Prelabs: 0
Code/Demo: 40
Report: 47
Delay: 0

Total: 87

Real-Time Embedded Computing
Lab 3 Post Report

Haoxiang Zhang

Due Date: October 7, 2019

47/50

Post Lab

Objectives and Lab Description

Week - 1: Polled Scheduling

5/5

For this part, we have to implement the traffic lights with a single real time task which use a scheduler. There are totally three traffic lights, **GREEN** and **YELLOW** lights are for the two direction of vehicle and **RED** light is for pedestrians. The order for lighting is GREEN light first, then YELLOW light, and then check button and light RED light if button is pressed.

Week - 2: Priority Scheduling

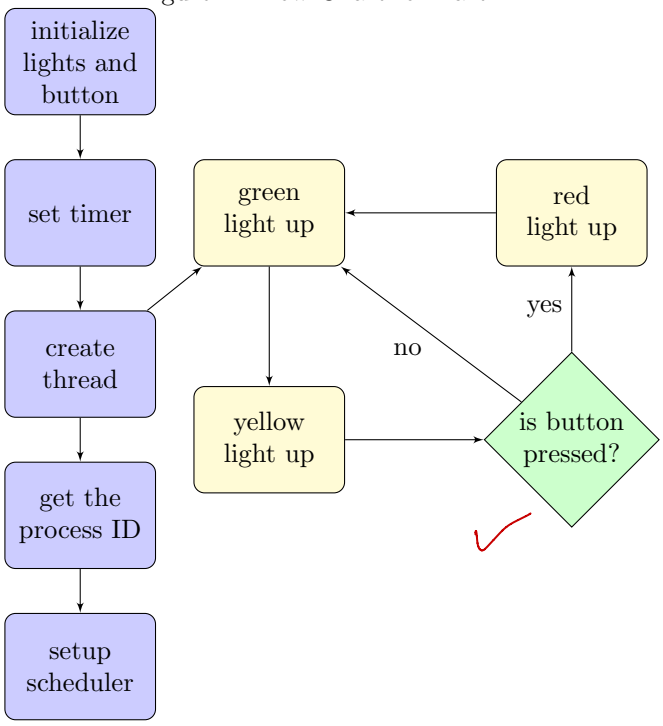
For this part, we need to create three threads that each one control each light. The main point is same as the part one, but for this part we will have to use **semaphores** to control the threads. And we will not use the **timer** and **read()** function. Also, try the different priority combinations and observe the result.

Implementation

Week - 1: Polled Scheduling

Firstly, I setup initialization that shows which lights are working and then turn off. Then, setup timer and priority. Thirdly, create a thread and then get the **pid (process ID)** and put it in the scheduler. In the thread, the function **traffic()**, in a while loop, I set that green light up 1 second, then turned off; then do the same with yellow light; after that check the button, red light will up 1 second if the button is pressed. After red light off clear the button which is set the red light PIN as **DOWN**.
The Flow Chart is below:

Figure 1: Flow Chart for Part - 1



Week - 2: Priority Scheduling

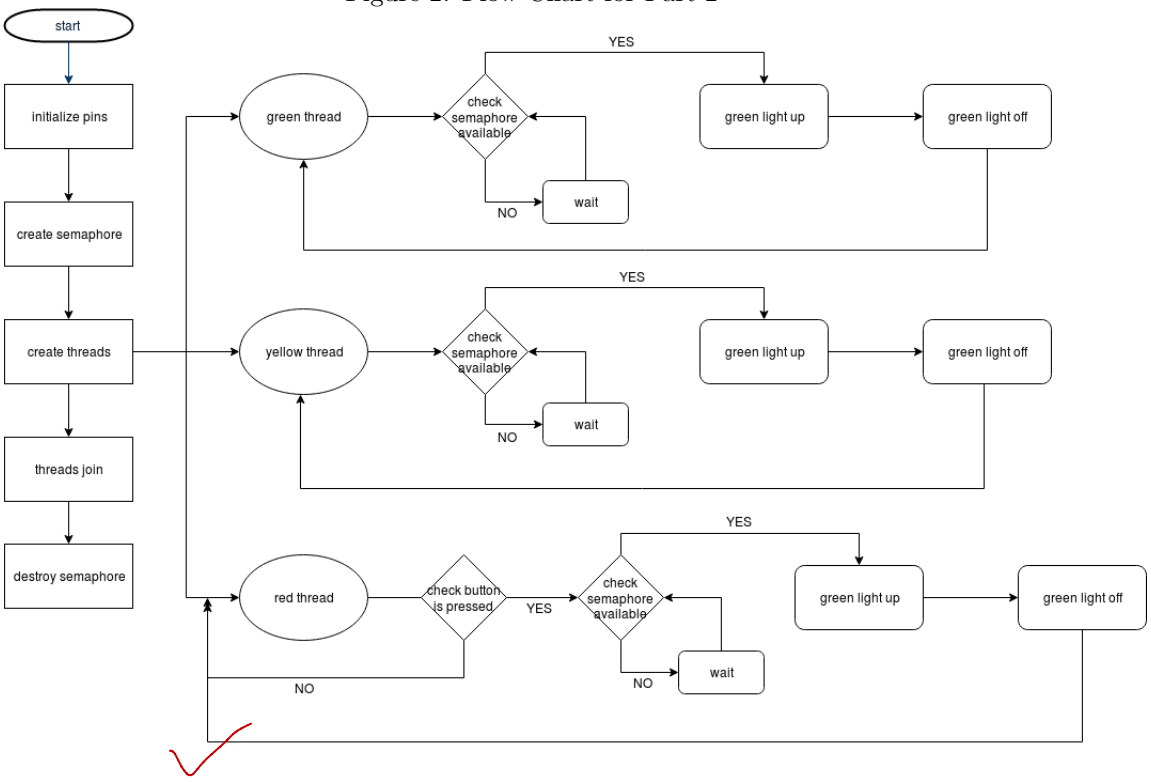
Firstly, initialize the lights and button like what do in part-1. Then declared sem_t type for **semaphore**. Third, initialize the semaphore. Fourth, create three functions that control the light in thread, and put three functions into three threads. At bottom of main function, destroy the semaphore.

In the light control functions, first, set scheduler with priority; and then, in a while loop, we have to use sem_wait() at beginning of the while loop and use sem_post() to control the threads. Between the two sem functions, put the goal action into it.

10/10

For the sem_init() function, the first param is the global variable, which sem_wait() will check that is whether available or not. The second param is set if the global variable is 0 then could run the action code. The third is the amount that could run at same time. For this part we just need to set only one thread at same time.

Figure 2: Flow Chart for Part 2



Experiments and Results

Week - 1: Polled Scheduling

When I run the code, at the beginning, the red, yellow, and green will all light up, then all light down. Then, the green LED lights up for 1 second, then yellow LED lights up for 1 second. If I do not press the button, the next is green LED lights up again. If I press the button, whenever I press, the red LED always lights up after yellow LED.

Week - 2: Priority Scheduling

Because I print the lightening LED in the terminal, I can clearly see how do the lights work. After I run the code. On the board, the green LED and yellow LED are switching every one second. When I press the button, later, the red LED lights up. Because of the different priority, the result after press the button is different.
The table below tells the different result with different priority combinations:

Table 1: Priority Combinations

Priority Problem	
Priority Combinations	Observations
PLT1 = PLT2 = PPL	Yellow and green LED lights alternately, when press button, red LED always lights up after green light.
PLT1 = PLT2 > PPL	When press button, the red LED never lights up.
PLT1 > PLT2 < PPL	If keeping press button, the yellow LED will not light up. Red LED and green LED lights alternately.
PLT1 < PLT2 > PPL	If press the button, the red LED always lights up after yellow light.
PLT1 < PLT2 < PPL	Whenever press the button, the red LED always lights up after previous lighting LED.
PLT1 < PLT2 = PPL	If keeping press button, the green LED will not light up. Red and yellow LED lights alternately.
PLT1 > PLT2 = PPL	Because the priority of yellow and red are same, so press the button when green LED lights, the next light will be red. If press the button when yellow is lighting, then the next light is green, and then is red.

