

16.480/552 Microprocessor Design II and Embedded Systems

**Lab 4: Multithreaded Programming**

Instructor: **Yan Luo**

TA: Ioannis Smanis

Group 4 Members

Chethan Nagesh Pal Advait Churi

Student Name; **Chethan Nagesh Pal** (student id :01741616) Hand in Date :12/21/2017

Lab Due Date:12/21/2017

*Section 2: Contributions /1 points*

1. Group Member 1 – Chethan Nagesh Pal

I helped the team in understanding the thread functions. I studied the link shared in GitHub and understood the operation of the threads .I helped the team in writing the code related to initializing the thread, adding mutex to the appropriate places in the code ,so the operation will go as expected without any data race or hazards. I helped the team in developing the code related to server access. I tried implementing face detection ,by adding haar file of .xml type to the appropriate location in OpenCV library

*Section 3: Purpose /0.5 points*

Purpose of this lab is to understand the POSIX thread libraries and utilize their functions to design the multithreaded programs for embedded multicore system. Here we are learning how to combine all the 3 project that we done previously by using threads, for faster action and sending the results to the server periodically .

*Section 4: Introduction /0.5 points*

Multithreading is a programming and execution model that allows multiple threads to exist within one process. These threads share the process’s resources. multithreading can allow an application to remain responsive to input throughout the program. In this lab we are utilizing this characteristic to combine all 3 programs that we built previously .we are using 3 threads and performing task decomposition ,we are assigning each threads with each different function .In first thread we taking input from the user and performing several operation related to PIC side of the program ,second we are using it to poll the sensor part of the program and in third thread we are updating server with the value that we have found in the previous 2 threads. we are using another important part of POSIX thread library, that is mutex. Mutex is used provide mutual exclusion to the critical region of the program that access the shared variable.There are chances of data race and deadlock because of the structure of the program we using ,in order to avoid these hazards we providing access to single thread to access the critical region at a given time . We are utilizing the following components from previous lab experiments.

*Section 5: Materials, Devices and Instruments /0.5 points*

Lab1: 1.PIC16F18857

2.PICkit3 3.Servo motor 4.FTDI cable

5.Ligh dependent resistor

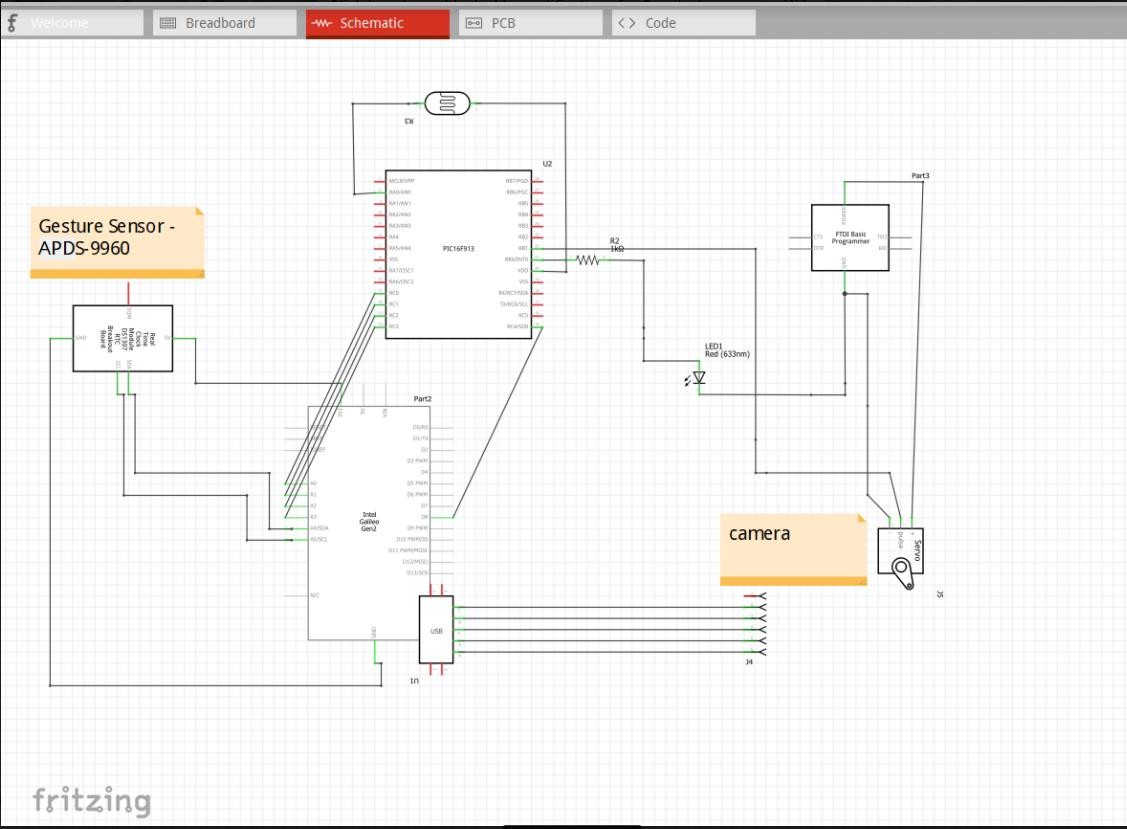
Lab2:

