|  |  |
| --- | --- |
| \\cetus.ece.missouri.edu\users\fischerjd\Desktop\Current Semester\common\logo-clear.png | University of Missouri – Columbia  Department of Electrical and Computer Engineering  *“A Proud Tradition, Engineering the Future!”* |

Thermostat and Home Security System

ECE 4220 Embedded systems   
Final Project Report

April 1st, 2015

— Engineering Team —

Alex Spiva, Electrical and Computer Engineering

— Course Instructor —

Luis Alberto

Department of Electrical and Computer Engineering

University of Missouri – Columbia

Table of Contents

[Abstract 3](#_Toc419487515)

[Introduction 3](#_Toc419487516)

[Background 4](#_Toc419487517)

[Implementation 4](#_Toc419487518)

[Android Device 4](#_Toc419487519)

[Python Script 5](#_Toc419487520)

[C Program 5](#_Toc419487521)

[Experiments and Results 6](#_Toc419487522)

[Conclusion 7](#_Toc419487523)

[Appendix 7](#_Toc419487524)

[Android Device 7](#_Toc419487525)

[Python 12](#_Toc419487526)

[C Program 14](#_Toc419487527)

# Abstract

The project will entail creating a thermostat and home security system that is accessible from a mobile device. The thermostat will sense ambient temperature in the house and outside light. Using the incoming data and inputted user temperature the thermostat will control window blinds and regulate temperature. The home security system, when activated, will monitor door motion and internal motion. If trigged the home security system will raise an internal alarm.

# Introduction

The project will be using Bluetooth to communicate between the mobile Android device and the Raspberry Pi 2 Model B. The Android device will have a simple home screen that will report the current temperature and weather the user would like to change the temperature or set the alarm. If the user chose to change the temperature the Raspberry Pi will check the current room temperature and the light from outside to determine if it should use (or stop using) the natural light from outside to heat (or cool) the room. The system will also activate the central air to raise or lower the temperature of incoming air. If the user decides to set the alarm then the Raspberry Pi will monitor magnetic switches at doors and a motion sensor. If any device is triggered a speaker system hooked up to the Raspberry Pi will sound an alarm. The alarm can be turned off by disabling the alarm from the Android device.

The motivation to the project came from our house here in Columbia being broken into over winter break. I was curious if creating your own security system would be feasible. Although it is not connected to direct line networks to contact authority, the project served as a proof of concept. Also, the motivation for the window blinds regulator came from my internship over the summer. There were complaints about the constant necessity to open and close blinds along with the high price of cooling/heating a large workspace.

## Background

The thermostat is a common household device that assists in the temperature regulation of the home. The standard thermostat has two basic functions: monitor ambient temperature and signal for specific temperature air. They are typically hardware interfaced allowing the user to only input desired temperatures directly from the device.

Home security systems range in complexity in modern times. Advanced security systems incorporate video surveillance, motion detection, window breaking detection, integrated 911 calling systems, remote activation, and backup data-basing. Less advance systems typically only monitor doors, don’t have calling systems, and signal an alarm internally.

# Implementation

The project was broken down into three primary coding systems: the Android device, a python script on the Raspberry Pi, and a c program on the Raspberry Pi.

## Android Device

The Android device was programmed through the Android Studio which is an extension off of Eclipse. The Android Studio allowed the device interface to be essentially dragged and dropped into place. The meat of the Android device problem solving was within the implementation of the Bluetooth configuration. Maintaining connection was the next hardest portion. The program required redundancies that allowed for the system to disconnect and reconnect with ease. After debugging the connection errors it was a matter of sending the proper information to the Raspberry Pi. If the alarm was set or disabled then the simple message “alarmOn” or “alarmOff” was sent. If the temperature was set then the message “temp XX” was sent with the desired temperature in place of the ‘XX’.

## Python Script

The Bluetooth connectivity worked the best when running a python program. The program establishes the socket, broadcasts its name, and connects when paired with the Android device. While connected it receives the data from the Android device, interprets the data, then sends the information to the c program through a pipe. The python program then checks the incoming information from the c program, the current temperature of the room, and sends it to the Android Device.

## C Program

The c program initializes and sets up the GPIO pin modes. The temperature sensor, photocell, magnetic switch, and motion senor are set to inputs while the motor control and air flow system are set to outputs. The speakers are not part of the GPIO modes due to the system calls that are necessary to run the speakers. The c program then creates the multiple threads needed to continuously monitor all incoming data from the external devices. Another thread is created to monitor the incoming messages from the python pipe, decode them, and set flags based off the incoming information. The last thread is created to process the logic behind the set flags. Since all of the external device threads set or unset global variables, the logic thread checks the flags to see if it should raise or lower the blinds, sound the alarm, or turn off the alarm. This thread also sends back the current temperature of the room to the python program.

