# Temperature Data Logger

ECE 4220 Real Time Embedded Systems

Final Project Report

Spring 2014

Michael Brauch

12347562

Instructor: Dr. DeSouza

## Abstract

This project was designed to be a temperature data logger that allows a user to not only measure the current temperature, but also the average temperature and change in temperature with respect to time. The project will enable users to obtain and monitor accurate temperature data for any applications that require precise readings of rapidly changing temperature environments. The motivation behind this concept is if you could place multiple temperature data loggers in several locations then you could theoretically use this data to have more control over climate monitoring and control. By connecting multiple of these devices to a central application, it would not only allow it more complex temperature measurements at multiple locations, but would allow the user to view these temperature changes through the user interface from a remote location. Applications could include ones such as heating/cooling systems or geological temperature surveying where temperatures are fluctuating rapidly and need to be recorded. To demonstrate how this concept works, this project aims to interface a single temperature data logger with a computer via a remote TCP connection. The project implements the following key concepts that we have discussed in class: **real time tasks, sockets, p-threads, and mailboxes.**

## Introduction

This project was originally intended to only display temperature readings obtained by the microcontroller onto the LCD interface and store them into a USB flash drive. However, my final design resulted in not only doing this, but also enabling the user to communicate with this process through Ethernet over a TCP socket connection, enabling an enhanced and remote user interface. Through this improvement, we not only allow the user to have more convenient readings from a remote computer, but also allow them to have control over how long the process runs and how often to take measurements. It also allows the finalized temperature averages and change in temperature rates to be displayed to the user from any computer that has an Ethernet connection. This project also addresses the issue of communication between two different programming languages. C/C++ is a more technical language that I found more suited to the programming of hardware, whereas Java tends to be a more robust language oriented towards graphical interfaces. Being able to utilize both would allow for a better user interface and more control over hardware elements. To approach this problem, I made use of a TCP connection to allow communication between these different languages via a common buffer. Overall, the main objective of the project is to create a remote application for allowing users to accurately monitor temperature changes with relation to time.

## Background

### Application

The main purpose of this project is to provide accurate and purposeful temperature readings over time to a user that is able to be controlled from a remote location. To do so, the controller is connected to a Java application via Ethernet to allow the user to control the unit as well as to visualize the data received from the sensor in a helpful manner. It also allows for people at the location of the unit to view the data on a LCD interface, and have the data written to a removable storage device (flash drive).

### Problem Statement

The project has two issues that it addresses, including inter process communication between multiple threads that control the hardware components as well as remote communication between the unit and the user. Each of the hardware components must be communicating with each other in real time to properly interpolate the data from the sensor as well as display it to the user. The program must also be able to successfully communicate between a C++ program controlled by the microcontroller and a Java application that displays the user interface.

## Proposed Implementation

### Hardware:

One of the main challenges with the project is to implement multiple hardware elements and have them work together to provide more convenient and visual data for the user. Below is the list of components that were implemented in this project:

* MBED LPC1768 Microcontroller
* HD44780 20x4 White Text on Blue Background LCD Interface
* TMP386 Temperature Sensor
* PRT-08535 – RJ45 MagJack-Compatible Ethernet Port
* USB Type A Female Breakout Board
* COM-09151 – Speaker – 0.5 W( 8 ohm)

We will first talk about the main hardware component of the project: the MBED LPC1768 microcontroller. The MBED microcontroller is an ARM microcontroller development board designed for rapid prototyping. Since this project had many different hardware components, the MBED was the clear choice. It is designed to support many different analog/digital devices and peripheral interfaces, and includes many libraries to help you do so. It is also conveniently designed with through-hole PCBs for breadboards and has a unique and easy to use USB flash programmer that allows you to download the binary file from their online compiler (located at mbed.org) and drag it to the USB drive it is connected to. Also, the main reason I chose it over other microcontrollers is it has real-time capabilities and supports multi-threading. The MBED Real-Time Operating System is based off of the RTX Real-Time Operating System which uses the CMSIS-RTOS API open standard for scheduling tasks. It runs at approximately 96 MHz, with 32KB RAM and 512KB FLASH memory. Below is the pin layout for the board, which accurately portrays the amount of devices it is capable of interfacing with.

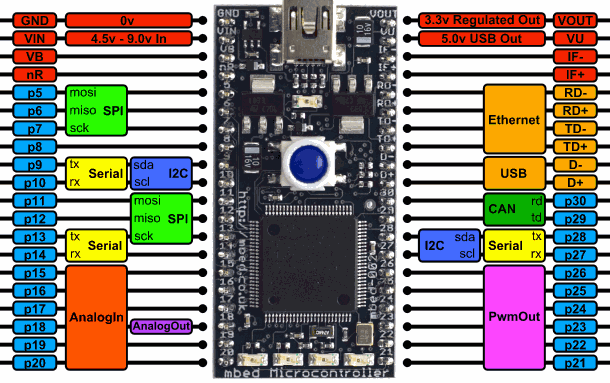


Figure 1 - MBED Microcontroller (mbed.org)

The HD44780 LCD interface is responsible for displaying all the data that is periodically being collected from the real-time tasks inside the MBED. While the timer is running, it will display the current time, the current temperature, and the average change in temperature. It was displays the temperatures in both Celsius and Fahrenheit for the convenience of the user. It also keeps track of the timer and displays the final information to the user. It is a 20x4 interface, with blue background and white text and has a backlit display. This is powered by the MBED’s VU pin which is ~5V. The TMP386 is the temperature sensor that collects all of the data and is connected to the MBED’s analog pin 17 and powered by the VOUT pin which is ~ 3.3V. It provides a voltage output that is linearly proportional to the Celsius temperature with a scale factor of 10 mV/C. The PRT-08535 is the Ethernet port that is responsible for connecting the board to the Java application through TCP. It is powered by the CAT5 Ethernet cable and is connected to MBEDs 4 Ethernet pins. USB Type A Female Breakout Board is responsible for connecting the external flash drive device to the MBED through the 2 USB pins. It is powered by the VU pin is ~5V. Finally, we have the COM-09151 speaker which will play a noise every time a temperature is read by the TMP386 as well as when the application is ended. It is hooked up to the following OP-AMP circuit which is powered by VU.

