CIS 542 Project Technical Documentation

Home Automation

Di Mu, Hangfei Lin

I. Objective

The objective of the project is to build a home automation system built based on Arduino microcontroller. Our system includes a home security system using motion and pressure sensors, a temperature analysis system and a remote LED display control system.

II. System Architecture

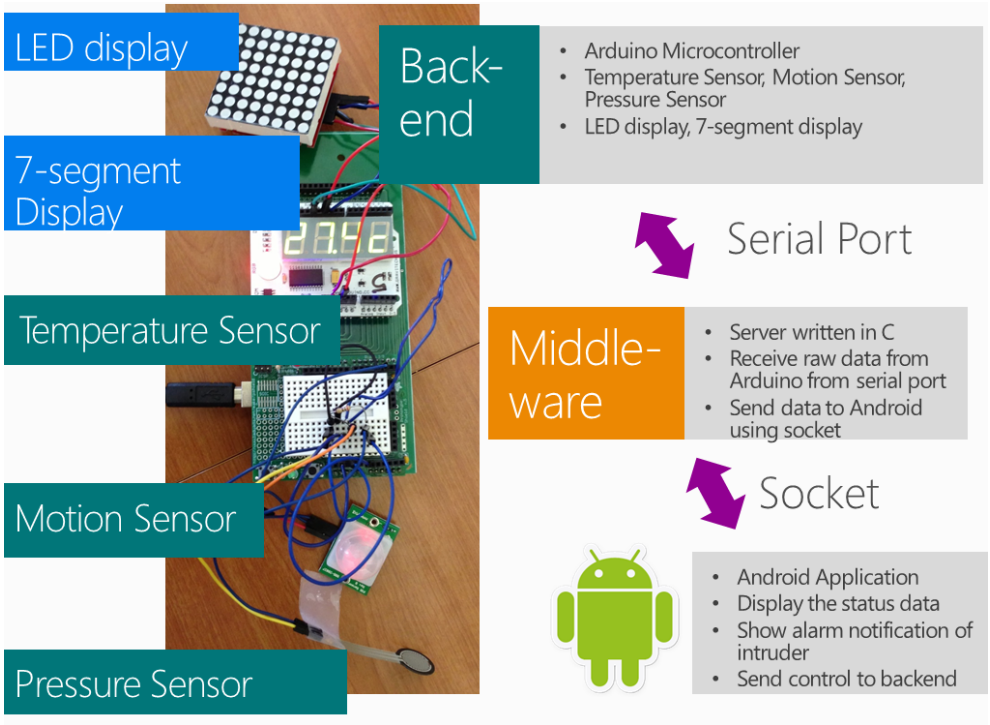

Figure 1. System Architecture

As shown in Figure 1, our system consists of three major components. The backend Arduino microcontroller, middleware C server and the front-end Android Application.

1. Back-end

Our backend consists of motion sensor, pressure sensor, temperature senor, 7-segment display and LED display. They are controlled by the Arduino microcontroller.

2. Middleware

Our middleware consists of a C code which generate three threads, one for reading the arduino data from the arduino, and the other is for setting up a server that sending data to the android app through socket and the last is for reading user control info from the Android App.

3. Front-end

Our front end is an Android app. It displays the user interface

III. Functionalities

1. When the pressure sensor detects pressure, a notification is sent to the Android device. The notification should appear in the user interface's "Status Bar". When the user receives the notification, he or she can then go into an app that will display the time at which it was received.
2. Same as #1, but for the motion sensor.
3. The user should be able to put the system into a "stand-by" mode in which it is not sending alerts if either sensor is tripped. When the system is in "stand-by" mode, the user should also be able to tell it to resume.
4. the Android device can send some sort of control signal to be displayed in the LED display
5. The user should be able to see the most recent temperature reading. If temperature readings are being reported by the sensor, the values should automatically update in the user interface.
6. The user should be able to see the average, low, and high temperature reading for the past hour (or some amount of time)
7. The user should be able to see a graphical representation (i.e., temperature readings displayed as a function of time) for the past hour.
8. The user should be able to set a threshold, such that, if the temperature goes above the threshold, the user will be notified even if she is not using the application. This should be done using an Android Notification.
9. the pressure and motion sensor can be used to change what's displayed in the LED display
10. there will be some sort of countdown timer shown on the 7-segment display

IV. Non-functional requirements

The data is updated every five seconds if allowed(connected and battery is good). If not, the chart will display blank data and the old data will pass to right. So the user can easily tell when the lasted data is sent.

The Arduino program only use 7KB of memory including everything.

The C program only use 10KB.

If the Android device has less than 20% of battery remaining, the reading is updated once every 15 seconds.

If the device has less than 10% of its battery remaining, the reading is updated once every minute.