1.Intel Galileo 2nd Generation 2.WIFI card

3.SD card Lab3:

1. Gesture sensor (APDS-9960) 2.Web Cam

*Section 6: Schematics /0.5 points*



*Section 7: Lab Methods and Procedure /2 points*

## Hardware design:

In this lab we are implementing all the hardware concepts and connection that we have performed for the previous lab experiments .so the hardware connection from all 3 previous labs remains unchanged with only exception that camera is mounted on the head of the servo motor.

Lab1 : **Sensor Design and Analog Digital Conversion**

Microcontroller (PIC16F18857) 1.Power Up

\*VPP to PICKIT3 VPP ,To Enable Programing mode of PIC

\*VDD to PICKIT3 VCC ,Positive Power supply to PIC

\*VSS to PICKIT3 GND,

1. Data and clock transfer

* ICSPCLK to PICKIT3 ICSPCLK ,In-Circuit Serial Programing and debugging Clock input and output

\*ICSPDAT to PICIT3 ICSPDAT,In-Circuit serial Programing and debugging data input and output

1. Input /output

* ANSA0/RA0; assigned for analog input

\*RB1 is to provide output to Servo motor

\*RB0 is to provide output to LED

\*RX ,TX to provide data to computer through FTDI

B. Servo Motor

\*VCC to FTDI VCC ,5 v

\*PWM (input) to RB1 of PIC ,Input from the PIC in PWM signals

\*GND to FTDI GND .

C.LDR

\*Upper part connected to PICKIT3 VCC,3.3V

\*Lower part connected in parallel to the ADC input (RA0)and 2.7 k Resistor

\*Overall circuit is connected in Voltage divider circuitry

D.LED

\*Connected to RBO of PIC

\*Resistor of 1 k is connected in series with the PIC and LED

Lab2: **Interfacing with a Sensor Device on an Embedded Computer System**

Intel Galileo Pin description

* Pin 8 (GPIO 40): strobe signal
* A3 (GPIO 54): D3 (most significant bit )
* A2 (GPIO 52) : D2
* A1 (GPIO 50): D1
* A0 (GPIO 48): D0 (least significant bit)

PIC Microcontroller

* RC3: Connected to A3
* RC2: Connected to A2
* RC1: Connected to A1
* RC0 :Connected to A0

We are setting up hardware wifi module in this project and connecting it our personal laptop through synching there IP addresss

Lab3:**Controlling an I2C Device**

1. Intel galileo => I2C interfacing

* GND => GND
* 3.3V => VCC
* A4 => SDA
* A5 => SCL

1. Intel galileo =>webcam residing on the stepper motor head

## Software design:

We have connected all the embedded components together and we are using task decomposition characteristics of the multithread and assigning each thread a different operation

## 1Initializing …………………………………………………………………………………………………………………………A1

* Here we are initialing the mutex that is used to lock the critical region of the code that access the shared variable
* We are initializing 3 different threads for 3 different operations, such as one for user input, second for polling the gesture sensor and last one for the updating the server with the values calculated from the first 2 threads
* We are destroying the created thread and mutex at last part

## 2.Thread 1:User\_thread……………………………………………………………………………………………………….A2

* We are taking inputs from the user and then processing requested operation
* We are providing the user to choose among 8 different operation to perform
* There are chances of other threads utilizing this area of code for its operations, so to avoid data race, we are applying the lock with the help of mutex at start of the function and end unlocking it at the end
* We are performing the PIC side of the program codes in this thread.