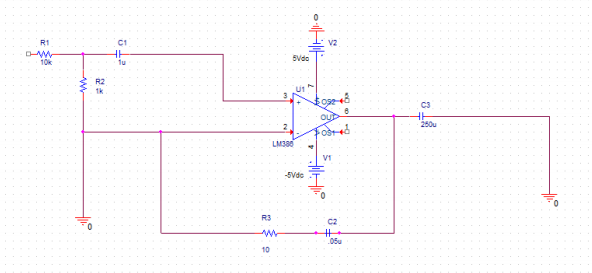


Figure 2 - OP-AMP Circuit for Speaker

It is also worth mentioning that the MBED has 4 built in LED lights that, along with the speaker, will turn on each time a temperature is read. Below is the Function Body Diagram that described communication between each of the hardware components.



Figure 3 - FBD of Hardware Components

### Software:

This project consisted of two programs: a C++ program for the microcontroller and a Java application for the user interface. These two programs communicate with eachother over a TCP socket. We will first talk about the C++ portion which includes all the concepts from class.

#### C++ Program

This program is made on the MBED online compiler and uses many of the built in MBED libraries including libraries for Ethernet interfaces, text LCD’s, and RTOS. I used the USB connection to communicate with the COM port on the computer to display debugging information seen in the results section.

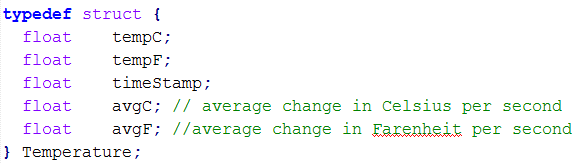
Initially, this program has to map all of the hardware components mentioned in the hardware section to the corresponding pins. Conveniently, the controller does not need memory addressing or bitwise manipulation to map each component, which makes it quite nice to work with when multiple hardware components are involved.

The main thread will initialize the flash drive and display all files on the flash drive to the COM port on the computer. It will then setup a TCP socket where the MBED controller is the server, and print out the IP address of the board. This only works if you are connected to Ethernet, otherwise an error message will display. Once this socket is set up, it will wait for the Java client to establish a connection to it. On the LCD it will display “Wait for connection...” until a proper connection is established. Once this happens, the LCD will display both the IP of the board and the IP of the computer running the Java application. For this project I used a default port of 20 but any port could be used. When this is done, it will wait for the run time and time interval from the Java application and, once received, proceed to initialize all of the threads, the real time task (setting the period to the time interval received), and start the timer. This program runs off of 5 p-threads and 1 real-time task. Below I have a diagram of the communication between each of the threads.



Figure 4 - Thread Communication

The Temperature Read Thread is the real-time task which periodically reads from the temperature sensor. The data is set up in a Temperature structure which is defined as follows:

 Every time the real-time task executes, it will allocate memory for a piece of mail of type Temperature, which is very similar to a FIFO (essentially an allocatable queue), Once this is done, it will pass the mail and shared buffer into the function calcTemp() to calculate the data received from the temperature sensor and store it on the mail and buffer. It will also signal the semaphore for the LED and speaker thread. It then puts this mail onto the Mailbox. Also, you must remember to free this memory after you put it onto the mailbox, otherwise you will likely run out of memory. The USB Write Thread will have already called mail\_box.get() which will block until the Temperature Read Thread has put this data onto the Mailbox. Once this has happened, it will write the data to the text file “Temperature\_Log.txt” on the flash drive, and also print the data through the COM serial port. The Speaker Thread will play a noise through the speaker every time the Temperature Thread signals the semaphore. In addition to the Mailbox, the Temperature Read Thread also put this information into a shared buffer, which is accessible by the LCD thread and the TCP Send Thread. The LCD thread will continually update the LCD interface with the current temperatures and average changes in temperatures from this buffer and the current time from the timer. Once the timer has reached the total time to run, it will set a Boolean to false notifying the program that it is complete. It then sends the final data to the Java application and prints the final data on the LCD. The TCP Send Thread, which is the main thread, will constantly be sending data from the shared buffer to the Java application through a TCP socket connection. It sends the data in the format “,temperature,timestamp\n” using the fnprint() function (which takes in binary data and converts it to a character array). I put the ‘,’ characters between the numbers so that the Java application can split the character array successfully as discussed later on in the report. Also, we need to include the /n character so that the Java application knows when to stop receiving. This thread will be sending this character array through the socket using the function send\_all(). Finally, the Pause Thread will continually wait for information from the Java application through the socket connection. The only commands that this thread can receive are pause, resume, and stop. If pause is received, it will stop the timer and set a global Boolean variable to true, informing all of the other threads to stop execution and letting the main thread know to delete the real time task. Once resume is received, the real time task will be recreated, the Boolean variable will be set to false informing the threads to resume execution, and it will start the timer. The stop command will set a Boolean variable to false to tell all threads to quit execution, and for the final results to be displayed. This thread is receiving data using the function receive().

