



Microprocessors II and Embedded Systems

EECE.4800

Multithreaded Programming

Yan Luo

Group 1

Grayson Colwell

Handed in: December 21, 2017

Lab Due: December 22, 2017

## Section 2: Contributions

/1 points

1. Group Member 1 – Grayson Colwell
  - Developed modules used for threads 1 and 2. Assisted with the debugging of threads 1 and 2. Brought together code from previous labs to communicate with the Galileo to the PIC, implement the temperature sensor via I<sup>2</sup>C, and the code used to name the pictures taken by the webcam.
2. Group Member 2 – Andrew Macgregor
  - Developed the module for thread 3 which communicates with the web server and sends data collected by thread 1 and 2. Developed the PIC code to allow the use of the PWM module for the servo to rotate continuously either 90 or 180 degrees. Assisted with the debugging of threads 1 and 2.
3. Group Member 3 – Jose Velis
  - Assisted with the debugging of threads 1 and 2 and wrote the temperature sensor code from Lab 3.

## Section 3: Purpose

/0.5 points

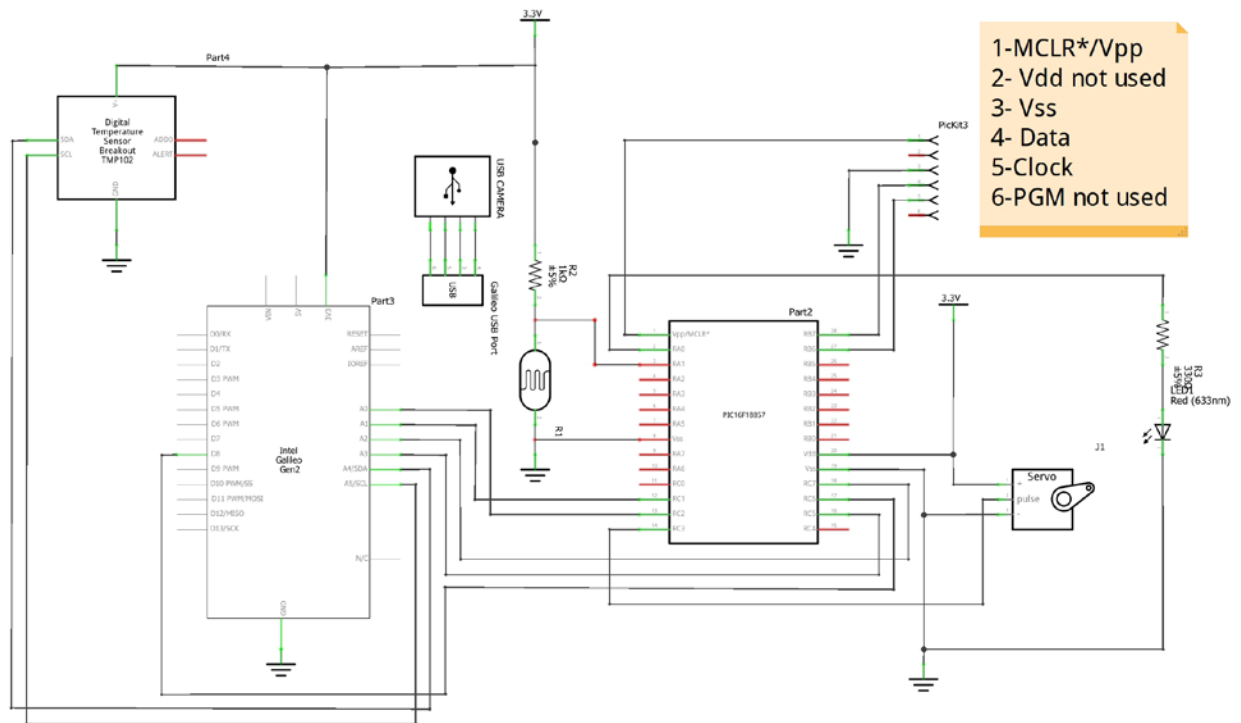
The purpose of this lab was to gain an understanding of how to implement multithreaded programming. This lab works to combine elements from the previous labs and combine them all together by having multiple threads perform different tasks. Synchronization must also be used to ensure that there are no data racing conditions with threads changing shared variables.