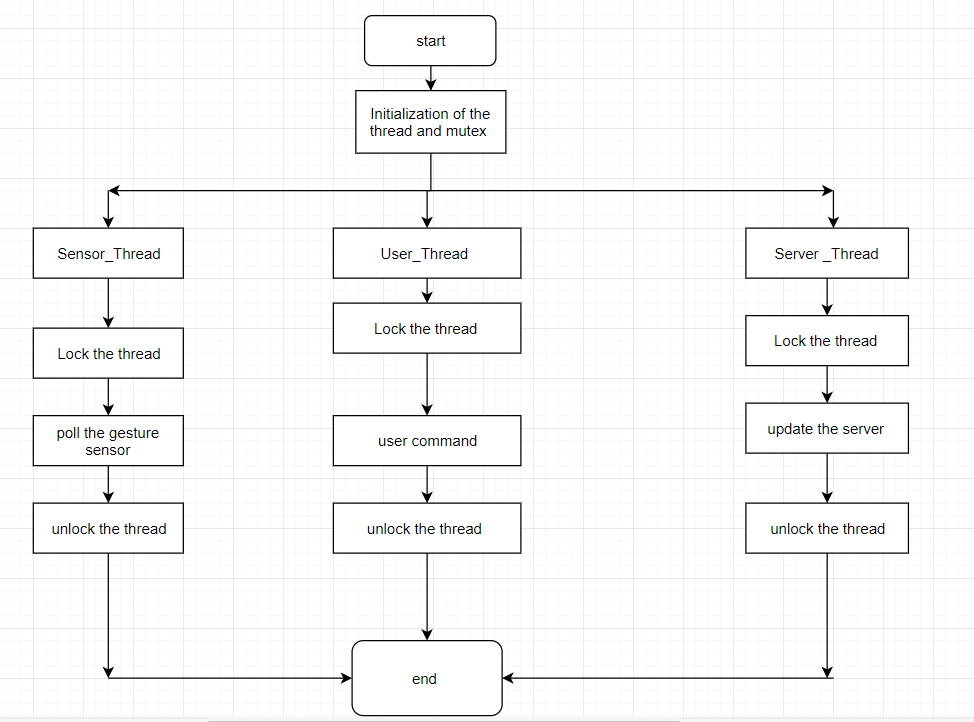
## 3.Thread 2:Sensor\_threard…………………………………………………………………………………………………A3

* We are polling the gesture value, once the value goes below the threshold value we are making the web cam which will be rotating slowly on the top of the stepper motor to stop and click a photo.
* We are using the mutex at the start of every function of the senor to get the control of the Galileo I/O pins, exclusive to the sensor thread.
* We are using malloc function to dynamically allocate the space for the images caputered whenever there is change in threshold

## 4.Thread3:Server\_thread…………………………………………………………………………………………………….A4

* We are sending the values that we are calculating in thread 1 and thread 2 to the amazon server
* We are updating the fields such as Groupid, student name, PIC adc value ,PIC status ,last update ,image file name .
* We are updating this fields every 2 seconds .

## Flow chart



*Section 8: Trouble Shooting /1 points*

Issue 1: when we designed the thread using task decomposition method i.e. By assigning different function to different threads . we were not getting the expected value ,then after reading further details about the thread operation ,we came to know that we need to use mutex in order to safe guard critical regions in the program .After applying mutex lock and unlock we were able to get the result as expected .