#### Java Application

This program will start by initializing the user interface by using the Java JFrame libraries. It will first ask the user for the IP address of the board and what port number they are using. If an incorrect IP or port number is entered, then the Java application will prompt the user to enter again. It will then ask for the total run time and a time interval for temperature readings in seconds which will be the period of the real time task. When start is pressed, it will send the data collected to the C++ program and the C++ program will initialize the threads and real-time task with the received times. The Java application will then continuously receive the temperature and time from the C++ program to display on the screen. There is also a pause/resume button to pause the timer and reading of temperatures as well as a stop button if you wish to completely stop the program. This works by sending a String through the TCP connection containing “resum”, “pause”, or “stopn”. If stopped, the program will then wait for the final data from the C++ program and display it to the user.

## Flow of the Programs:

## C++ program initializes Flash Drive and Ethernet Port. Waits for TCP Connection.

## Java application asks for IP Address and Port Number. Connects to C++ program.

## Java application requests Run Time and Time Interval. Sends to the C++ program.

## C++ program receives Run Time and Time interval. Initializes real-time task and threads.

1. Each time Real-Time Task executes, write to flash drive via Mailbox, flash LEDs, and play noise through speaker.
2. Upon reaching the run time (or user presses Stop), C++ program displays finalized data on LCD and sends to the Java Application.
3. Java Application displays finalized data.

## Results

In this section, we will go through step by step the results I received from my test as well as any errors I have discovered while running the programs.

First, I loaded the binary file onto the MBED microcontroller and hit the reset button. Through the COM port, the following message appeared:

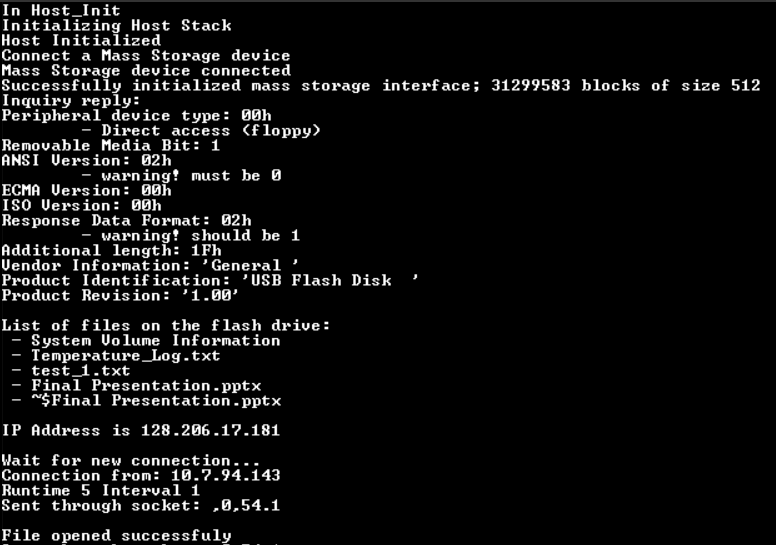


Figure 5 - Initializing the Program

As you can see, it setup the USB Flash Disk and displayed all of the files on the flash drive. For this project, Temperature\_Log.txt is where all the information will be stored. It then proceeds to display the IP Address of the MBED which is this case is 128.206.17.181. It then proceeds to wait for a connection from the Java application. When a connection is received it will, take the runtime and interval from the Java application and start the real time task. As you can see above, one item was sent through the socket, and the USB file was opened successfully.

In the Java application, on startup it displays the following prompts:

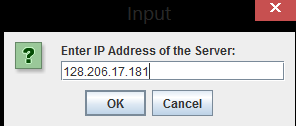


Figure 6 - Prompt for IP Address of Server

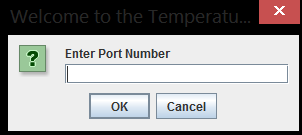


Figure 7 - Prompt for Port Number

Note that if an incorrect IP address is input, then a pop-up window will show saying that the information is incorrect and go through these prompts again. Once the correct IP and port are obtained, you will continue to the main application as shown below:

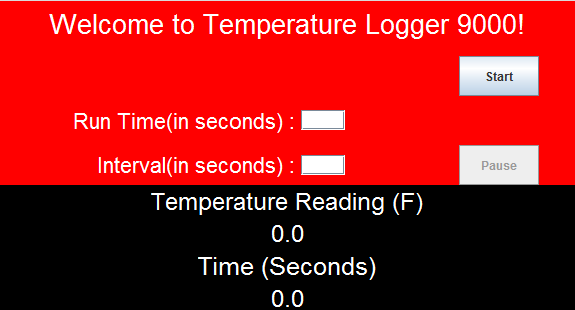


Figure 8 - Start Screen of Application

You will need to enter a run time and interval, and then press the start button. After this, both programs will start running and displaying the data. The minimum interval used was 1 second.

As proof of the data being sent through the socket, I am printing each time I send the current time in seconds and temperature to the Java application. Since it is rapidly being sent, you will notice that multiple of the same pieces of data are being sent. In my code, I did mess with wait statements to decrease the amount of same results being sent, but found it to occasionally make the Java application lag slightly so decided to leave them out and allow it to continue to send multiples. You can also see the data was successfully written to the text file twice.

