Course: Microprocessors Design II and Embedded Systems

Course #: EECE 4800/5520

Course title: Parallel computing of embedded sensors using Multithreading programming

Instructor: Yan Luo

Group #: 12

Student name: Zubair Nadaph

Hand in Date: 12/21/17

Lab Due Date: 12/21/17

Section 2: Contributions

1. Group Member 1 – Zubair Nadaph

Worked on getting all the codes until lab4 together. Altered the pic code to just receive the ADC value and send it to galileo.

2. Group Member 2 - Aravind Dhulipalla

Worked on configuring the I2C communication between the intel Galileo Gen2 and Gesture sensor APDS-9960. Debugging the codes.

3. Dhushyanth Kadari

Worked on configuring the I2C communication between the intel Galileo Gen2 and Temperature sensor TMP102. Debugging the codes.

Section 3: Purpose

It focuses on multithreading programming. In this lab one is supposed to develop user command interface (taking option from user for the sensors), activating the sensors (gesture, temperature and ldr) and updating the data on server. With the following set of activities, this lab allows an individual to understand importance of multithreading and the efficient way of designing it.

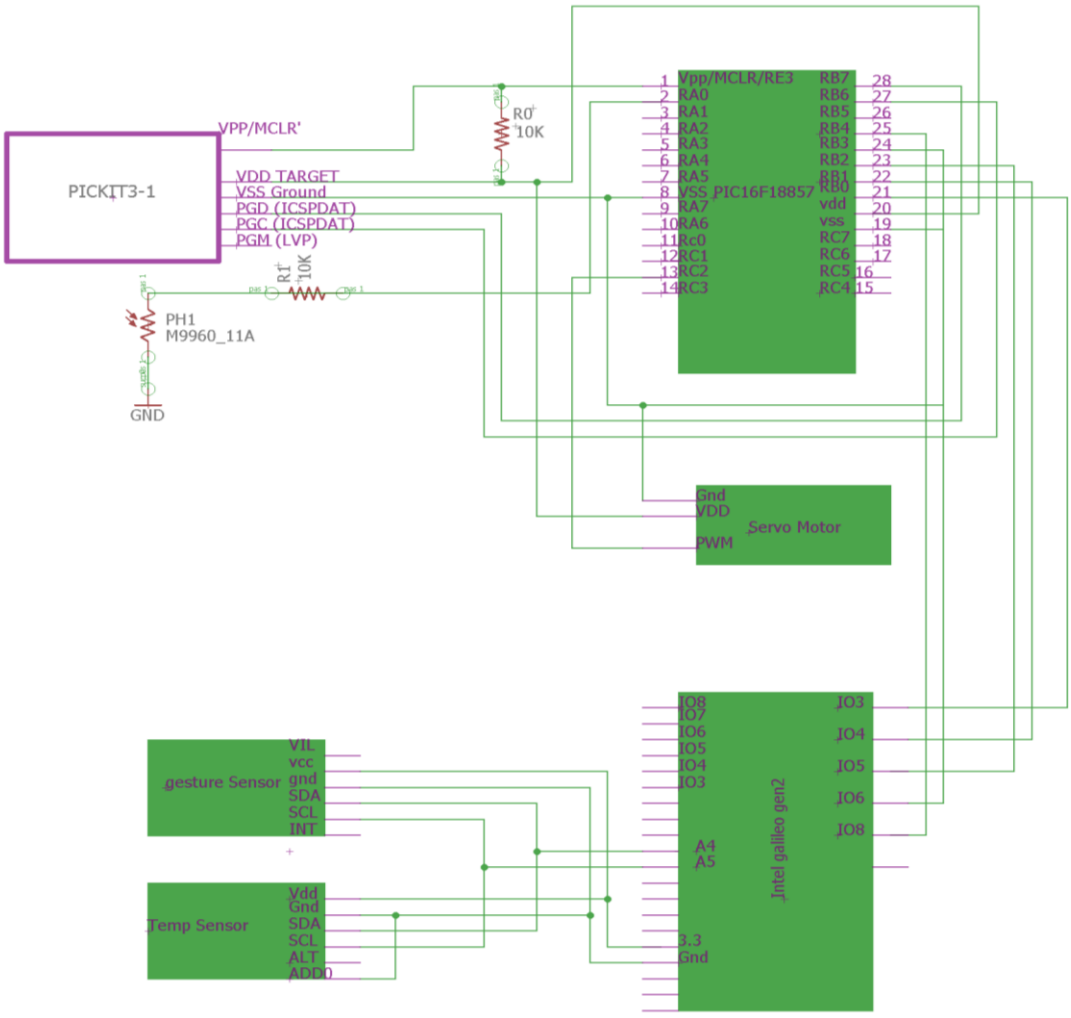
Section 4: Introduction

According to definition of it thread consist of sequence of code meant to do specific task, and multiple threads designed in a process can perform their respective task independently. Threading is divided into ‘concurrency’ and ‘parallelism’. Concurrency occurs when two or more threads run simultaneously on a single core and parallelism occurs when the same event occurs but on different cores of same processors. The design concepts of multithreading are:

