Prelab: 10
Demo : 31+10
Report: 47
Delay : -6
(2days)
Total: 92

Real-Time Embedded Computing
Lab-4 Report

Haoxiang Zhang

Due date: October 21, 2019

47/50

Post Lab

Objectives and Lab Description

This project is a big one. There are totally three parts: GPS, Main, Real-time Task. The whole project should go like this:

We will have to run an exe file GPS_device to send the GPS to N_pipe1. The main function will receive the GPS from N_pipe1 and store in a shared memory. In the Real-time task file, we have to check the button whether is pressed or not, and we should recored the time that when the button is pressed. The period of the real-time task is 60 ms. The button pressed time should send to N_pipe2. Back to Main function, it will check whether the real-time task send the data to pipe2 or not.

5/5

Because the GPS's period is 250 ms, which means the time that press the button could be up to 5 times. In this case, we have to manage the data for the five times button press. Therefore to create child threads to get the data is a good choice. These child threads could be called **Dynamic Pthreads** The final step, also the bonus part is create another pipe to receive and get data, and print the information on screen. The information displayed should include: Previous GPS, Previous Time GPS, Button GPS, Button Time, Next GPS, and Next Time.

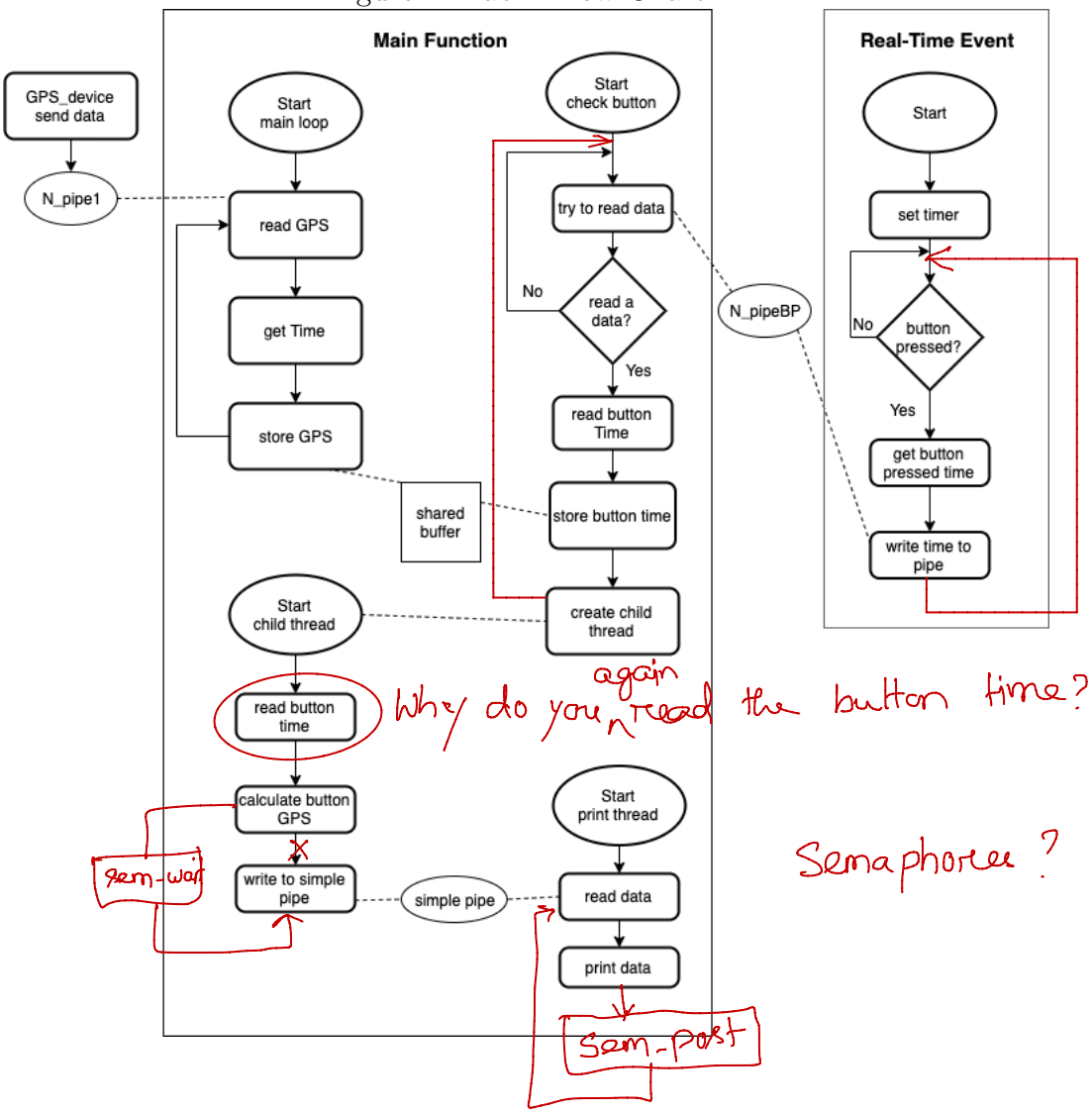
✓

Could have given a shorter description.

Implementation

Flow Chart

Figure 1: Lab-4 Flow Chart



Part 1: Main loop and GPS data

Because the GPS_device is sending the GPS data to N_pipe1, the first step, I have to read the data in N_pipe1 in the main function. Using open() and read() to read the data in the file. Before reading, it is better to set current GPS and current time as previous. After read the data we have to get the time and store the new GPS and new time to shared memory which is the global variables.

Part 2: RT Events and Named Pipe 2

Part 2 is the Real-Time event, so we have to se the timer and scheduler for the event. The period of the real-time event is 60 ms. I set the second as 0 and nanosecond as *6e7* which is 60,000,000 nanosecond or 60 ms. Note: if you want to use 6e7, you should include *math.h* file