*When displaying any data or message from the embedded components, the data should be no more than five seconds old. If it is more than five seconds old, a message should be shown to the user (you can assume that the clocks on all devices are synchronized).

* The Arduino program must use no more than 32KB of memory, including space needed for the code

* The C program (middleware) must use no more than 32KB of heap space
* If the Android device has less than 20% of its battery remaining and the user is viewing the most recent temperature reading, the reading should be updated once every 15 seconds.
* If the device has less than 10% of its battery remaining, the reading should be updated once every minute.

V. Communication Protocols

The Arduino gets readings from each component and sent the combined readings using BAUD9600 in the following format:

t1000 T25.00 P10 m0(time, temperature, pressure, motion).

The Arduino only reads a byte each loop, so the control signal would be sent in byte. Only three control signal allowed:

'1'-LED warning, '8'-LED smiling face, '0'-LED normal,
'7'-7-Segment Display countdown.

The C server only reads and sent data without processing the readings.
The C server only reads control singal from Console: type 'q' to quit the C server.
The Android App will reads the data, analyze the data and stored in the staticArdData class.
The Android App will sent control signal in byte(and it will be sent to the Arduino via C server). Other format is not allowed.

VI. Algorithm and data structure

The Arduino reads sensor readings from each components and sends data to the C server and recceive byte data from the C. The data is sent and recevied in 1 second interval.
The C server has 4 thread. Two reads from Arduino and sends data to the Arduino; the other reads data from Android and sends data to the Android.
The Android main activity will start a service(ReadArdService) reading data from the C, and store data in the staticArdData class staic data fields.(Considering we will need a very small amount of data storage, we only use static data fields instead of local disk to store the data.) The staticArdData will store current data and historical data in recent 10 hours(The data is stored in AChartEngine XYSeries). All other classes can access the data using getters.

VII. Error Handling and robustness

For C server, if there is any disconnection, it will close the program gracefully.
For Android, if C server shuts down unexpectedly or Arduino/Arduino component lost connection, it will still run and store blank data with respect to time. The Android App will not shutdown and will tell the user when is the latest data sent from the C server. The Android App will try to reconncet. Once the C server restarts, the Android will connect and displays the data.

VIII. Optimization

For power consumption, if the battery is bigger than 20%, the Android will read every 1 seconds. When the battery is lower than 20%, the Android will only reads the every 15 seconds. If the battery is lower than 10%, the Android will only reads the data every 1 minute.

For network bandwidth, the Android will reads a string form C server every 1 seconds when power allowed. And if there is no new data(we have a isNew flag), the Android would not sent data.

For memory, we only store approximately recent 100 readings: recent 100 seconds temperature, recent 100 seconds motion readings, recent 100 seconds pressure readings, recents 600 minutes temperature readings(each minute one reading).

For CPU utilization, basically the Android App need a every small amount of computation and need some resource to display UI and Chart. So what we do is to make UI and chart simpler and refresh the charts every 1 second. For C server, we used following techniques: avoid big if-else structures, some compiler optimizations.

IX. Analysis

For analysis tools:

Helgrind

-clear

Massif

-clear

Splint:

Our command: splint -warnposix -type -unrecog -retvalint -retvalother -compdef -nullret -usedef -paramuse -nullpass -exportlocal readArd2.c

Reasons to use these flags:

-We would pass NULL pointer in the pthread which would result in warnings. However we did not find better way to avoid this.

-We use the -warnposix: we dont want to go into posix library code

-We use the -unrecog: bzero is not recognized by splint

-retvalint, -retvalother: default sock lib

- -compdef: default socket connect lib problem

-nullret: we dont need return value and we dont need any argument passed into thread functions

gcov:

-coverage 80%, cover every part except for error handling code we couldn't test

-usedef: socket lib

-paramuse -nullpass : we dont need thread argument

-exportlocal: thread will share the public variable

output message:

gcov readArd2.c

File 'readArd2.c'
Lines executed:90.27% of 113
readArd2.c:creating 'readArd2.c.gcov'

gcov -b readArd2.c
File 'readArd2.c'
Lines executed:90.27% of 113
Branches executed:100.00% of 32
Taken at least once:78.13% of 32
Calls executed:82.35% of 51

readArd2.c:creating 'readArd2.c.gcov'

X. Other

Makefile:

2: C Server
3: gcov

Reference:

1. Project Website http://www.seas.upenn.edu/~cdmurphy/cis542/project/
2. Motion sensor http://learn.parallax.com/kickstart/555-28027
3. LED https://www.sparkfun.com/tutorials/201
4. Temperature Sensor http://www.gravitech.us/7segmentshield.html
5. Pressure Sensor http://bildr.org/2012/11/force-sensitive-resistor-arduino/
6. Graphical Representation Archartengine http://code.google.com/p/achartengine/