1. Threading for functionality: it is assigning different threads to different functions of the application, reducing the chances of overlapping and easier control of execution of concurrent functions.
2. Threading for performance: it is to thread serial code in the application to increase the performance.
3. Decomposing the work: It is logic breaking of the application into individual tasks and looking for dependencies.

Based on ‘decomposing the work’ this lab is divided into 3 threads named as interface, sensor and client. All the threads run continuously with interface thread designed to give user options to do activities like reset the pic, get the recent ADC value, turn the servo motor either by certain degrees or keep scanning continuously and finally change the threshold value of sensor. Second thread ‘sensor’ thread reads the values from the sensors when the option of sensor reading is selected and finally the ‘client’ thread updates the recent value of ADC, most recent picture taken by camera, time and date to server. Since the threads have few common variables hence the concept of mutex locking and unlocking is used which avoids the use of same part of code or variable simultaneously by two or more threads, details of which has been explained under procedure section.

Section 6: Schematics



Section 5: Materials, Devices and Instruments

1. Intel Galileo Gen2
2. APDS 9960 (sensor)
3. Pic16f18857
4. Pickit 3
5. LDR
6. Webcam
7. FTDI cable
8. Putty and Winscp software
9. MPLAB X IDE
10. XC8 compiler
11. 10k and 220 ohm resistor
12. Servo motor – 2.5 Kg-cm, 4.8 – 6 V
13. Breadboard and jumper wires
14. 10K and 220 ohm resistors
15. Micro SD card and adaptor
16. Notepad C++ editor
17. Analog Discovery2- Xilinx

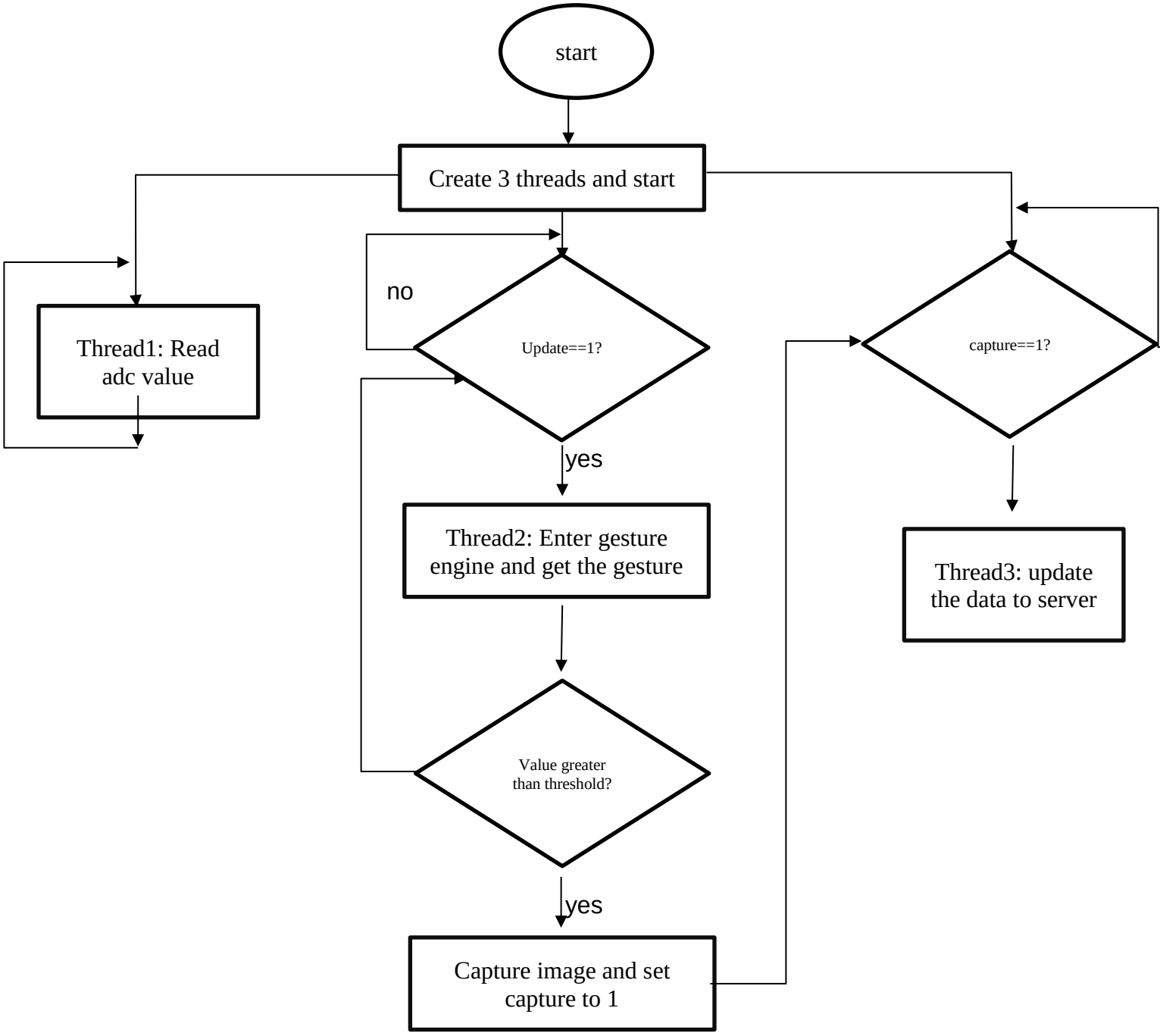
Hardware design:

Section 7: Lab Methods and Procedure /2 points

The aim of this lab was to club all the labs until now and implement them using multithreading. In this lab just like lab2 the IO pins of Galileo were connected to pic16f18857 with pin 3, 4, 5 and 6 connected to portA pins 1, 2, 3 and 4 as data bus and pin 7 of Galileo was connected to port A pin 5 for handshaking strobe signal. The idea was to communicate the message between two using the bus and in synchronism with strobe signal. Since the pic16f18857 was used to keep rotating the camera and get the ADC value the connections of the servo and LDR were at pin 0 of port A and pin2 of port C respectively. A 10k ohm resistor was used as voltage divider circuit for LDR (to get Analog value) with one end of voltage divider circuit tied to 5 V and another end gnd. For gesture part A4 and A5 of intel Galileo (SDA and SCL resp.) were connected to SDA and SCL of gesture sensor and it was made sure that the power and ground were connected to. As for power and ground pickit 3 provided 3.3 V to pic16f18857, gesture sensor takes power from Galileo board (3.3V), the servo is powered using external power supply supplying 5 V.

Software design:

As already said that this lab comprises of 3 threads, the first of which takes the input from user, a thread named ‘interface’ is created. Under this thread a forever while loop prints option #s from 1-6 along with description as mentioned earlier and based on the response from user the integer value is then taken in the form of case and the lines within the case are executed. Since the code is designed to send the int (value-1) given by the user to pic and gesture, it is required to set the gpio linux drivers which was done in lab2. In second thread a forever while loop runs the gesture engine, but before getting into the engine it checks flag bits ‘update’ which it gets from the thread 1 and runs the gesture engine only if it’s integer value is equal to 1. Since ‘update’ is taken by thread one hence it is required to avoid race scenario, this is achieved using mutex locking the line for it is: “pthread_mutex_lock(&mutex)”, it is unlocked at the end of thread. This locking and unlocking is also done in thread 1. When the gesture is detected another flag ‘capture’ is set high this is done to update the captured image onto server. Again, mutex locking and unlocking is done in this thread. Finally, the client thread checks the value of flag ‘capture’ and if set high updates the values to server. Under this code the sprintf updates a string variable ‘buf’ with ip address or host name, password, name, ldr value, time/date and name of the recent image taken. For this a http function (provided that the curl header files are included) taking ‘buf’, buffer and size of it are written. The detail flow charts of the threads are:


start
Create 3 threads and start
Thread1: Read adc value
no
Update==1?
yes
Thread2: Enter gesture engine and get the gesture
Value greater than threshold?
yes
Capture image and set capture to 1
capture==1?
Thread3: update the data to server

Section 8: Troubleshooting

1. Stopping one thread for the result of other thread:

We used mutex to handle shared variables. When mutex was not used it just gave out garbage values for gesture and failed to take images the reason being flag ‘update’ being used for thread 1 and 2. This same was also done for flag ‘capture’, generated in thread2 and used in thread1.