As required, I se the pipe name as *\tmp\N_pipeBP* which is Button Pressed. Because we are allowed to use *ece4220lab3.h*, we do not have to set the PIN_MODE, and we have *check_button()* and *clear_button()*. Check the button if it is pressed, then get the time and send time to pipe: *N_pipeBP*, and clear the button.

Part 3: Pthread0, Dynamic Pthreads, and Interpolation

This part, the require is that when button is pressed, create a dynamic pthread to get the information and store into the shared memory. In the check_button() thread function, it was get the button pressed time and print it out. Now I setup it like read the button time stamp from pipeBP and store it in a global array, and create a one time child thread.

In the child thread, first, I read the time stamp in the global array that stores the button time stamp. Then put a while loop that the thread will wait until the new GPS is updated. After that, we have everything beside button GPS. so I declared a float type variable btn_gps and do some calculate with it and make the result as string. Third, I printed the string in the console.

Part 4: Simple Pipe and Printing Thread

This part is required us to read the result information via a print thread, Which mean we have to create another pipe to receive the data and get the data. Therefore, semaphore is necessary. I declared a print_info() to get the result information from the new pipe. The first and the important, I have to set print thread and child thread a higher priority. I removed the printing part in the child thread and put the write function instead of the print. Then, in the print_info(), I read the result from the new pipe and print it out.

For this new pipe, it is a really simple pipe. In order to create the pipe. First, declare a int array with size 2 as a global variable. Why does the pipe need size 2? It is because one is for input of the pipe and the other one is for the output of the pipe. In the Main function, we have to use function pipe() to create it.

Experiments and Results

Part 1: Main loop and GPS data

Before the run the code for part 1, I set the permission of GPS_device with command *sudo chmod -x GPS_device*, and install two mods: ece4220lab3.ko and ece4220lab4.ko. When I run the code, the console print the GPS value continually each 250ms. Also, when the process running, in the other terminal, GPS_device was printing X for each 250ms.