Figure 1 - Flow Diagram of the FInal Project

Temperature Sensor

Light Sensor

Magnetic Switch(es)

Motion Sensor

Raspberry Pi Processor

Window Blind Motor

Air Flow System

Speaker

Mobile Device

# Experiments and Results

The system wasn’t overwhelmingly experimented on. The biggest experimentalization came with the setup of the Raspberry Pi and the Android device. The Raspberry Pi itself is one big guess and check operation. The entire process of installing and uploading the operating system, combined with enabling Bluetooth, and installing packages for speaker support was quite the experiment. In the end the Bluetooth could connect and the speakers could play sounds but that is where the experimenting stops. Unfortunately the project never came completely together causing the “trial and error” process to unfold. The Android system was originally set up from a project that I developed a while ago. But there was a considerable amount of testing the redundancy of the Bluetooth connectivity. The system had to account for connections, disconnects, other pairings, and disabling of Bluetooth ability which caused an awful lot of situational testing.

# Conclusion

In my personal opinion the project is unfinished. There was a great deal of thought and logic that went into the conception of the individual pieces of the project. Each individual piece should serve as an insight to the direction of the project and the failure of connecting the pieces shouldn’t be the predominating thought. The Android application is robust and completely reusable for future project implementation. The python program is simplistic but implores the thought of sending direct information between python and c program through the stdin and stdout functions. The c program is a standard multithreaded program but takes the monitoring of individual components to a new level by incorporating a different mindset of logic. Overall the project was considerably intriguing and it was fun getting my hands dirty with the Raspberry Pi, linux, python, and android development. The connection between the three main components in the systems down fall but the individual systems should stand alone in concept and coding.

# Appendix

Here are the full codes, and comments, to each parts of the program.

## Android Device

package ams7c9.embeddedsystemsproject;

import android.app.Activity;

import android.bluetooth.BluetoothAdapter;

import android.bluetooth.BluetoothDevice;

import android.bluetooth.BluetoothSocket;

import android.os.Bundle;

import android.text.format.Time;

import android.view.Menu;

import android.view.MenuItem;

import android.widget.TextView;

import android.widget.Toast;

import java.io.IOException;

import java.io.InputStream;

import java.io.OutputStream;

import java.util.Set;

import java.util.UUID;

public class MainActivity extends Activity {

public BluetoothAdapter mBluetoothAdapter = null; //Instatiate the Bluetooth headers

private BluetoothDevice bluetoothDevice = null;

private BluetoothSocket bluetoothSocket = null;

private OutputStream outputStream; //output and input streams for bluetooth

private InputStream inputStream;

public String deviceName = "00:06:66:68:17:D8"; //My android device

public TextView curretTemp = (TextView) findViewById(R.id.textCurrentTemp); //Declare TextView Variables

public TextView alarmSet = (TextView) findViewById(R.id.textAlarmSet); //Declare TextView Variables

public int i = 0;

private String newMessage;

private String messageParts[];

@Override

protected void onCreate(Bundle savedInstanceState) { //On create see if device supports bluetooth

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();

if (mBluetoothAdapter == null) {

Toast.makeText(getApplicationContext(), "ERROR - Device does not support Bluetooth", Toast.LENGTH\_SHORT).show();

}

}

@Override

public void onStart() { //on start

super.onStart();

Thread connectionThread = new Thread(new Runnable() {

@Override

public void run() {

while (true) {

if (!mBluetoothAdapter.isEnabled()) { //while the bluetooth is enabled on the phone

//Check to see is the Bluetooth is enabled

runOnUiThread(new Runnable() {

@Override

public void run() {

Toast.makeText(getApplicationContext(), "ERROR - Please enable your Bluetooth", Toast.LENGTH\_SHORT).show();

//if it isn't enabled

}

});

try {

Thread.sleep(500);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

else {

break;

}

}

while (true) {

Set<BluetoothDevice> availableDevices = mBluetoothAdapter.getBondedDevices(); //check all avaible devices

if (availableDevices.size() > 0) { //compile the avalible devices into a set

for (BluetoothDevice x : availableDevices) {

if (x.getAddress().equals(deviceName)) {

bluetoothDevice = x; //check if any of the devices are mine

break;