2. Servo motor failure:

We failed to get the servo running, despite using prototype function. At first it seemed the pulse was not coming out of the portA pin2 but when checked using logic probe seemed to be working fine. This was the part which we could not get it working at all.

Section 8: code

Not all of code has been included for this lab as it was clubbing previous labs, however the code related to threading is included.

```
void *Interface(void *Interfaceid)
{
  int cmd,a,adc,data;
   while(1)
  {
              char a = getchar();
              if(a=='\n')
              {
                        printf("Enter pressed");
              pthread_mutex_lock(&mutex);
              update = 1;
              pthread_mutex_unlock(&mutex);
              printf("Give any one of the command \n
1.Reset 2.Ping 3.PIC LDR VALUE 4.TURN 30
5.TURN 90 6.TURN 120 7.Temperature\n");
     scanf("%d",&cmd);
              //make the strobe high
  switch(cmd)
  {
    case 1:
               Export();
               SetGPIO_output();
                        system("echo 1 >
/sys/class/gpio/gpio40/value");
                        system("echo 0 >
/sys/class/gpio/gpio6/value");
                        system("echo 0 >
/sys/class/gpio/gpio0/value");
                        system("echo 0 >
/sys/class/gpio/gpio1/value");
                        system("echo 0 >
/sys/class/gpio/gpio38/value");
                        usleep(10000);
                        system("echo 0 >
/sys/class/gpio/gpio0/value");
                  UnExport();

0 > /sys/class/gpio/gpio1/value");
                        system("echo 0 >
/sys/class/gpio/gpio38/value");
                        usleep(10000);
                        system("echo 0 >
/sys/class/gpio/gpio40/value"); Export();
                  SetGPIO_Input();
                        system("echo 1 >
/sys/class/gpio/gpio40/value");
```

```
                a = read_gpio();
                usleep(10000);
                system("echo 0 > /sys/class/gpio/gpio40/value");
        UnExport();
                if(a!=ACK)
                {
                        printf("pic not available");
                }
                break;
                case 2:
                Export();
        SetGPIO_output();
                system("echo 1 > /sys/class/gpio/gpio40/value");
                system("echo 1 > /sys/class/gpio/gpio6/value");
                system("echo 0 > /sys/class/gpio/gpio0/value");
                system("echo 0 > /sys/class/gpio/gpio1/value");
                system("echo 0 > /sys/class/gpio/gpio38/value");
                usleep(10000);
        system("echo 0 > /sys/class/gpio/gpio40/value");
                UnExport();
                Export();
        SetGPIO_Input();
                system("echo 1 > /sys/class/gpio/gpio40/value");
                a=read_gpio();
                usleep(10000);
                system("echo 0 > /sys/class/gpio/gpio40/value");
        UnExport();
                if(a!=ACK)
                {
                        printf("pic not available");
                }
                break;
                case 3:
                Export();
        SetGPIO_output();
                system("echo 1 > /sys/class/gpio/gpio40/value");
                system("echo 0 > /sys/class/gpio/gpio6/value");
                system("echo 1 > /sys/class/gpio/gpio0/value");
                system("echo
                UnExport();
                Export();
        SetGPIO_Input();
                system("echo 1 > /sys/class/gpio/gpio40/value");
                a = read_gpio();
                usleep(10000);
                system("echo 0 > /sys/class/gpio/gpio40/value");
                if(a==ACK)
                {
                system("echo 1 > /sys/class/gpio/gpio40/value");
                int data = read_gpio();
                sleep(0.01);
                system("echo 0 > /sys/class/gpio/gpio40/value");
                system("echo 1 > /sys/class/gpio/gpio40/value");
                data = data | (read_gpio()<<4);
                sleep(0.01);
                system("echo 0 > /sys/class/gpio/gpio40/value");
                system("echo 1 > /sys/class/gpio/gpio40/value");
                data = data | (read_gpio() << 8);
                sleep(0.01);
                system("echo 0 > /sys/class/gpio/gpio40/value");
                UnExport();
                pthread_mutex_lock(&mutex);
                ldrvalue = data;
                printf("%d\n",data);
                pthread_mutex_unlock(&mutex);
                }
                else
```

```
                {