Part 2: RT Events and Named Pipe 2

When run this part. There are three terminal are running at same time. GPS_device still print Xs. There is nothing print out in Main function. However, when I click the button, Main function print out the time stamp of button press. When I set check button function as 1 and run the code, the time printed out very fast, but the period between two times is 60ms which I set it up. There is no problem.

Part 3: Pthread0, Dynamic Pthreads, and Interpolation

When I run the code, I found the time sometimes cannot print out immediately. Also, the button's GPS printed out with a really big number. So, I checked each variable and I found that previous time is great than button time. I went through the code and check. Finally I found that I did not setup priority for child thread and print thread. That is why the result cannot print out immediately. And that is why the previous time is great than button time, because some other process is running when I run the code. After I fixed the problem, the result print out on time when I pressed the button. I tried set the conditional judgment: check button as True in the while loop in the Real-time event. When I run it, the result print out for each 60ms and also the GPS time still is 250ms period. There are up to 5 times that show the button GPS between the period of GPS update.

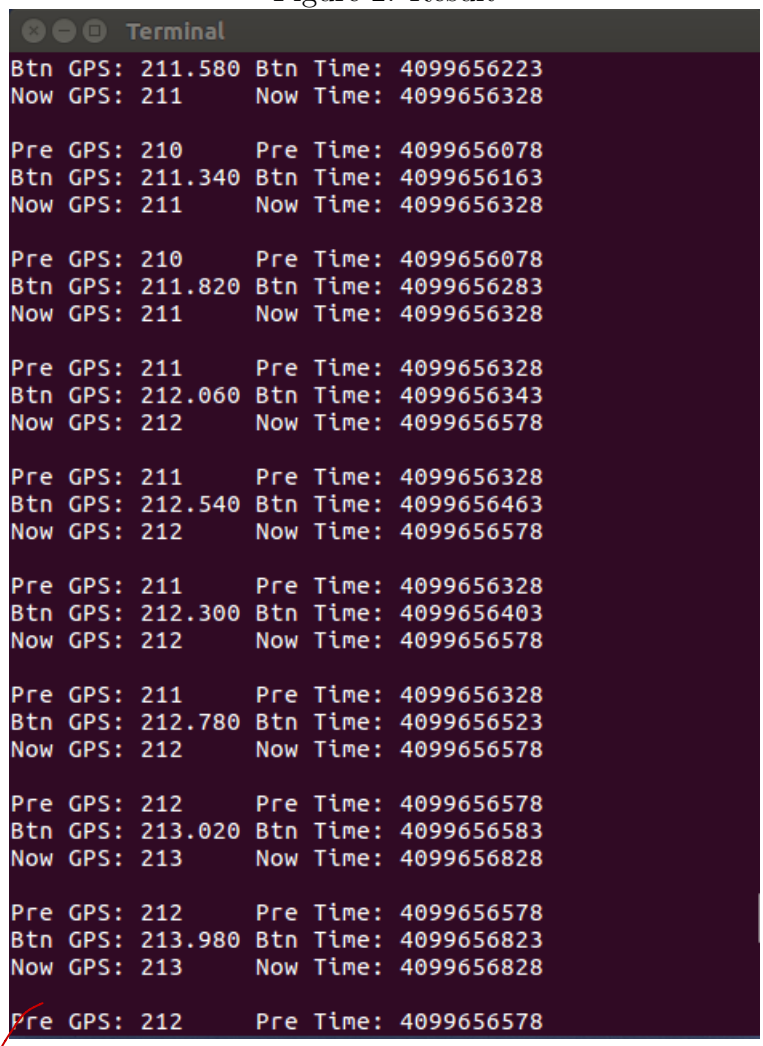
Put this explanation in "Discussion" section.

Part 4: Simple Pipe and Printing Thread

This part is about create an extra thread to read the data from a simple pipe and print out to console. When I run it, and press the button, the result is also correct. In the Figure ??, It shows the part of result.