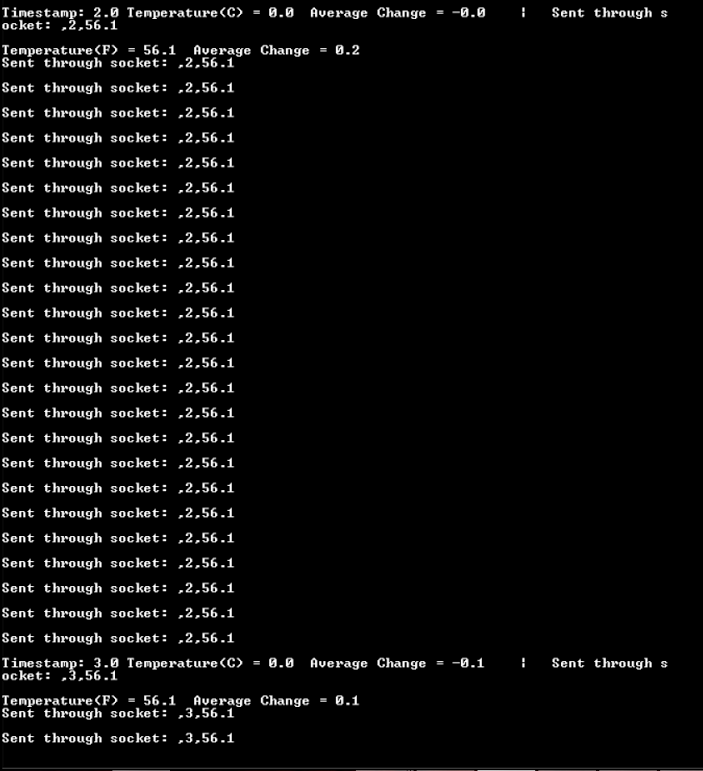
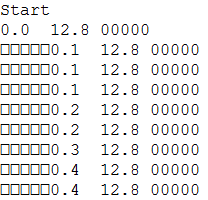


Figure 9 - Data sent through Socket

You may wonder why the information being sent to the Java application is separated by commas. This is because there was a formatting issue between the C++ char array that is sent through the socket and the Java String that is received from it. Initially when I tried to print the information received from the socket in Java I obtained strange box character like in the below picture. I attempted to convert these two ASCII by shifting each element and adding 30H, and found each of these boxes were ‘0’s. I am still not quite sure why these boxes were showing up, but if I separated each number with a comma and used the Java split function I was able to parse out the correct numbers and display them on the Java application. Below is a picture of the strange characters I received, followed by the data without the commas, and finally the converted ASCII characters of the mysterious characters.



Mysterious “Box” Characters

Converted “Boxes” to ASCII

Figure 10 - Mysterious "Box" Characters

Finally, the last two pictures will show the final information displayed on the Java application and the final values stored on the USB drive. Overall, the program worked perfectly, though sometimes the Java application can lag behind for a few milliseconds. Also, I should note I was not able to get the program to work when the USB Flash Drive was **not** plugged in.

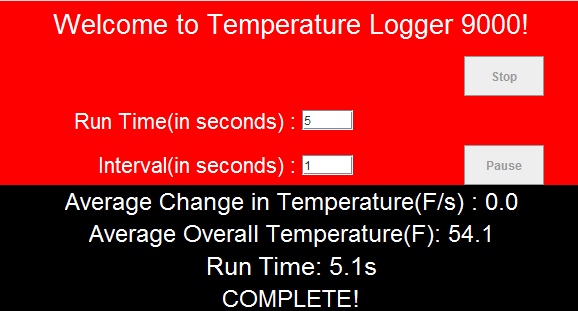


Figure 11 - Completion Screen of Application

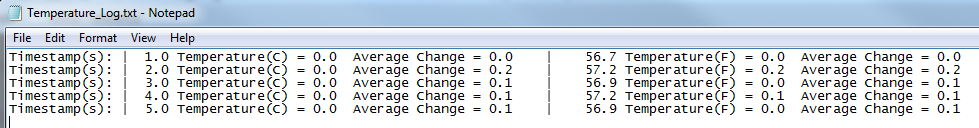


Figure 12 - Data Written to Text File

Throughout the project I did run into problems with running out of memory, and the board freezing up. I found this to be due to the Mailbox allocating space for the mail but never actually freeing it, resulting in an insufficient amount of memory error. To fix this, I simply traced when I wrote to memory, and when I released, and found that I was getting caught in a loop where I was never able to get to the free statement to release the mail.

In addition, I ran into issues implementing the pause/resume button on the Java application. When pressing the pause button, I had to find a way to stop the real-time task and resume it at the correct time so that it would keep on track with it original time interval. Initially, I was able to pause and resume the timer, but since the real time task continued to run, it was out of sync with the rest of the program. Thus, I realized I needed to delete the real time task for each pause and create a new one when the resume button was pressed. However, when creating the real-time task, it did not execute at the time of creation (i.e. it would wait for the next interval and thus skip one reading). To fix this, I had to separately read the temperature when creating the new task so that it would not skip a measurement.

## Conclusions

Although, the system may rarely lag for a few milliseconds, the successful tests above prove that the project was functional and successful in obtaining its goal of allowing a user to monitor the temperature changes and the average temperature of a room. At first, I was worried that the microcontroller would not have enough speed to process all of the elements of the circuit quickly enough, but after many successfully demonstrations I can conclude that the MBED was definitely up to the task. We can assume successfully that the real-time based task and multithreading involved in the software implementation performed as desired. As the correct data was displayed on all peripherals (LCD, Java application, and USB Text File), we can confirm that all the threads worked without issues and the correct data was recorded. Also, since the LED lights and speaker correctly turned on when the USB Text File was written to, we can assume that the write was successful each time (which we can also confirm by looking at the data on the flash drive). I can also conclude that we successfully were able to communicate between the C++ and Java programs via TCP which I found to be quite useful information that could be implemented in a number of different projects. Essentially, since each component operated as were intended, I conclude that the project proved to be a successful temperature data logger.

For future optimization of the project, I would likely explore having multiple boards running at once, so that it would allow a user to measure many different locations at once. I would also hope that MBED may come out with a fast microcontroller so that any additional elements I wish to add would not slow down the process of collecting temperature data. Furthermore, I would attempt to make the board completely wireless by replacing TCP with some other method that would not require an Ethernet cable and powering the MBED with a battery instead of the USB cable. Potentially, making the circuit completely wireless would be optimal. Lastly, adding additional climate sensors such as a humidity sensor could prove to be useful in any climate control applications.