Round Robin / Starvation question is not answered -2

Discussion and Post-lab questions

Week - 1: Polled Scheduling

1. What are the limitations of this approach?
   - The limitation of this approach is that the traffic lights are always follow the order. If pedestrian button is pressed, the red LED will always light up after yellow LED, which mean the pedestrians have to wait a long time to cross the road. It supposed that the red LED lights up after the current lightening LED. ✓
2. How can you improve the scheduling so that the implementation acts more like areal traffic light system?
   - What I will do is that do three traffic lights separate to three threads. Then, when button is pressed, then the pedestrian traffic light will light up after the current lightening light. After the red LED is turned off, then will continue to light up the traffic light that supposed to light up next. ✓
3. The problem I had.
   - For this part, I tried to set the scheduler in the main function. the `sched_setscheduler()` first param is the pid which is process ID. I asked Tushar, he said the pid could get after the thread created. Therefore, I tried to get pid with `pid_t getpid()` to get the current process ID and put it in the `sched_setcheduler()`. It works! ✓

12/15

Week - 2: Priority Scheduling

1. What are some problems that you ran into with this implementation?
   - The only problem I had is at beginning, I did not put a sleep at the bottom of each while loop in light control functions. This caused that the traffic lights are not switching. The reason is that if did not put sleep at the bottom, then the resource will not be released. Thus, for example, after green LED lights down, the thread still hold the resource which means the green LED could run again, so we have to put sleep at the bottom in order to other threads could use the resource. ✓
2. How did you fix your code to overcome these problems?
   - Actually, when I meet the problem, I usually go to search the documentation of the function, example, or other people's explanation. For me, when I learn something, I will imagine how does it work in my mind according to the lecture slides or instructor and TA's words.
3. What happens if the pedestrian button is pressed frequently/rapidly?
   - According to what I have observed, when I pressed button frequently or rapidly, the red LED did not light on all the time. The red LED still will follow the scheduler and the priority to light up. Rarely, for some priority combinations, the red LED should be lighten after yellow light. When press the button frequently or rapidly, the red LED has a really low probability to light up after green LED ✓

Along with answering the questions in the discussion part, also add explanation about what have you learnt in this lab. -1

Comments: 5/5

40/40

Coding Section

Week - 1: Polled Scheduling

```
#include <stdlib.h>
#include <stdio.h>
#include <wiringPi.h>
#include <sys/timerfd.h>
#include <stdint.h>
#include <unistd.h>
#include <sys/types.h>
#include <sched.h>
#include "ece4220lab3.h"
#define RED 8
#define YELLOW 9
#define GREEN 7
#define BLUE 21
#define BTN 27

void init_pins(){
        /*
          to initialize the light.
          this function will show which lights will work.
          after 1 second, all the lights will be turned off.
          here will set blue LED off, because we are not using it.

          to initialize the button.
          the button is the most left button.
        */
        digitalWrite(RED,1);
        digitalWrite(YELLOW,1);
        digitalWrite(GREEN,1);
        digitalWrite(BLUE,0);
        // sleep unit: second
        sleep(1);
        digitalWrite(RED,0);
        digitalWrite(YELLOW,0);
        digitalWrite(GREEN,0);
```

```
}

/*
        note for P2: in each thread
        if yellow is on, then set yellow's priority higher than others
        if green is on, then set green's priority higher than others
        if button is on, the set red's priority higher than others
*/
void* traffic(){
//      int timer_fd;
//      timer_fd = timerfd_create()

        while(1){
                digitalWrite(GREEN,1);
                sleep(1);
                digitalWrite(GREEN,0);
                digitalWrite(YELLOW,1);
                sleep(1);
                digitalWrite(YELLOW,0);
                if (check_button()) {
                        digitalWrite(GREEN,0);
                        digitalWrite(YELLOW,0);
                        digitalWrite(RED,1);
                        sleep(1);
                        digitalWrite(RED,0);
                        clear_button();
                }
        }
}

int main(){
        // set for wiringPi
        wiringPiSetup();

        // set goal pins to operatable
        pinMode(RED,OUTPUT);
        pinMode(YELLOW,OUTPUT);
        pinMode(GREEN,OUTPUT);
        pinMode(BLUE,OUTPUT);
        pinMode(BTN,INPUT);

        // initialize all the pins
        init_pins();
        printf("All lights are set ...\n");
        // clear the button, now button is off
        clear_button();
        // if button is off then print notice
        if(!check_button()){
```

```
            printf("checked button is off\n");
        }

        /*
            start
        */
        int timer_fd = timerfd_create(CLOCK_MONOTONIC, 0);
        struct itimerspec itval;
        itval.it_interval.tv_sec = 4;
        itval.it_interval.tv_nsec = 0;
        itval.it_value.tv_sec = 0;
        itval.it_value.tv_nsec = 0;
        timerfd_settime(timer_fd, 0, &itval, NULL);
        struct sched_param param;
        param.sched_priority = 51;
        // create thread
        pthread_t thread;
        pthread_create(&thread, NULL, traffic, NULL);
        pthread_join(thread);
        // get pid
        pid_t pid = getpid();
        sched_setscheduler(pid, SCHED_FIFO, &param);
        uint64_t num_periods = 0;
        read(timer_fd, &num_periods, sizeof(num_periods));
        if(num_periods > 1) {
            puts("MISSED WINDOW");
            exit(1);
        }

        pthread_exit(0);


        return 0;
}
```