                        printf("pic not found");
                        update = 0;
                }
                break;
                case 4:
                                Export();
        SetGPIO_output();
                system("echo 1 > /sys/class/gpio/gpio40/value");
                system("echo 1 > /sys/class/gpio/gpio6/value");
                system("echo 1 > /sys/class/gpio/gpio0/value");
                system("echo 0 > /sys/class/gpio/gpio1/value");
                system("echo 0 > /sys/class/gpio/gpio38/value");
                usleep(10000);
                system("echo 0 > /sys/class/gpio/gpio40/value");
                UnExport();
                Export();
        SetGPIO_Input();
                system("echo 1 > /sys/class/gpio/gpio40/value");
                a = read_gpio();
                sleep(0.01);
                system("echo 0 > /sys/class/gpio/gpio40/value");
        UnExport();
                break;
                case 5:
                                Export();
        SetGPIO_output();
                system("echo 1 > /sys/class/gpio/gpio40/value");
                system("echo 0 > /sys/class/gpio/gpio6/value");
                system("echo 0 > /sys/class/gpio/gpio0/value");
                system("echo 1 > /sys/class/gpio/gpio1/value");
                system("echo 0 > /sys/class/gpio/gpio38/value");
                usleep(10000);
                system("echo 0 > /sys/class/gpio/gpio40/value");
                UnExport();
                Export();
        SetGPIO_Input();
        system("echo 1 > /sys/class/gpio/gpio40/value");
                a = read_gpio();
                usleep(10000);
                system("echo 0 > /sys/class/gpio/gpio40/value");
        UnExport();
        if(a!=ACK)
                {
                        printf("pic not ready");
                }
                break;
                case 6:
                                Export();
        SetGPIO_output();
                system("echo 1 > /sys/class/gpio/gpio40/value");
                system("echo 1 > /sys/class/gpio/gpio6/value");
                system("echo 0 > /sys/class/gpio/gpio0/value");
                system("echo 1 > /sys/class/gpio/gpio1/value");
                system("echo 0 > /sys/class/gpio/gpio38/value");
                usleep(10000);
        system("echo 0 > /sys/class/gpio/gpio40/value");
                UnExport();
                Export();
        SetGPIO_Input();
        system("echo 1 > /sys/class/gpio/gpio40/value");
        a = read_gpio();
                usleep(10000);
```

```
                    system("echo 0 > /sys/class/gpio/gpio40/value");
            UnExport();
                    if(a!=ACK)
                    {
                            printf("pic not ready");
                    }
                    break;

                    case 7:

                    unsigned char  Temp = Temperature();
                    break;


   }
        sleep(2);
         }
 }
}
 Thread2:

void *Sensors(void *Sensorsid)
{
        while(1)
        {
        pthread_mutex_lock(&mutex);//since update is used in thread 1 as well
                int cmd = update;
                pthread_mutex_unlock(&mutex);
                                if(update == 0)
                        {
              unsigned char Temp_value = Temperature();
                        //char *dev = "/dev/i2c-0";

        pthread_mutex_lock(&mutex);
             fd = open(dev, O_RDWR );
       int i,r;
                        int addr = 0x39;
                        if(fd < 0)
                        {

        perror("\nOpening i2c device node\n");
                        }
                        r = ioctl(fd, I2C_SLAVE, addr);
                        if(r < 0)
                        {

        perror("\nSelecting i2c device\n");
                        }
                        gesture_enable();
                        r = APDS9960_write(0x80,0x4D);
                        if(r<0)
                        {

        perror("\ngesture engine not started\n");
                        }
                        printf("\ngesture engine started\n");
                        usleep(delay);
                        unsigned char value = read_gesture();

        if(!APDS9960_write(0xAB,0x00))
                        {
                                printf("Error during write to sensor");
                        }

        if(!APDS9960_write(0xE7,0x00))
                        {
                                printf("Error during write to sensor");
                        }

        if(!APDS9960_write(0x80,0x00))