## Section 4: Introduction

/0.5 points

To be successful in this lab, there were a few key concepts that needed to be understood. First, successful completion of the previous labs is a huge bonus in this lab because this lab works to bring together most of the components from the previous lab and to collect data collected by the PIC and the temperature sensor and send it to the web server. There were three threads used in this lab which will perform these functions, the user command thread which allows the user input commands that the PIC can perform. The second thread, the sensor control thread, is responsible for updating the values from the PIC LDR sensor and also to read the temperature from the temperature sensor. Lastly, the third thread is used to send the data collected to the web server based on conditions set forth from thread 1.

- **Servo Motor SG90:** Motor which rotates between 0 and 180 degrees depending on the duty cycle.
- **Photoresistor (LDR):** Resistor with a variable resistance depending on the amount of light that is read by it.
- **PIC16F18857:** Embedded microcontroller which would be the brain of the circuit controlling the servo, ADC module, and LED.
- **Light Emitting Diode (LED):** Used as a visual indicator of how the photoresistor operates.
- **Analog Discovery, Serial #210244630597:** Used the Discovery to act as a voltage source and as an oscilloscope to measure the voltage of the LDR.
- **1k $\Omega$  and 330 $\Omega$  Resistors:** The 1k $\Omega$  resistor was used to form a voltage divider with the LDR and the 330 $\Omega$  was in series with the LED to ensure that it didn't burn out from being given too much voltage.
- **Intel Galileo Gen2 embedded computer:** The embedded computer used to gain an understanding of I2C communication between devices.
- **TMP102 Temperature Sensor:** temperature sensor which reads the temperature.
- **USB webcam:** camera used to send pictures based off the conditions determined by thread 1.



fritzing

Figure 1: Pinout and layout of the circuit used for this lab drawn using Fritzing.

In Figure 1, the schematic shown is how the circuit was configured to allow the Galileo to interact with the PIC and temperature sensor to gain temperature and LDR readings.

### Hardware design:

When looking at the implemented circuit, this lab combined elements from all the previous labs and no changes from those labs were made to our hardware design. The sensor circuit designed in Lab 2 was used to connect and communicate with the PIC from the Galileo. The same connected between the Galileo and both the camera and the temperature sensor from Lab 3 were also reused for this lab.