15/15

Figure 2: Result



Discussion and Post-lab questions

1. Given the specified periods for the RT task and the incoming GPS data, up to how many dynamic threads do you expect to be created between two consecutive GPS events? Make sure that your program can handle all events. Test using both kernel modules. Try pushing the button multiple times really fast, and try keeping the button pushed for a few seconds. Report your observations.

    - The maximum of dynamic threads is 5. Because the period of GPS is 250 ms and the period of button event is 60 ms, the process could create up to 5 threads.

2. Imagine that you want to do the interpolation with more GPS points in order to have more accurate result. What would be changed in your program? What kind of buffer would you need? Why? You don't need to implement this.

    - If the more GPS points all write to pipe at a same time, we could us create a matrix for store the data, because we could store all the points at one time and we also could store all the information from the dynamic thread. I think the program do not need to change a lot to implement.

13/15

3. When your program reads from the pipes, is that a blocking operation or nonblocking operation? How do you know?

    - I think it is a nonblocking operation, because if it is a blocking operation, the data only could go like one in one out. This could not share the data.

-2

Comments: 5/5

31+10
40

i) Sometimes the values are greater than next GPS number. because he way using mutex before & after "while()" loop. (-3)

ii) Parsing same structure to every thread. (-2)

iii) Button press event : only 1 (-4)

iv) Extra (+10)

Coding Section

Main Function

Listing 1: Main thread to read GPS and Push Button and interpolate the Event position

```
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <sched.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <time.h>
#include <math.h>
#include <string.h>

unsigned int    pre_gps = 0;
unsigned int pre_time = 0;
unsigned int    now_gps = 0;
unsigned int now_time = 0;

sem_t mutex;
// pthread lock
pthread_mutex_t mut = PTHREAD_MUTEX_INITIALIZER;
//pthread_cond_t cond = PTHREAD_COND_INITIALIZER;

int simple_pipe[2];

void* print_info(){
  unsigned char info[200];
  while(1) {
```

```
29      read(simple_pipe[0], info, sizeof(info));
30      printf("%s\n", info);
31    }
32    pthread_exit(0);
33  }
34
35  void* child_thread(void*timestamp) {
36    unsigned int btn_time;
37    btn_time = (unsigned int)timestamp;
38    //printf("ts: %u\n", btn_time);
39    unsigned int pre = now_time;
40    unsigned int pgps = pre_gps;
41    if (pre_time != now_time) {
42      pre = pre_time;
43    }
44    // lock the pthread
45    pthread_mutex_lock(&mut);
46    while (btn_time > now_time) { }
47    // unlock the pthread
48        pthread_mutex_unlock(&mut);
49
50    unsigned int now = now_time;
51    unsigned int ngps = now_gps;
52    if (pre == now) printf("!!!!\n");
53
54    float btn_gps;
55    btn_gps = (float)(btn_time-pre)/(float)(now-pre) + (float)
       pre_gps;
56    // store the information into a string
57    unsigned char info[200];
58    sprintf(info,
59      "Pre GPS: %u      Pre Time: %u \nBtn GPS: %.3f Btn Time: %u \
       nNow GPS: %u      Now Time: %u \n ",
60      pgps, pre, btn_gps, btn_time, ngps, now);
61    // write the information into a simple pipe
62    sem_wait(&mutex);
63    write(simple_pipe[1], info, sizeof(info));
64    sem_post(&mutex);
65    pthread_exit(0);
66  }
67
68  void* check_button(){
69    int fd_cb;
70    char* pipeBP = "/tmp/N_pipeBP";
71    size_t* timestamp;
```

```
72    timestamp = (size_t*)malloc(sizeof(size_t));
73    if (timestamp == NULL){
74      printf("Memory allocation failed\n");
75      exit(1);
76    } else {
77      printf("successful\n");
78    }
79    int i = 0;
80    while(1){
81      if (fd_cb = open(pipeBP, O_RDONLY)){
82        read(fd_cb, timestamp+i, sizeof(timestamp));
83        // create child thread
84        pthread_t thread_child;
85        pthread_create(&thread_child, NULL, child_thread,(void*)*(
      timestamp+i));
86        i++;
87      }
88    }
89    free(timestamp);
90  }
91
92  int main(){
93    // set time
94    struct timeval spec;
95
96    // initialize sem
97    sem_init(&mutex, 0, 1);
98
99    // create simple pipe
100   pipe(simple_pipe);
101
102   // create thread to print
103   pthread_t thread_print;
104   pthread_create(&thread_print, NULL, print_info, NULL);
105
106   // create thread to check button
107   pthread_t thread_check_btn;
108   pthread_create(&thread_check_btn, NULL, check_button, NULL);
109
110
111   // reading the information from N_pipe1
112   int fd;
113   char* pipe1 = "/tmp/N_pipe1";
114   int gps=0;
115   while(1){
```

```
116     if (fd=open(pipe1, O_RDONLY)) {
117       pre_gps = now_gps;
118       pre_time = now_time;
119       // Read the GPS
120       read(fd, &gps, 1);
121       // store GPS
122       now_gps = gps;
123       // get the time and store
124       gettimeofday(&spec, NULL);
125       now_time = spec.tv_sec*1000 + spec.tv_usec/1000;
126       sleep(0.01);
127     }
128
129   }
130   close(fd);
131   sem_destroy(&mutex);
132
133   return 0;
134 }
```