}

}

}

if (bluetoothDevice == null) { //check if we could connect

runOnUiThread(new Runnable() {

@Override

public void run() {

Toast.makeText(getApplicationContext(), "ERROR - Couldn't connect to the Bluetooth device", Toast.LENGTH\_SHORT).show();

}

});

try {

Thread.sleep(500);

} catch (InterruptedException e) {

e.printStackTrace();

}

} else {

break;

}

}

while(true) {

UUID uuid = UUID.fromString("94f39d29-7d6d-437d-973b-fba39e49d4ee"); //UUID of my raspberry pi

try {

bluetoothSocket = bluetoothDevice.createRfcommSocketToServiceRecord(uuid);

bluetoothSocket.connect();

outputStream = bluetoothSocket.getOutputStream();

inputStream = bluetoothSocket.getInputStream();

} catch (IOException e) {

e.printStackTrace();

}

try {

Thread.sleep(500);

} catch (InterruptedException e) {

e.printStackTrace();

}

if (bluetoothSocket.isConnected()) { //check if connected before continuing on

break;

}

}

while (true) {

byte inByte, buffer[] = new byte[1024]; //Give the buffer 1 kB of data //read in information

int currentTemp;

i = 0;

try {

while ((inByte = (byte) inputStream.read()) != '#') { //Check for a delimiter

buffer[i++] = inByte;

}

buffer[i] = '\0';

newMessage = new String(buffer); //Store the message

currentTemp.setText(newMessage.trim()); //set the current temp on the phone

if (alarmSet.toString().toUpperCase().equals("On")) //send the proper info for the alarm to the Raspberry Pi

outputStream.write(1);

else

outputStream.write(0);

}

catch (Exception e) {

e.printStackTrace();

}

//Check button press

//send data

}

}

});

}

@Override

public boolean onCreateOptionsMenu(Menu menu) {

// Inflate the menu; this adds items to the action bar if it is present.

getMenuInflater().inflate(R.menu.main, menu);

return true;

}

@Override

public boolean onOptionsItemSelected(MenuItem item) {

// Handle action bar item clicks here. The action bar will

// automatically handle clicks on the Home/Up button, so long

// as you specify a parent activity in AndroidManifest.xml.

int id = item.getItemId();

if (id == R.id.action\_settings) {

return true;

}

return super.onOptionsItemSelected(item);

}

## Python

import os

import glob

import time

import RPi.GPIO as GPIO

from bluetooth import \*

from subprocess import Popen, PIPE

os.system('modprobe w1-gpio')

os.system('modprobe w1-therm')

out\_Pipe = Popen("./threads", stdin=None, stdout=PIPE) #establish the pipe from python to c program

in\_Pipe = Popen("./threads", stdin=PIPE, stdout=None) #establish the pipe from c program to python

server\_sock=BluetoothSocket( RFCOMM ) #set up bluetooth

server\_sock.bind(("",PORT\_ANY))

server\_sock.listen(1)

port = server\_sock.getsockname()[1]

uuid = "94f39d29-7d6d-437d-973b-fba39e49d4ee" #establish UUID

advertise\_service( server\_sock, "RaspberryPi", #broadcast socket

service\_id = uuid,

service\_classes = [ uuid, SERIAL\_PORT\_CLASS ],

profiles = [ SERIAL\_PORT\_PROFILE ],

)

print "Waiting for connection on RFCOMM channel %d" % port

client\_sock, client\_info = server\_sock.accept() #find client

print "Accepted connection from ", client\_info

while True:

try:

in\_data = client\_sock.recv(1024) #receive incoming data

if len(in\_data) == 0: break #determine if it is acceptable data

print "received [%s]" % in\_data

if 'tempSet' in data: #determine which piece of data came in

out\_data = data

elif data == '1':

out\_data = 'alarmOn'

elif data == '0':

out\_data = 'alarmOff'

else:

out\_data = 'ERROR - Couldnt read data'

out\_Pipe.stdin.write(out\_data) #send information down the pipe

in\_Pipe.stdout.read(temp) #check the pipe from the c program

client\_sock.send(temp) #send the current temp to the android device

print "sending [%s]" % temp

except IOError:

pass

except KeyboardInterrupt:

client\_sock.close() #close the bluetooth connection

server\_sock.close()

break

## C Program

#include <wiringPi.h>

#include <stdio.h>

#include <stdlib.h>

#include <sys/stat.h>

#include <pthread.h>

#include <sys/time.h>

#include <fcntl.h>

#include <semaphore.h>

#include <sys/types.h>

int alarmOn, currentTemp, setTemp, lightOutside, motion, magSwitchOpen, shadesUp, playSpeaker;

void \*speakersThread(void \*args) { //play speaker if playSpeaker = 1

while(1) {

if (playSpeaker) {

system("omxplayer -o local example.mp3"); //through a system call

sleep(3); //sleep long enough for full speaker sound

}

}

}

void \*messageThread (void \*args) {

char buffer[100];

char temp[100];

int i, c;

while(1) {

i = 0;

while(EOF != (c = fgetc(stdin))) { //read in the pipe info

buffer[i++] = c;