## Software design:

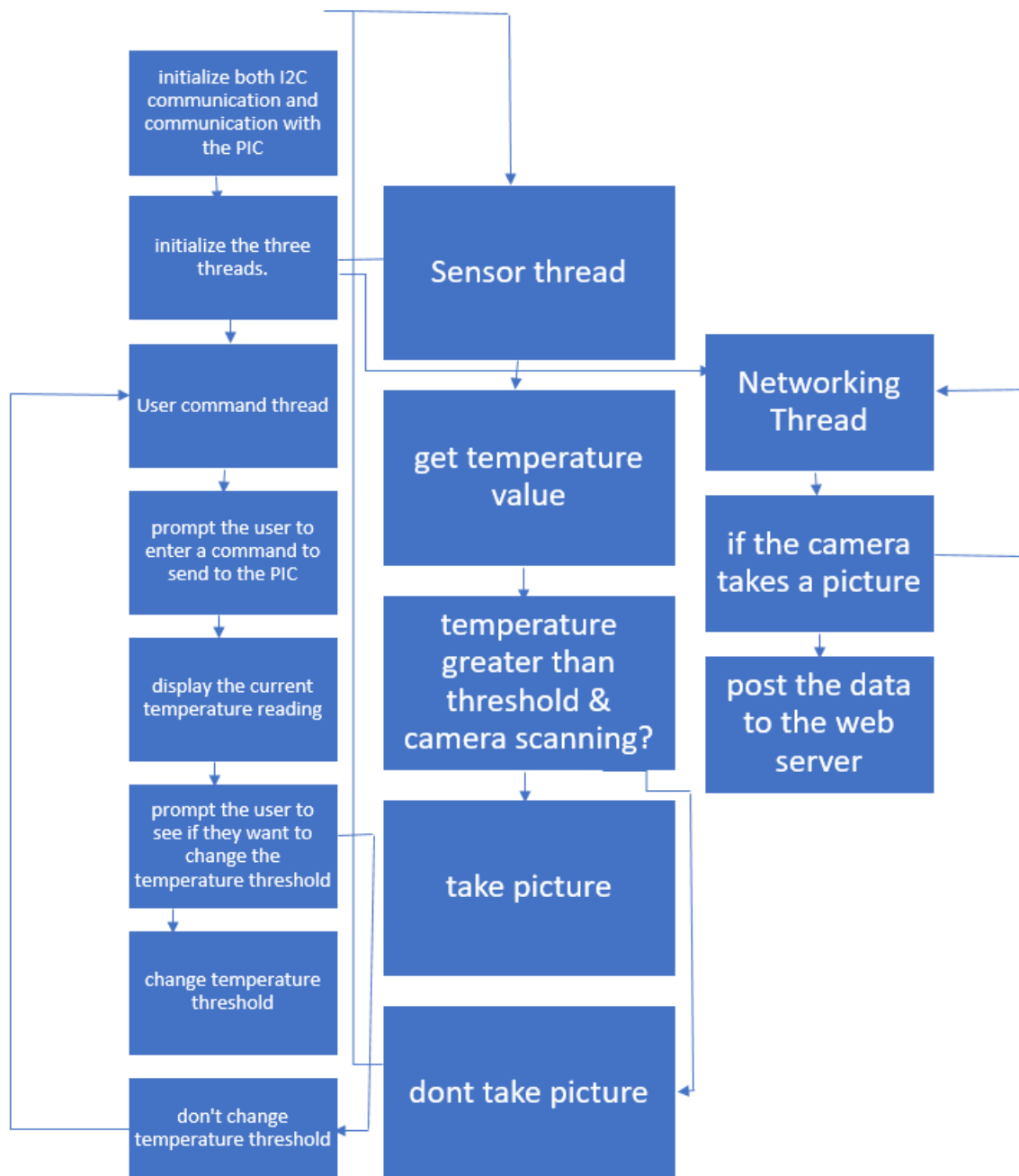


Figure 2: Software Design Flowchart.

In Figure 2, the flowchart is shown for the use of the three threads in this program. Thread 1 is the user command thread which asks the user which PIC command they would like to do, display the current temperature reading, and prompts the user whether they want to change the temperature threshold. The second thread, the sensor thread, updates the value of the LDR and the temperature value and also

checks to see if the temperature value is greater than the threshold value and if the camera is in scanning mode, the sensor thread will take a picture. The third thread, the networking thread will wait until the picture is taken and once this picture is taken, the thread will send the data to the webserver.

```
////create threads
pthread_t command_thread;
pthread_t sensor_control_thread;
pthread_t net_thread;

////run threads
pthread_create(&command_thread, NULL, commandLoop, NULL);
pthread_create(&sensor_control_thread, NULL, sensor_control, NULL);
pthread_create(&net_thread, NULL, serverPostLoop, NULL);

pthread_join(command_thread, NULL);

//jump out if the command loop exits
printf("Main thread stopped\n");
pthread_exit(NULL);
```

Figure 3: Initializing the threads.

In Figure 3, the three threads used in this lab were initialized. Each of the threads performs goes to a different function and all the threads will run forever unless the command thread receives the quit command.

```
pthread_mutex_lock(&myData_m);
if(cur_temp > -50)
    printf("(1)Current Temperature: %lf\n", cur_temp);
pthread_mutex_unlock(&myData_m);
```

Figure 4: Example of using the mutex.

In Figure 4, an example of using a mutex to avoid data racing is shown. In this example, the current temperature reading from the temperature sensor is printed and mutexes are necessary in this case so the current temperature is not updated mid print statement.

```

while(true)
{
    sleep(2);

    //update data
    server_data currentState = getCurrentState();

    //format post string
    strncpy(filename, currentState.fileName, MAX_FILENAME);
    if(currentState.picOnline)
    {
        strncpy(picStatus,PIC_ONLINE, 10);
    }
    else
    {
        strncpy(picStatus, PIC_ERROR, 10);
    }
    picAdcValue = currentState.adcData;

    getPostRequest(postBuffer, MAX_POST, picAdcValue, picStatus, timeStamp, filename);

    //send it
    //printf("(3)Post: %s\n\n", postBuffer);
    HTTP_POST(postBuffer);
}

```

Figure 5: Server thread code.

In Figure 5, the server thread is displayed. This code will send the values read from thread 2 to the server and will also send the picture captured to the server and if there is not a picture taken, then the thread will send a message stating there is no picture. This thread will also only send an update to the web server every two seconds as described in the specification.