## Appendices

### C++ Program

/\*

Author: Mike Brauch

Date: 5/11/2014

Description: This project was built around being able to read data from a temperature sensor and to display this data to the user in a

useful manner relating to time. Using an HD44780 20x4 White Text on Blue Background LCD, you were able to start this application via a Ethernet TCP socket

controlled by a Java application on a computer. Via this application, you are able to specify the run time, the interval in which it should take readings,

pausing/resuming the application, and stopping it. On the event of a stop button it will take what data it has gathered and calculate the final information.

Upon reaching the max run time, it will also calculate this information.

Final Info Includes:

Average Change in Temp in both Celsius and Farenheit

Average Overall Temperature in both Celsius and Farenheit

Total Run Time

MBED Libraries Used: EthernetInterface, FatFileSystem, RTOS, MSCFileSystem, TextLCD

Concepts Used from Class: Sockets, Mailbox, Pthreads, Real-time tasks

\*/

#include "mbed.h"

#include "MSCFileSystem.h"

#include "EthernetInterface.h"

#include "TextLCD.h"

#include "rtos.h"

#define SERVER\_PORT 20

#define FSNAME "msc"

// Temperature Structure

typedef struct {

float tempC;

float tempF;

float timeStamp;

float avgC; // average change in celsius per second

float avgF; //average change in farenheit per second

} Temperature;

/\*\*\*\*\*\*\* Defines all Analog/Digital devices connected to microcontroller\*\*\*\*\*\*\*/

MSCFileSystem myUSBPort(FSNAME);

DigitalOut led1(LED1);

DigitalOut led2(LED2);

DigitalOut led3(LED3);

DigitalOut led4(LED4);

TextLCD lcd(p10, p12, p15, p16, p29, p30, TextLCD::LCD20x4); // rs, e, d0-d3

AnalogIn tempSensor(p17); //Temeperature sensor

PwmOut Speaker(p21);

Mail<Temperature, 16> mail\_box; // allocatable queue - very similar to FIFO

Timer t; //timer

FILE \*fp; //temperature logging File

bool cont = true, pause = false; //boolean for stopping and pausing application

int numOfReadings = 0; //total number of temperatures read

float totalC = 0, totalF = 0; //total temp read

int TimeToRun = 0, timeInterval = 0; //how long the application runs given by JAVA application

TCPSocketConnection client; //client connecting to JAVA app

int pauseFlag = 5;

/\*\*\*\*\*\*\*\*\*\* This function takes the pointer to mail and shared buffer to calculate and store temperature from sensor \*\*\*\*\*\*\*/

void calcTemp(Temperature \*mail, void const \*args){

Temperature\* myTemp = (Temperature\*)args;

float holdC, holdF;

//conversion to degrees C

holdC = myTemp->tempC;

holdF = myTemp->tempF;

mail->tempC = ((tempSensor\*3.3)-0.600)\*100.0;

myTemp->tempC = ((tempSensor\*3.3)-0.600)\*100.0;

//convert to degrees F

mail->tempF = (9.0\*mail->tempC)/5.0 + 32.0;

myTemp->tempF = (9.0\*mail->tempC)/5.0 + 32.0;

//storing time

mail->timeStamp = t.read();

myTemp->timeStamp = t.read();

//computing average change in temperature

printf("%.1f + %.1f - %.1f = ", myTemp->avgF, myTemp->tempF, holdF);

mail->avgC = mail->avgC + (mail->tempC - holdC);

myTemp->avgC = myTemp->avgC + (myTemp->tempC - holdC);

mail->avgF = mail->avgF + (mail->tempF - holdF);

myTemp->avgF = myTemp->avgF + (myTemp->tempF - holdF);

printf("%.1f\n", myTemp->avgF);

numOfReadings++; //increase total number of readings

totalC += myTemp->tempC; //increase total temperature read in C

totalF += myTemp->tempF; //increase total temperature read in F

}

void TextLCD(void const \*args){

Temperature\* myTemp = (Temperature\*)args;

char buffer[20];

while(cont) { // loop to update LCD interface

lcd.cls();

lcd.locate(0,0);

lcd.printf("Temperature Logger");

lcd.locate(0,1);

lcd.printf(" Time = %.1f", t.read());

lcd.locate(0,2);

lcd.printf("C = %.1f F = %.1f", myTemp->tempC, myTemp->tempF);

lcd.locate(0,3);

lcd.printf("C/s= %.1f F/s= %.1f", (myTemp->avgC/t.read()), (myTemp->avgF/t.read()));

Thread::wait(50); // wait 0.5ns to reduce the number of double sent times

if (t.read() >= TimeToRun) //break loop if time to end

cont = false;

}

snprintf(buffer, sizeof(buffer), ",#,%.1f,%.1f,%.1f, \n", (float)totalF/numOfReadings, (myTemp->avgF/t.read()), t.read()); //puts final data to application in buffer

client.send\_all(buffer, sizeof(buffer));//sends final data to application

lcd.cls();

/\*\*\*\*\* Display all final data on LCD\*\*\*\*\*/

lcd.locate(0,0);

lcd.printf("COMPLETE");

lcd.locate(0,1);

lcd.printf("AvgC= %.1f", (float)totalC/numOfReadings);

lcd.locate(0,2);

lcd.printf("AvgF= %.1f", (float)totalF/numOfReadings);

lcd.locate(0,3);

lcd.printf("C/s= %.1f F/s= %.1f", (myTemp->avgC/t.read()), (myTemp->avgF/t.read()));