                        {
                                printf("Error during write to sensor");
                        }
                        if((Temp_value>20) ||(value == UP))
                        {
```

```
                                        Imagecapture();
                                        cout<<"Gesture Recognised and Picture taken" << endl;

        pthread_mutex_lock(&mutex2);
                                        capture = 1;

        pthread_mutex_unlock(&mutex2);
                                }
                                else
                                {
                                        cout <<"Gesture Not Correct or Recognised" << endl;
                                }
                                close(fd);
      pthread_mutex_unlock(&mutex);
                                }
        }


}
Thread3:
void HTTP_POST(const char* url, const char* image, int size){
        CURL *curl;
        CURLcode res;

        curl = curl_easy_init();
        if(curl){
                curl_easy_setopt(curl, CURLOPT_URL, url);
          curl_easy_setopt(curl, CURLOPT_POST, 1);
                curl_easy_setopt(curl, CURLOPT_POSTFIELDSIZE,(long) size);
          curl_easy_setopt(curl, CURLOPT_POSTFIELDS, image);
                res = curl_easy_perform(curl);
                if(res != CURLE_OK)
                        fprintf(stderr, "curl_easy_perform() failed: %s\n",
                        curl_easy_strerror(res));
                curl_easy_cleanup(curl);
        }
}

char *time_stamp(){

char *timestamp = (char *)malloc(sizeof(char) * 16);
time_t ltime;
ltime=time(NULL);
struct tm *tm;
tm=localtime(&ltime);

sprintf(timestamp,"%04d%02d%02d%02d%02d%02d", tm->tm_year+1900, tm->tm_mon+1,
    tm->tm_mday, tm->tm_hour-5, tm->tm_min, tm->tm_sec);
return timestamp;
}

void *Client(void *clientid)
{
        while(1)
        {
        if(capture ==1)
        {
        printf("sending pic value\n");
        const char* hostname="ec2-54-202-113-131.us-west-2.compute.amazonaws.com"; // Server Hostname or IP address
        const int   port=8000;                  // Server Service Port Number
        const int   id=12;
        const char* password="password";
        const char* name="Zubair";
        const int   adcval=ldrvalue;
        const char* status="HelloAll";
        const char* timestamp=time_stamp();
        char* filename="img.jpg";  // captured picture name + incremented file number
        //fgets(buffer,100,stdin);
        //filename = (char *)malloc(strlen(buffer)+1);
        //strcpy(filename,buffer);

        char buf[1024];
```

```
sprintf(buf,"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);
    //.....
    // use sprintf() call here to fill out the data "buf":
     // use the provided URL Protocol in the lab description: replace the "server_hostname", "portnumber", "var_xxxx" with the related format specifiers "%d" or "%s"
    //.....



        // ========== Don't bother the lines below
        FILE *fp;
        struct stat num;
        stat(filename, &num);
        int size = num.st_size;
        char *buffer = (char*)malloc(size);
        //fp = fopen(filename,"rb");
        //int n = fread(buffer, 1, size, fp);
   // ========== Don't bother the above lines
        HTTP_POST(buf, buffer, size);
        fclose(fp);
        pthread_mutex_lock(&mutex2);
        capture = 0;
        pthread_mutex_unlock(&mutex2);
        }
        }
}



int main(void)
{
        pthread_mutex_init(&mutex,NULL);
        pthread_mutex_init(&mutex2,NULL);
        pthread_t thread_client,thread_Interface,thread_Sensors;
        pthread_create(&thread_Interface,NULL,Interface,NULL);
        pthread_create(&thread_Sensors,NULL,Sensors,NULL);
        sleep(0.01);
        pthread_create(&thread_client,NULL,Client,NULL);
        pthread_join(thread_Interface,NULL);
        pthread_join(thread_Sensors,NULL);
        pthread_join(thread_client,NULL);
        pthread_mutex_destroy(&mutex);
        pthread_mutex_destroy(&mutex2);
        return 0;

}
```

Section 10: Result

The result of threading operations are as follows:

Group ID	Student Name	PIC ADC Value	PIC Status	Last Update	Image File Name
11	Ehsan_Qiyassi	257	ALIVE	2017-12-13_18:22:57	no_picture_taken_yet
10	Kyle	734	Online	2017-12-17_19:11:49	No face detected
12	Zuber_arvind	512	Alive	2017/12/21_16:30:51	imag8.jpg
15	Matt	0	Offline	2017-12-20_10:15:39	pic0.jpg
3	group3	10	ONLINE	2017/12/18_23:53:25	no_image
2	Aman_and_Gian	595	ONLINE	2017/12/13_23:00:17	
4	Advait_Chetan_Shubham	410	Alive	2017/12/21_03:32:14	image1.jpg