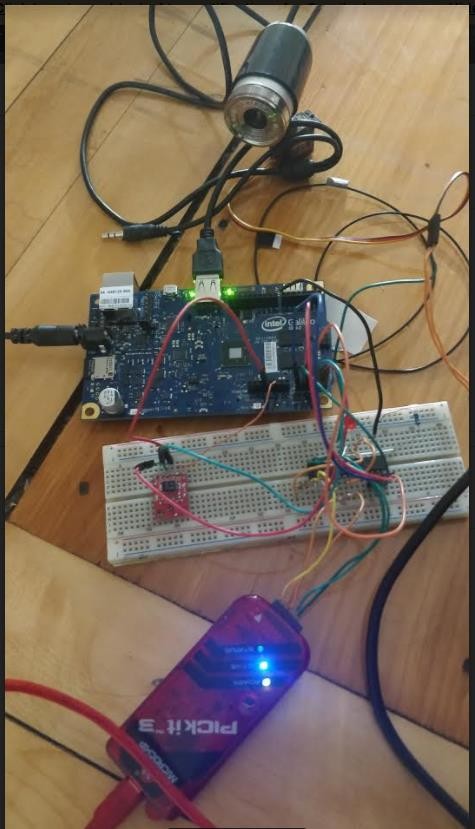
Issue 2: we planned to do the face detection, but the issue was when we included .xml file in the program and tried to run it, we were facing the issue for haar file of .xml type not found. we further studied it and found out that .xml file has to be present in the path /usr/local/OpenCv

.we added the .xml folder to that path and the error was resolved .

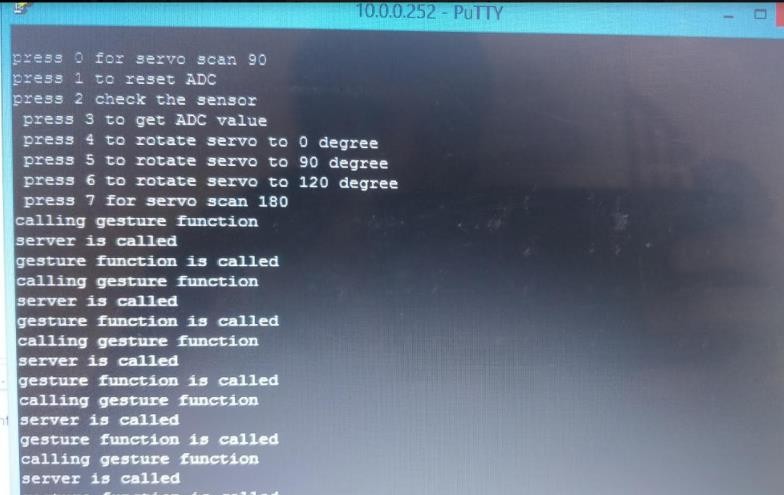
Issue3: According to the problem statement we need to take multiple photos whenever the value goes below the threshold. We initially planned to allocating an array for storing the images .But since we won’t be knowing how much photos user wants to take ,we were facing issue allocating space to the array ,so we came up with a solution of adapting malloc function of C ,which allocates memory dynamically .so the size extends on its own depending upon user input .