/\*\*\*\*\* Play final noise on speaker\*\*\*\*\*\*/

Speaker.period(1.0/500.0); // 500hz period

Speaker =0.01; //50% duty cycle - max volume

Thread::wait(1500); // wait 1.5s

Speaker=0.0; // turn off audio

led1 = 1;

led2 = 1;

led3 = 1;

led4 = 1;

cont = false;

}

//write temp from analalog port 17 to shared buffer args

//This is a real-time task and will only execute every given interval from application

void readTemp(void const \*args){

if (pause == true){

while(1){

if ((int)t.read()%timeInterval == 0 && pause == false){

break;

}

}

}

//turn on LED's when reading temp

led1 = 1;

led2 = 1;

led3 = 1;

led4 = 1;

Temperature \*mail = mail\_box.alloc(); //allocate mailbox to memory

calcTemp(mail, args);

mail\_box.put(mail); //put it on the mailbox

mail\_box.free(mail);

}

// Speaker/LED thread - will play noise and flash LED everytime temp is read

void speaker(void const \*args)

{

while(cont) {

if(led1 == 1){ //use LED as a flag

Speaker.period(1.0/1000.0); // 500hz period

Speaker =0.05; //50% duty cycle - max volume

Thread::wait(1); // short wait

Speaker=0.0; // turn off audio

Thread::wait(500); // wait 0.5s

led1 = 0;

led2 = 0;

led3 = 0;

led4 = 0;

}

}

}

void Pause(void const \*args) //will await message from application for pausing/resume

{

char buffer[5];

while(cont) {

client.receive(buffer, sizeof(buffer)); //will wait until info recieved

printf("%s\n", buffer);

if (strcmp(buffer, "Pause") == 0){ //pauses all except for temp reader (stops timer)

t.stop();

pause = true;

}

else if (strcmp(buffer, "Resum") == 0){ //resumes all threads

t.start();

pause = false;

}

else if (strcmp(buffer, "Stopn") == 0){ // stops all threads

cont = false;

break;

}

memset(&buffer, 0, sizeof(buffer)); //clear buffer

}

}

/\*\*\*\*\*\* Thread responsible for writing to the USB Flash Drive\*\*\*\*\*\*\*\*\*\*/

void USBWrite(void const \*args)

{

FILE \*fp = fopen( "/" FSNAME "/Temperature\_Log.txt", "w"); //open file on flash drive

if ( fp == NULL ) //if error

{

error("Could not open file for write\n");

}

else

printf("File opened successfuly\n");

while(cont){

if (pause == false){

osEvent evt = mail\_box.get(); // this will wait until something is put onto the mailbox

if (evt.status == osEventMail && pause == false && cont == true) { //if there is mail and application is not paused or stopped

Temperature \*mail = (Temperature\*)evt.value.p;

fprintf(fp, "Timestamp(s): | %.1f Temperature(C) = %.1f Average Change = %.1f | ", mail->timeStamp, mail->tempC, (mail->avgC/t.read()));

printf("Timestamp: %.1f Temperature(C) = %.1f Average Change = %.1f | ", mail->timeStamp, mail->tempC, (mail->avgC/t.read()));

fprintf(fp, "%.1f Temperature(F) = %.1f Average Change = %.1f\r\n", mail->tempF, (mail->avgF/t.read()));

printf("Temperature(F) = %.1f Average Change = %.1f\n", mail->tempF, (mail->avgF/t.read()));

mail\_box.free(mail);

}

}

}

fclose(fp); //closes file

}

int main() {

Temperature myTemp;

myTemp.avgF = 0;

myTemp.avgC = 0;

myTemp.tempC = ((tempSensor\*3.3)-0.600)\*100.0;

myTemp.tempF = (9.0\*myTemp.tempC)/5.0 + 32.0;

led1 = 0;

led2 = 0;

led3 = 0;

led4 = 0;

DIR \*d;

struct dirent \*p;

char\* runtime;

char\* interval;

int n = 0;

char buffer[15];

d = opendir("/" FSNAME);

/\*\*\* Display all files on the flash drive \*\*\*/

printf("\nList of files on the flash drive:\n");

if ( d != NULL )

{

while ( (p = readdir(d)) != NULL )

{

printf(" - %s\n", p->d\_name);

}

}

else

{

error("Could not open directory!");

}

EthernetInterface eth;

eth.init(); //Use DHCP

eth.connect();

printf("\nIP Address is %s\n", eth.getIPAddress()); //print IP of MBED

lcd.locate(0,0);

lcd.printf("Wait for connection...");

TCPSocketServer server;

server.bind(SERVER\_PORT);

server.listen();

printf("\nWait for new connection...\n");

server.accept(client); //wait and accept new client

client.set\_blocking(true, 1500); // Set to blocking

printf("Connection from: %s\n", client.get\_address()); //print JAVA IP on COM port

/\*\*\*\*\* Print Ethernet Info on LCD Interface \*\*\*\*/

lcd.cls();

lcd.locate(0,0);

lcd.printf("My IP: %s", eth.getIPAddress());

lcd.locate(0,1);

lcd.printf("Wait for info from");

lcd.locate(0,2);

lcd.printf("%s", client.get\_address());

n = client.receive(buffer, sizeof(buffer)); //will wait until info recieved frin JAVA application for runtime and interval

if (n == 0) //if error receiving

printf("Error recieving data");

runtime = strtok(buffer, " "); //tokenize buffer

interval = strtok(NULL, " ");

TimeToRun = atoi(runtime);

timeInterval = atoi(interval);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\* Start all threads \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

printf("Runtime %d Interval %d\n", TimeToRun, timeInterval);

Thread writeToLog(USBWrite);