Week - 2: Priority Scheduling

```
#include <stdlib.h>
#include <stdio.h>
#include <wiringPi.h>
#include <sys/timerfd.h>
#include <stdint.h>
#include <unistd.h>
#include <sys/types.h>
#include <sched.h>
#include <semaphore.h>
#include "ece4220lab3.h"
#define RED 8
#define YELLOW 9
#define GREEN 7
#define BLUE 21
#define BTN 27
// Priority of lights
#define PRIORITY_GREEN  53
#define PRIORITY_YELLOW 52
#define PRIORITY_RED    52


sem_t mutex;    // declaration of mutex

void init_pins(){
        /*
          to initialize the light.
          this function will show which lights will work.
          after 1 second, all the lights will be turned off.
          here will set blue LED off, because we are not using it.

          to initialize the button.
          the button is the most left button.
        */
        digitalWrite(RED,1);
        digitalWrite(YELLOW,1);
        digitalWrite(GREEN,1);
        digitalWrite(BLUE,0);
        // sleep unit: second
        sleep(1);
        digitalWrite(RED,0);
        digitalWrite(YELLOW,0);
        digitalWrite(GREEN,0);
}

/*
        note for P2: in each thread
```

```
        if yellow is on, then set yellow's priority higher than others
        if green is on, then set green's priority higher than others
        if button is on, the set red's priority higher than others
*/
void* greeLight(){
        struct sched_param param;
        param.sched_priority = PRIORITY_GREEN;
        // set scheduler
        sched_setscheduler(0, SCHED_FIFO, &param);
        // runing

        while(1){
                sem_wait(&mutex);
                printf("                green light\n");
                digitalWrite(GREEN,1);
                sleep(1);
                digitalWrite(GREEN,0);
                sem_post(&mutex);
                sleep(1);
        }

        pthread_exit(0);
}
void* yellowLight(){
        struct sched_param param;
        param.sched_priority = PRIORITY_YELLOW;
        // set scheduler
        sched_setscheduler(0, SCHED_FIFO, &param);
        // runing
        while(1){
                sem_wait(&mutex);
                printf("yellow light\n");
                digitalWrite(YELLOW,1);
                sleep(1);
                digitalWrite(YELLOW,0);
                sem_post(&mutex);
                sleep(1);
        }

        pthread_exit(0);
}
void* pedestrianLight(){
        struct sched_param param;
        param.sched_priority = PRIORITY_RED;
        // set scheduler
        sched_setscheduler(0, SCHED_FIFO, &param);
        // run
        while(1){
```

```
                sem_wait(&mutex);
                if (check_button()) {
                        printf("                                        Red light\n");
                        digitalWrite(RED,1);
                        sleep(1);
                        digitalWrite(RED,0);
                        clear_button();
                }
                sem_post(&mutex);
        }

        pthread_exit(0);
}

int main(){
        // set for wiringPi
        wiringPiSetup();

        // set goal pins to operatable
        pinMode(RED,OUTPUT);
        pinMode(YELLOW,OUTPUT);
        pinMode(GREEN,OUTPUT);
        pinMode(BLUE,OUTPUT);
        pinMode(BTN,INPUT);

        // initialize all the pins
        init_pins();
        printf("All lights are set ...\n");
        // clear the button, now button is off
        clear_button();
        // if button is off then print notice
        if(!check_button()){
                printf("checked button is off\n");
        }

        /*
                start
        */

        // create thread
        pthread_t thread_green, thread_yellow, thread_pedestrian;
        // initialize semaphore
        sem_init(&mutex, 0, 1);
        // threads create
        pthread_create(&thread_green, NULL, greeLight, NULL);
        pthread_create(&thread_yellow, NULL, yellowLight, NULL);
        pthread_create(&thread_pedestrian, NULL, pedestrianLight, NULL);
        // threads join
```

```
        pthread_join(thread_green, NULL);
        pthread_join(thread_yellow, NULL);
        pthread_join(thread_pedestrian, NULL);

        // destroy semaphore
        sem_destroy(&mutex);


        return 0;
}
```