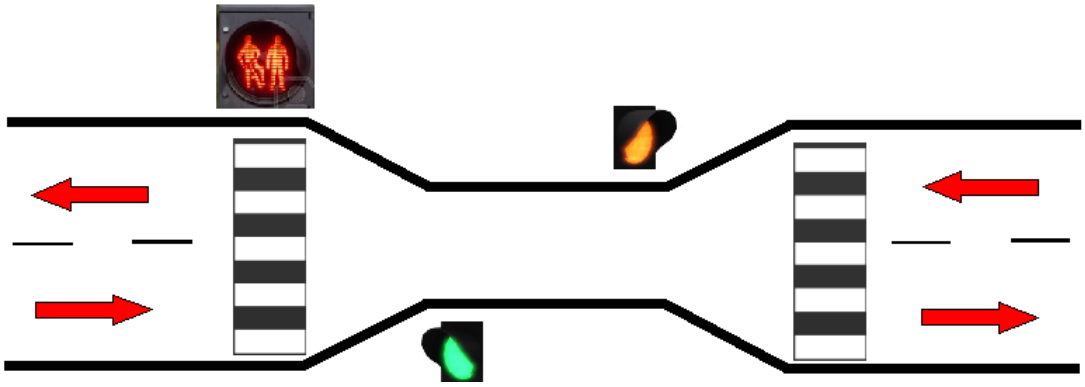

Figure 1: Traffic Lights

Extra Module and Library

For this lab, the pedestrian button will need to be checked to see if it has been pushed. You will use the following functions:

```
// The following function returns the status of the
   button: 1 if it has been pushed, 0 if it hasn't.
int check_button(void);

// After you have handled a pushed button event, call
   this function
void clear_button(void);
```

These functions are a part of the static library **libece4220lab3.a** that needs to be linked to your program. For the functions to work properly, the kernel module **ece4220lab3.ko** needs to be installed before running your program. The module, the library and a header file, **ece4220lab3.h**, that you need to include in your project, are posted on Canvas (*Modules* > *Additional Lab Files* > *ece4220lab3.zip*). Follow the instructions stated in the header file.

Part 1: Polled Scheduling

For this part of the lab, you are to set up a single real time task that acts as a scheduler. It should turn on the light corresponding to one direction, then the light for the other direction, and then check if the pedestrian light needs to be turned on by checking the status of the button.

Questions:

1. What are the limitations of this approach?
2. How can you improve the scheduling so that the implementation acts more like a real traffic light system?

Part 2: Priority Scheduling

For Part 2 you are to create three threads. Each one is responsible for turning ON and OFF one of the lights. Because the lights are a shared resource for all tasks, you need to implement some sort of protection to ensure that no more than one light can be on at any given time. That will be done through **semaphores**.

In your code, you will need to change some parameters of the **pthreads** that you will create. Those are the priority and the scheduling policy. The TA will discuss a basic sequence of steps that your threads need to execute.

For this section, you should experiment with the following combination (table-1 of priority levels for each task and report the corresponding observations.

Case No.	**Priority Combinations**	**Observations**
1	PTL1 = PTL2 = PPL	
2	PTL1 = PTL2 > PPL	
3	PTL1 = PTL2 < PPL	
4	PTL1 > PTL2 > PPL	
5	PTL2 < PTL1 < PPL	
6	PTL1 < PTL2 = PPL	
7	PTL1 > PTL2 = PPL	

Table 1: ***Note****: PTL1, PTL2 and PPL are the priorities of traffic light-1, traffic light-2 and the pedestrian light, respectively.*

Which configuration(s) of priorities will lead to a round robin scheduling scheme? Which configuration(s) will cause a task to starve? What other configurations can you find?

Make sure you report all your findings and explain/discuss the results thoroughly.

Questions:

1. What are some problems that you ran into with this implementation?
2. How did you fix your code to overcome these problems?
3. What happens if the pedestrian button is pressed frequently/rapidly?

Post Lab:

Be sure your lab report contains answers to all of the questions asked in this lab plus the description of your tasks/threads/functions, goals, comments, results, conclusions, etc.