## Section 8: Trouble Shooting

/1 points

### Issue 1: Data not updating correctly

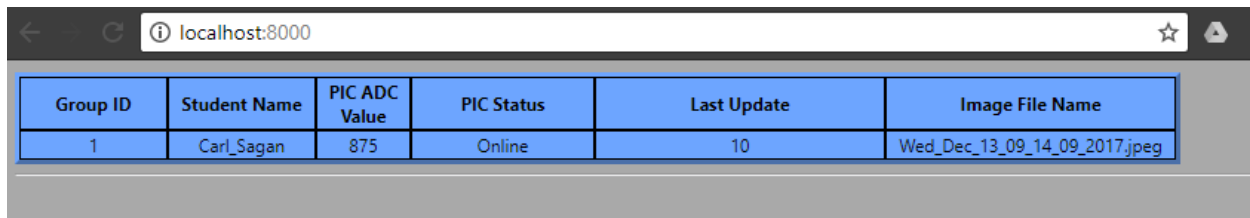
When we first began attempting to read data from the server thread, we were running into an error where the value we were getting was not what we were expecting it to be. We thought that it was since the sensor thread was updating to frequently so we added a two second delay after the sensor thread updated and when we were still running into problems. Because of this, we looked into the mutexes and we realized that we had improperly initiated and used our mutex so once we fixed that, we no longer ran into problems with our code.

### Issue 2: Using the PWM module

When we first began working on the servo module to have it rotate continuously, we were going to use the PWM module to interface with the servo motor. We constantly were running into problems with initializing the PWM module on the PIC and were seemingly going nowhere as it was not rotating the servo correctly. After many hours of debugging, we were finally able to get the PWM module to work so that it was able to rotate the servo continuously both 90 and 180 degrees.

## Section 9: Results

/0.5 points

A screenshot of a web browser window displaying a table of data. The browser's address bar shows 'localhost:8000'. The table has six columns: 'Group ID', 'Student Name', 'PIC ADC Value', 'PIC Status', 'Last Update', and 'Image File Name'. There is one data row with the following values: '1', 'Carl Sagan', '875', 'Online', '10', and 'Wed\_Dec\_13\_09\_14\_09\_2017.jpeg'.

| Group ID | Student Name | PIC ADC Value | PIC Status | Last Update | Image File Name               |
|----------|--------------|---------------|------------|-------------|-------------------------------|
| 1        | Carl Sagan   | 875           | Online     | 10          | Wed_Dec_13_09_14_09_2017.jpeg |

Figure 6: Output to the webserver.

In Figure 6, the output to the webserver is shown. The program could properly output the PIC ADC value and the name of the most recent image taken to the web server.



## Section 10: Appendix

### A1. CommandLoop Thread code:

```
void*
commandLoop(void*)
{
    int input;
    int temp_thresh_input;
    int scanf_test;
    double temp;
    int ping_ret; //return value of ping function

    do {
        //print the main menu
        printf("(1)Select a number for desired action: \n\n");
        printf("1. Reset\n");
        printf("2. Ping\n");
        printf("3. Get ADC value\n");
        printf("4. Turn Servo 30 degrees\n");
        printf("5. Turn Servo 90 degrees\n");
        printf("6. Turn Servo 120 degrees\n");
        printf("7. Rotate the Servo continuously 90 degrees
enable\n");
        printf("8. Rotate the Servo continuously 180 degrees
enable\n");
        printf("9. Rotate the Servo continuously 90 degrees
disable\n");
        printf("10. Rotate the Servo continuously 180 degrees
disable\n");
        printf("11. Exit\n");

        //print the temp if a valid reading is obtained
        pthread_mutex_lock(&myData_m);
        if(cur_temp > -50)
            printf("(1)Current Temperature: %lf\n",
cur_temp);
    }
```

```

pthread_mutex_unlock(&myData_m);

//check for input.
input = 0;
scanf_test = scanf("%d", &input);

//If ipmproperly formatted,
// set input to 0 to prompt user to input again
if (scanf_test == 0)
    input = 0;
switch (input)
{
case 1:
    reset();
    break;
case 2:
    ping_ret = ping(10);

    //update pic status
    pthread_mutex_lock(&myData_m);
    if(ping_ret == MSG_ACK)
        myData.picOnline = true;
    else
        myData.picOnline = false;
    pthread_mutex_unlock(&myData_m);

    break;
case 3:
    //print adc value collected from sensor loop
    pthread_mutex_lock(&myData_m);
    printf("LDR Value = %d\n", myData.adcData);
    pthread_mutex_unlock(&myData_m);
    break;
case 4:
    servo_30();
    break;
case 5:
    servo_90();
    break;
case 6:

```

```

        servo_120();
        break;
    case 7:
        rotate_servo_90_E();
        pthread_mutex_lock(&myData_m);
        scan_solo = 1;
        pthread_mutex_unlock(&myData_m);
        break;
    case 8:
        rotate_servo_180_E();
        pthread_mutex_lock(&myData_m);
        scan_solo = 1;
        pthread_mutex_unlock(&myData_m);
        break;
    case 9:
        rotate_servo_90_D();
        pthread_mutex_lock(&myData_m);
        scan_solo = 0;
        pthread_mutex_unlock(&myData_m);
        break;
    case 10:
        rotate_servo_180_D();
        pthread_mutex_lock(&myData_m);
        scan_solo = 0;
        pthread_mutex_unlock(&myData_m);
        break;
    case 11:
        return NULL;
    default:
        printf("Please enter a valid number (1 -
10)\n");

        break;
}
//print the current temperature reading
pthread_mutex_lock(&myData_m);
printf("Current Temperature settings:\n\ttemp =
%lf\n\tthreshold = %lf\n", cur_temp, temperature_threshold);
pthread_mutex_unlock(&myData_m);

//ask the user to change the temp threshold
printf("Would you like to change the Temperature
Threshold?\n (1 = yes)\n");

```

```

scanf(" %d", &temp_thresh_input);
if (temp_thresh_input == 1)
{
    pthread_mutex_lock(&myData_m);
    printf("Enter the new Temperature
Threshold:\n");

    scanf("%lf", &temperature_threshold);
    pthread_mutex_unlock(&myData_m);
}
} while (input != 11);
}

```

## A2. Sensor thread loop

```

void*
sensor_control(void
*)
{
    while (1)
    {
        time_t date = time(NULL);
        char* cdate;

        //probe the buses for sensor data. Use temp variables
        to avoid hold and wait
        //adc_value is mutex protected itself
        //printf("(2)Calling adc_value\n");
        int tmpLdr = adc_value();

        //probe temp sensor on the i2c bus
        //printf("(2) Calling get Temp.");
        double tmpTemp = get_temp();

        //printf("(2)Read temp: %lf, and ADC: %d\n", tmpTemp,
        tmpLdr);

        //update shared state variables
        pthread_mutex_lock(&myData_m);
        myData.adcData = tmpLdr;
        cur_temp = tmpTemp;
    }
}

```

```

pthread_mutex_unlock(&myData_m);

//take a picture if above the temp threshold and in
scan mode
if((cur_temp > temperature_threshold) && (scan_solo ==
1))
{
    //form the filename. Filter out spaces, colons
and newlines

    date = time(NULL);
    cdate = asctime(localtime(&date));
    for(int i = 0; i < 25; i++)
    {
        if (cdate[i] == ' ')
        {
            cdate[i] = '_';
        }
        else if (cdate[i] == ':')
        {
            cdate[i] = '_';
        }
        else if (cdate[i] == '\n')
        {
            cdate[i] = '_';
        }
    }
    printf("(2)Taking picture: %s \n", cdate);
    capture_and_save_image(cdate);

    //update latest filename
    pthread_mutex_lock(&myData_m);
    strcpy(myData.fileName, cdate);
    pthread_mutex_unlock(&myData_m);
}

sleep(2);

}
}

```

### A3. Server Thread

```
void*
serverPostLoop(void
* x)
{
    char picStatus[10];
    strcpy(picStatus, PIC_ERROR);
    char filename[MAX_FILENAME];
    strncpy(filename, DEFAULT_FILENAME, MAX_FILENAME);
    char timeStamp[MAX_TIMESTAMP];
    int picAdcValue = 0;

    char postBuffer[MAX_POST];

    while(true)
    {
        sleep(2);

        //update data
        server_data currentState = getCurrentState();

        //format post string
        strncpy(filename, currentState.fileName, MAX_FILENAME);
        if(currentState.picOnline)
        {
            strncpy(picStatus, PIC_ONLINE, 10);
        }
        else
        {
            strncpy(picStatus, PIC_ERROR, 10);
        }
        picAdcValue = currentState.adcData;

        getPostRequest(postBuffer, MAX_POST, picAdcValue, picStatus,
timeStamp, filename);

        //send it
```

```
//printf("(3)Post: %s\n\n", postBuffer);  
HTTP_POST(postBuffer);  
  
}
```