Button Event

Listing 2: Real-time Event: Check button is pressed and write the time stamp into a pipe

```
1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <fcntl.h>
4  #include <sys/stat.h>
5  #include <sys/types.h>
6  #include <stdint.h>
7  #include <unistd.h>
8  #include <pthread.h>
9  #include <wiringPi.h>
10 #include <semaphore.h>
11 #include <sched.h>
12 #include "ece4220lab3.h"
13 #include <time.h>
14 #include <math.h>
15 #include <string.h>
16
17
18 int main() {
19   // set for wiringPi
20   wiringPiSetup();
```

```
21  pinMode(27,INPUT);  // btn1
22  // set timer
23  int timer_fd;
24  timer_fd = timerfd_create(CLOCK_MONOTONIC, 0);
25  if(timer_fd == -1) printf("ERROR: timerfd_create1\n");
26  struct itimerspec itval;
27  itval.it_interval.tv_sec = 0;
28  itval.it_interval.tv_nsec = 6e7;
29  itval.it_value.tv_sec = 0;
30  itval.it_value.tv_nsec = 6e7;
31  // start the timer
32  if(timerfd_settime(timer_fd, 0, &itval, NULL) == -1) printf("
     ERROR: timeset1\n");
33  // set pipe
34  int fd;
35  // FIFO file path
36  char* pipe2 = "/tmp/N_pipeBP";
37  mkfifo(pipe2, 0777);
38  unsigned int bp;
39
40
41  // initialize current thread as Real Time
42  struct sched_param param;
43  param.sched_priority = 55;
44
45  // wait to timer expire
46  uint64_t num_periods = 0;
47  ssize_t rr;
48
49  struct timeval spec;
50  clear_button();
51  sched_setscheduler(0, SCHED_FIFO, &param);
52  printf("start\n");
53  while (1) {
54    if (!check_button()) {
55      // set time
56      gettimeofday(&spec, NULL);
57      bp = spec.tv_sec*1000 + spec.tv_usec/1000;
58      //printf("bp: %u\n", bp);
59      fd = open(pipe2, O_WRONLY);
60      write(fd, &bp, sizeof(bp));
61
62      clear_button();
63
64    }
```

```
65      rr = read(timer_fd, &num_periods, sizeof(num_periods));
66      if (rr == -1) printf("ERROR: read1\n");
67      if(num_periods > 1) {
68              puts("MISSED WINDOW");
69                exit(1);
70        }
71
72    }
73    close(fd);
74    pthread_exit(NULL);
75    return 0;
76  }
```