Thread speakerThread(speaker);

Thread pauseThread(Pause);

lcd.cls();

Thread LCD\_thread(TextLCD, &myTemp);

RtosTimer tempReader(readTemp, osTimerPeriodic, &myTemp);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

t.start(); //start timer

tempReader.start(timeInterval\*1000); //setup real time task with period of 5 seconds

memset(&buffer, 0, sizeof(buffer)); //Preclear buffer

while(cont){

if (pause == true){

tempReader.stop();

pauseFlag = 0;

}

if ((int)t.read()%timeInterval == 0 && pauseFlag == 0){

tempReader.start(timeInterval\*1000); //setup real time task with period of 5 seconds

led1 = 1;

led2 = 1;

led3 = 1;

led4 = 1;

Temperature \*mail = mail\_box.alloc(); //allocate mailbox to memory

calcTemp(mail, &myTemp);

mail\_box.put(mail); //put it on the mailbox

pauseFlag = 1;

}

snprintf(buffer, sizeof(buffer), ",%d,%.1f\n", (int)t.read(), myTemp.tempF); // put information in char\* buffer for TCP transmit

printf("Sent through socket: %s\n", buffer); //Verify on COM port

client.send\_all(buffer, sizeof(buffer)); //Send buffer over TCP

//Thread::wait(500); // wait 0.5ns to reduce the number of double sent times

if (n == 0) // If failed

printf("Error sending to client");

memset(&buffer, 0, sizeof(buffer)); //Clear buffer

}

tempReader.stop();

client.close(); //close tcp connection

Thread::wait(osWaitForever);

}

### Java Program

package JavaClient;

import java.awt.event.ActionEvent;

import java.awt.event.KeyEvent;

import java.awt.event.MouseEvent;

import javax.swing.JFrame;

import java.awt.\* ;

import java.awt.event.\* ;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import javax.swing.\* ;

public class JavaClient extends JFrame implements ActionListener, MouseListener {

/\*\*

\*

\*/

private static final long serialVersionUID = 1L;

JTextField runTime, timeInterval;

JPanel masterPane, p1, p2, Title;

JLabel rTime, tInterval, textTitle, tempTitle, tempDisplay, timeTitle, timeDisplay;

JButton start, end, pause;

private JFrame frame = new JFrame("Client");

//ButtonHandler handler;

static java.net.Socket socket;

static PrintWriter out;

static BufferedReader in;

int timeToRun, tempInterval;

boolean cont = true;

public static void main(String[] args) throws Exception {

// TODO Auto-generated method stub

JavaClient client = new JavaClient();

client.setSize(600,360); // set to 375x300 window

client.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

client.setVisible(true);

client.connectToServer();

client.recieveData();

}

void connectToServer() throws Exception {

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

boolean connected = false;

while(connected == false){

String serverIP = JOptionPane.showInputDialog(

frame,

"Enter IP Address of the Server:",

JOptionPane.QUESTION\_MESSAGE);

if (serverIP == null)

System.exit(0);

String port = JOptionPane.showInputDialog(

frame,

"Enter Port Number",

"Welcome to the Temperature Logger!",

JOptionPane.QUESTION\_MESSAGE);

if (port == null)

System.exit(0);

try{

socket = new java.net.Socket(serverIP, Integer.parseInt(port)); //create socket to C++ program

connected = true;

} catch (java.net.ConnectException e){

JOptionPane.showMessageDialog(null, "Could not establish connection, try again!");

} catch (NumberFormatException e){

JOptionPane.showMessageDialog(null, "Could not establish connection, try again!");

}catch (java.net.SocketException e){

JOptionPane.showMessageDialog(null, "Could not establish connection, try again!");

}

}

out = new PrintWriter(socket.getOutputStream(), true); // output to socket

in = new BufferedReader(new InputStreamReader(socket.getInputStream())); //input from socket

}

void recieveData() throws Exception {

String[] tokens = {" ", " "};

char[] ASCII = {'0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'}; //represent HEX 0-F

while (cont) {

//tokens = (in.readLine().getBytes());

try{

tokens = in.readLine().split(","); //split according to comma

if (tokens[1].charAt(0) == '#'){ //if starts with # then final data arrived – display all data

System.out.println("Average Change in temperature(F/s): " + tokens[2]);

System.out.println("Average overall temperature(F): " + tokens[3]);

System.out.println("Run Time: " + tokens[4]);

tempTitle.setText("Average Change in Temperature(F/s) : " + tokens[3]);

tempDisplay.setText("Average Overall Temperature(F): " + tokens[2]);

timeTitle.setText("Run Time: " + tokens[4] + "s");

timeDisplay.setText("COMPLETE!");

start.setEnabled(false);

pause.setEnabled(false);

break;

}

else

System.out.println(tokens[1] + " " + tokens[2]); // take in temperature and timestamp

/\* To find out what the strange "box" characters were

System.out.print(" " + ASCII[(readIn.charAt(0) >> 4)]);

System.out.print(ASCII[(readIn.charAt(1) >> 4)]);

System.out.print(ASCII[(readIn.charAt(2) >> 4)]);

System.out.print(ASCII[(readIn.charAt(3) >> 4)]);

System.out.println(ASCII[(readIn.charAt(4) >> 4)]);

All five of them were '0'

\*/

} catch (java.net.SocketException e){

JOptionPane.showMessageDialog(null, "Connection Closed");

}

timeDisplay.setText(tokens[1]);

tempDisplay.setText(tokens[2]);

}

socket.close();