}

buffer[i] = '\0'; //end in terminating character

if (strncmp(buffer, "tempSet", 7) == 0) { //determine the incoming message

temp = strtok(buffer, ' ');

temp = strtok(NULL, ' ');

setTemp = atoi(temp);

}

else if (strcmp(buffer, "alarmOn") == 0) { //set alarmm if enabled

alarmOn = 1;

}

else if (strcmp(buffer, "alarmOff") == 0) {

alarmOn = 0;

}

}

}

void \*logicThread (void \*args) { //thread that determines the logic of the program

while(1) {

if (alarmOn) { //if the alarm is set

if (motion | magSwitchOpen) { //if the motion senor or the magnetic switch goes off

playSpeaker = 1; //sound the alarm

}

}

if (!alarmOn) { //if alarm is disabled

playSpeaker = 0; //turn off the alarm

}

if (currentTemp < setTemp) { //if the current temp is less than the temp that was user set

if (lightOutside & !shadesUp) { //if it is light outside and the shades down

//open blinds //open the blinds

}

}

if (currentTemp > setTemp) { //if the current temp is more than the temp that was set by the user

if (lightOutside & shadesUp) { //if it is light outside and the shades are up

//lower blinds //lower the blinds

}

}

if (!lightOutside & shadesUp) { //if it is dark outside and the shades are up

//lower blinds //lower the blinds

}

}

}

void \*monitorTempThread (void \*args) {

double adc, millivolts, tempC;

while (1) {

//didn't get an adc

//lost implementation of current temperature readings

//if I did have a proper adc then here is what the code would look like:

/\*

analogData = readadc.readadc( sensor\_pin,

readadc.PINS.SPICLK,

readadc.PINS.SPIMOSI,

readadc.PINS.SPIMISO,

readadc.PINS.SPICS);

millivolts = analogData \* (5000/1024);

tempC = ((millivolts - 100) / 10) - 40;

currentTemp = "%.1f" % tempC;

\*/

}

}

void \*monitorLightThread (void \*args) {

int reading;

while (1) {

reading = 0;

pinMode(16, OUTPUT); //set the photocell to output and low

digitalWrite(16, LOW); //allows the capacitor to discharge

usleep(10000); //sleep to let the capacitor to discharge completely

pinMode(16, INPUT); //set it to input

while (digitalRead(16) == 0) //read the input until the read is now digital high

reading += 1; //more counts ~ longer time it takes to fill capacitor ~ higher resistance ~ more light

if (reading < 100) //number of counts that is determined to be the cutoff between light or no light

lightOutside = 1;

else

lightOutside = 0;

}

}

void \*monitorSwitchThread (void \*args) { //thread that checks the magnetic switch

if (digitalRead(20)) //if it is high / opened

magSwitchOpen = 1;

else //if it is low / closed

magSwitchOpen = 0;

}

void \*monitorMotionThread (void \*args) { //thread to monitor the motion sensor

if (digitalRead(21)) //if it is high / passed by

motion = 1;

else //if it is low / un passed by

motion = 0;

}

int main (void) {

wiringPiSetup(); //set up the GPIO

wiringPiSetupGpio();

pinMode(12, INPUT); //temp sensor

pinMode(16, INPUT); //light sensor

pinMode(20, INPUT); //magnetic switch

pinMode(21, INPUT); //motion sensor

pinMode(13, OUTPUT); //motor

pinMode(19, OUTPUT); //air flow system

pthread\_t message, logic, monitorTemp, monitorLight, monitorSwitch, monitorMotion, speaker;

pthread\_create(&message, NULL, messagethread, NULL); //create the threads necessary to check the external devices and process the logic to the global variables

pthread\_create(&logic, NULL, logicThread, NULL);

pthread\_create(&monitorTemp, NULL, monitorTempThread, NULL);

pthread\_create(&monitorLight, NULL, monitorLightThread, NULL);

pthread\_create(&monitorSwitch, NULL, monitorSwitchThread, NULL);

pthread\_create(&monitorMotion, NULL, monitorMotionThread, NULL);

pthread\_create(&speaker, NULL, speakerThread, NULL);

while(1) {}

return 0;

}