*Section 9: Results /0.5 points*

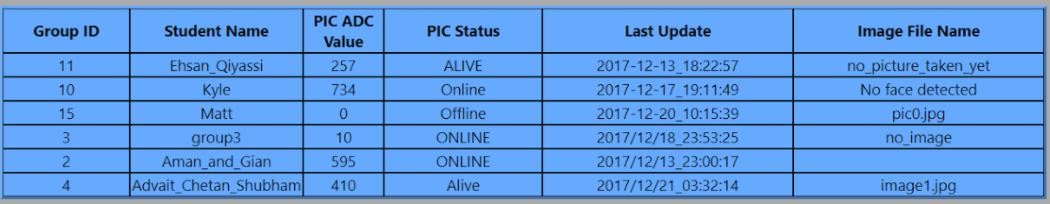
1. Setup



1. Display of threads operation



4.Server update



*Section 10: Appendix*

# A1

**//main**

**int main(void)**

**{**

**pthread\_mutex\_init(&mutex1,NULL);** //initializing the mutex thread

**int id1=1,id2=2,id3=3;**

**pthread\_t sensor,usr,svr;** //declaring the variables of pthred

**printf("calling sensor thread\n"); sleep(1);**

**pthread\_create(&sensor, NULL, sensor\_thread, (void \*)id2);** //creating the thread of sensor

**sleep(1);**

**printf("calling user thread\n"); sleep(1);**

**pthread\_create(&usr, NULL, user\_thread, (void \*)id1);** //creating the thread of user command

**sleep(1);**

**printf("calling server thread\n"); sleep(1);**

**pthread\_create(&svr, NULL, server\_thread, (void \*)id3);** //creating the thread for server command

**sleep(2); pthread\_join(sensor ,NULL);**

**pthread\_join(usr ,NULL); pthread\_join(svr ,NULL);**

**pthread\_mutex\_destroy(&mutex1);**

# }

**A2**

**void \*user\_thread(void \*arg2)** //calling the first thread

**{**

**while(1)**

**{**

**pthread\_mutex\_lock(&mutex1);** //locking the function

**high\_GPIO\_out(40);**

**Printf("press 0 for servo scan 90\n");** //accepting the user command

**printf("press 1 to reset ADC\n"); printf("press 2 check the sensor\n "); printf("press 3 to get ADC value\n ");**

**printf("press 4 to rotate servo to 0 degree\n "); printf("press 5 to rotate servo to 90 degree\n "); printf("press 6 to rotate servo to 120 degree\n "); printf("presss 7 for servo scan 180\n"); scanf("%d",&input);**

**pthread\_mutex\_unlock(&mutex1); pthread\_exit(NULL);**

**pthread\_mutex\_lock(&mutex1);**

**switch(input)** //performing the PIC side of the program,

**{**

**case 0:**

**{**

**// i2c(); break;**

**}**

**case 1:**

**{**

**reset\_ADC(); break;**

**}**

**case 2:**

**{**

**chk\_sensor(); break;**

**}**

**case 3:**

**{**

**get\_ADC(); break;**

**}**

**case 4:**

**{**

**servo\_0(); break;**

**}**

**case 5:**

**{**

**servo\_90(); break;**

**}**

**case 6:**

**{**

**servo\_180(); break;**

**}**

**default:**

**{**

**printf("please press number between 1-6"); break;**

**}**

**}**

**pthread\_mutex\_unlock(&mutex1); pthread\_exit(NULL);**

**}**

# A3

**void \*sensor\_thread(void \*arg2)** //calling the sensor thread

**{**

**while(1)**

**{**

**pthread\_mutex\_lock(&mutex1); printf("calling gesture function\n"); sleep(1);**

**int gesture();**

**printf("gesture function is called\n"); sleep(1);**

**sleep(0.1); pthread\_mutex\_unlock(&mutex1);**

**}**

**}**

# A4

**void \*server\_thread(void \*arg3)** //calling the server thread

**{**

**pthread\_mutex\_lock(&mutex1);**

**//const char\* hostname="ec2-54-202-113-131.us-west-2.compute.amazonaws.com"; // Server Hostname or IP address**

**const char\* hostname="10.0.0.237"; // Server Hostname or IP address const int port=8000; // Server Service Port Number**

**const int id=04;**

**const char\* password="password"; const char\* name="Bhau\_cha\_dhakka"; int adcval=adc\_value;**

**printf("Server adc val is %d",adc\_value); const char\* status="Alive";**

**// const char\* timestamp="2017-12-10\_10:10:44"; const char\* timestamp=time\_stamp();**

**const char\* filename="image1.jpg"; // captured picture name + incremented file number char buf[1024];**

**snprintf(buf,1024,"http://%s:%d/update?id=%d&password=%s&name=%s&data=%d&status=%s&timestamp=% s&filename=%s",hostname,port,id,password,name,adcval,status,timestamp,filename);**

**FILE \*fp;**

**struct stat num; stat(filename, &num); int size = 0; num.st\_size;**

**char \*buffer = (char\*)malloc(size); fp=fopen(filename,"rb");**

**int n=fread(buffer,1,size,fp);**

**HTTP\_POST(buf, buffer, size);**

**// fclose(fp);**

**// return 0; pthread\_mutex\_unlock(&mutex1);**

**}**

## References

1. Linux I2C library documentation, https://[www.kernel.org/doc/Documentation/i2c/dev-interface](http://www.kernel.org/doc/Documentation/i2c/dev-interface)

1. Temperature sensor, ht[tps://w](http://www.sparkfun.com/products/11931)ww.sp[arkfun.co](http://www.sparkfun.com/products/11931)m[/products/11931](http://www.sparkfun.com/products/11931)
2. Gesture sensor, https://[www.sparkfun.com/products/12787](http://www.sparkfun.com/products/12787)