}

/\*\*\*\*\*\*\* Formats Graphical Interface \*\*\*\*\*\*\*\*\*\*\*\*/

public JavaClient(){

super("Temperature Logger");

masterPane = new JPanel();

masterPane.setLayout(new BoxLayout(masterPane, BoxLayout.Y\_AXIS));

masterPane.setBackground(Color.black);

start = new JButton("Start");

pause = new JButton("Pause");

pause.setEnabled(false);

start.addMouseListener(this);

pause.addMouseListener(this);

start.setPreferredSize(new Dimension(80, 40));

pause.setPreferredSize(new Dimension(80, 40));

Title = new JPanel();

Title.setBackground(Color.red);

p1 = new JPanel();

p1.setBackground(Color.red);

textTitle = new JLabel("Welcome to Temperature Logger 9000!");

textTitle.setFont(new Font(textTitle.getName(), Font.PLAIN, 27));

textTitle.setForeground(Color.white);

Title.add(textTitle, BorderLayout.SOUTH);

GridBagLayout gb = new GridBagLayout();

GridBagConstraints gc = new GridBagConstraints();

p1.setLayout(gb);

rTime = new JLabel("Run Time(in seconds) : ");

tInterval = new JLabel("Interval(in seconds) : ");

rTime.setFont(new Font(textTitle.getName(), Font.PLAIN, 22));

tInterval.setFont(new Font(textTitle.getName(), Font.PLAIN, 22));

rTime.setForeground(Color.white);

tInterval.setForeground(Color.white);

runTime = new JTextField("");

timeInterval = new JTextField("");

gc.insets=new Insets(10,0,0,0);

gc.ipadx=0;

gc.weightx = 0.25;

gc.gridx=0;

gc.gridy=1;

gc.anchor=GridBagConstraints.EAST;

p1.add(rTime,gc);

gc.insets=new Insets(10,0,0,0);

gc.ipadx=40;

gc.weightx = 0.25;

gc.gridx=1;

gc.gridy=1;

gc.anchor=GridBagConstraints.WEST;

p1.add(runTime,gc);

gc.insets=new Insets(10,0,0,0);

gc.ipadx=0;

gc.weightx = 0.25;

gc.gridx=0;

gc.gridy=2;

gc.anchor=GridBagConstraints.EAST;

p1.add(tInterval,gc);

gc.insets=new Insets(10,0,0,0);

gc.ipadx=40;

gc.weightx = 0.25;

gc.gridx=1;

gc.gridy=2;

gc.anchor=GridBagConstraints.WEST;

p1.add(timeInterval,gc);

gc.insets = new Insets(10,0,0,0);

gc.ipadx = 0;

gc.weightx = 0.25;

gc.gridx = 2;

gc.gridy = 0;

gc.anchor=GridBagConstraints.CENTER;

p1.add(start,gc);

gc.insets = new Insets(10,0,0,0);

gc.ipadx = 0;

gc.weightx = 0.25;

gc.gridx = 2;

gc.gridy = 2;

gc.anchor=GridBagConstraints.NORTH;

p1.add(pause,gc);

tempTitle = new JLabel("Temperature Reading (F)");

tempDisplay = new JLabel("0.0");

timeTitle = new JLabel("Time (Seconds)");

timeDisplay = new JLabel("0.0");

p2 = new JPanel();

p2.setBackground(Color.black);

p2.setLayout(new BoxLayout(p2, BoxLayout.Y\_AXIS));

tempTitle.setFont(new Font(tempTitle.getName(), Font.PLAIN, 24));

tempDisplay.setFont(new Font(tempDisplay.getName(), Font.PLAIN, 24));

timeTitle.setFont(new Font(timeTitle.getName(), Font.PLAIN, 25));

timeDisplay.setFont(new Font(timeDisplay.getName(), Font.PLAIN, 24));

tempTitle.setForeground(Color.white);

tempDisplay.setForeground(Color.white);

timeTitle.setForeground(Color.white);

timeDisplay.setForeground(Color.white);

tempTitle.setAlignmentX(Component.CENTER\_ALIGNMENT);

tempDisplay.setAlignmentX(Component.CENTER\_ALIGNMENT);

timeTitle.setAlignmentX(Component.CENTER\_ALIGNMENT);

timeDisplay.setAlignmentX(Component.CENTER\_ALIGNMENT);

p2.add(tempTitle);

p2.add(tempDisplay);

p2.add(timeTitle);

p2.add(timeDisplay);

masterPane.add(Title);

masterPane.add(p1);

masterPane.add(p2);

add(masterPane, BorderLayout.NORTH);

}

@Override

public void actionPerformed(ActionEvent e) {

// TODO Auto-generated method stub

}

@Override

public void mouseClicked(MouseEvent arg0) {

// TODO Auto-generated method stub

}

@Override

public void mouseEntered(MouseEvent arg0) {

// TODO Auto-generated method stub

}

@Override

public void mouseExited(MouseEvent arg0) {

// TODO Auto-generated method stub

}

@Override

public void mousePressed(MouseEvent e) {

// TODO Auto-generated method stub

System.out.println("Button pressed");

if (e.getSource().equals(start)){

System.out.println("Start");

if (start.getText().equals("Start")){

start.setText("Stop");

pause.setEnabled(true);

timeToRun = Integer.parseInt(runTime.getText());

tempInterval = Integer.parseInt(timeInterval.getText());

out.println(timeToRun + " " + tempInterval);

}

else

{

out.println("Stopn");

pause.setEnabled(false);

start.setEnabled(false);

cont = false;

}

}

if (e.getSource().equals(pause)){

if (pause.getText().equals("Pause")){

pause.setText("Resume");

out.println("Pause");

}

else if (pause.getText().equals("Resume")){

pause.setText("Pause");

out.println("Resum");

}

}

}

@Override

public void mouseReleased(MouseEvent arg0) {

// TODO Auto-generated method stub

}

}