CS 3410 Systems Programming

Final Project Write Up

Group Name: NextGen Productions

Group Members: Lian Showl, Simon Patel, Timothy Lytkine

Background

For our final project, we built our own fitness monitoring device that is capable of running the basic analytics on information given to it. There are two main components of the system which are data collection and data processing. The sensors used were the heart rate sensor, real time clock, one display device with a matrix display, 2x7 segment LED, LCD screen, one environment sensor that can output temperature and humidity.

The Arduino sends a sensor reading to a host program (written in C) running on a UNIX-like system. The two programs communicate over the Arduino's serial connection. The host program stores a histogram-like structure of readings which can be easily queried and outlier readings can be found.

One of the the goals of this project was to detect when the current heart rate is an outlier. We accomplished this by using a histogram stored in memory on the host machine. We know that people's heart rates are different at different times of day. Clearly, someone's heart rate will be lower during the night when they are asleep than during their daily 8AM gym trip. Therefore, a separate histogram was is outputted to the screen for different parts of the day and different ranges of BPM in order to more accurately compare heart rate readings.

In part 3, the system was further enhanced by adding in knowledge about the user's surrounding environment. An environment sensor was integrated (a temperature sensor and humidity sensor) into the system in order to later on calculate a linear regression of its readings with the heart rate data. This component introduced a second process to the host program. In order to avoid communication errors, a child process and parent process was created using function loops that were controlled via forking. One process continues to run heart rate data collection and storage (as in Part 2) and the other runs a command line interface (as in Part 1). The goal is to allow data handling and processing to happen in a background process and not interfere with the user-facing command line interface.

In part 4, a database was added in order to store all the statistics that the heart rate monitor was storing over the progression of each part. After this final part was implemented, the system now is able to provide more advanced data analytics. All sensor readings are now stored in a persistent database *in addition to* the existing histogram structure. This separation allows data to be stored with differing degrees of precision in different mediums. Each storage medium has different trade-offs - querying the histogram in memory is faster than executing a SQL SELECT on a database, although the full database is more flexible in what it supports.

Specifications

We began using two sensors: the heart rate sensor and LCD display device. We have written an Arduino sketch that reads in the current heart rate and outputs it to the display device. Whenever the display device is showing real-time data, the green LED on the breadboard is on (and is off otherwise). Additionally, the Arduino flashes a red LED at the same rate as the detected heartbeat.

For the second part of this project, the information from the user's detected heartbeat is stored into a histogram. Imagine a two dimensional array: one axis is separated into 15 minute time increments (based on the time of day when the reading was taken) and the other axis contains "buckets" of possible heart rates. When a new reading is processed, we find where the proper time "bucket" intersects with the correct heart rate "bucket". Increment a counter at this position in the 2D array.

One additional consideration when creating this is that we want the data stored to be persistent - it should not be lost when the host program stops running. This requirement precludes the use of malloc/free like we have done in the past. Instead, we used a function called mmap to load the contents of a file into our program's virtual memory space. The advantages of this is that accessing virtual memory is much less computationally expensive than using physical memory. We used the open function to obtain a file descriptor for the backing file and passed the result to mmap to load it into memory. In addition, we had to use munmap and close on the mapping and file descriptor, respectively.

Finally, we integrated an *I*2*C* real-time clock into our system. We have used this device to provide precise timing information for our heart rate readings. When the Arduino starts up, it synchronizes timing information with the host program over the serial connection.

All sensor readings are in the database along with a timestamp signifying when the reading was obtained. In order to understand if there is a relationship between heart rate and temperature (the environment sensor values) we calculated a linear regression between our two variables (environment sensor reading and heart rate) to see what kind of relationship (if any) exists. We grouped the heart rate and temperature values together at the same time and inserted them into the database simultaneously in pairs.

The heart rate is the independent variable and the temperature value is the dependent variable when calculating the linear correlation.

The final specifications of this project needed the following functionality:

- show X: Set the output device to show X as the current heart rate instead of the current real-time value. In addition, print the value to the console.

- pause: Pause the output and keep the display device showing the current reading. Indicate this somehow in your circuit (using an LED for example).

- resume: Show the real-time heart rate on the display device. This should be the default mode of the system. Indicate this somehow in your circuit (using an LED for example).

- rate: Query the value of the heart rate sensor at the current time and print it to the console

- env: Query the value of the environment sensor from the Arduino and print it to the console.

- hist: Print a representation of the current time block's heart rate histogram to the console

- hist X: Print a representation of the given time block's heart rate histogram to the console

- date: Show the current value of the real-time clock on the console

- regression X: Calculate a linear regression between the environment and heart rate data for a given time block. Prints the regression and the appropriate coefficient of determination (*r*2). If X is not provided, default to the current time block.

- stat X: Print the following statistics for the given time block (for both heart rate and environment data): *reading count*, *mean*, *median*, *mode*, *standard deviation*. If X is not provided, default to the current time block.

- reset: Clear all data from the backing file

- exit: Exits the host program

Implementation

Part 1

The code we wrote in the Arduino sketch allows two LEDs to blink (one to the user’s live heartbeat and one to fades an LED to the user’s live heartbeat). After setting up the loop function with different pins to correspond to the circuit, the loop function is where the Arduino communicates with our C code.

With the boolean paused is set to false, the if statement continuously prins the user’s live BPM on the LCD screen. The way that this changes is by inputting different commands through the terminal (running the C file).

The loop in the sketch checks for serial input through Serial.available() and reads the user’s command string. These strings are commands that are sent after the user is prompted to enter showX, pause or resume. Once that is read in, the string from the terminal is compared to the hardcoded strings that are declared in the sketch. From there it compares them to each if-statement. If the resume command is read in, the boolean paused is set to false, and then clears the LCD to reset the user’s real time BPM data. If the command is show X, then the boolean paused is true and clears the LCD, and then resets the LCD to print “X”. Finally, if the command is set to pause, then the boolean paused is set to true and the LCD clears its screen and prints the last BPM read before the command was called.

Using the template from Lab 8, we started with the skeleton and built our code from there. We reset the dev path to our device’s port to ensure that the C file would be communicating with the correct port.
We have a while loop that continually runs until the user decides to quit the program. We then use strcmp to compare the strings that the user is typing in and compare them to the hardcoded preset char* variables in the C file which are then communicated to the Arduino using the send_cmd() function (provided in the template code). There is an if-statement for each command (resume, pause, exit and showX). We also added another command called ‘help’ that the user can use, and describes what each option does.

Part 2

The real time clock is configured using the I2C communication protocol . The time initially is set using the function setDS2321time. The parameters of the function are obtained upon compile time from the DATE and TIME predefined macros in C.

After this, the time is read from the real time clock using the readDS3231time function. The inputs to the function: second, minute, hour, dayOfWeek, month and year are all populated with values from the real time clock which starts counting from the time it is set to upon compile time. The time is read every time a command is sent from the C code (using the send_cmd function) to the Arduino and then the time is sent back to the C code in order to determine the index of the array that stores the histogram. The messages that are sent from the Arduino which include BPM, hour, minute and second are sent using Serial.write() byte by byte. There is a main loop that takes user input within the C code that allows the corresponding commands to be sent. Based upon the command requested, the values are then read into the buffer and then each byte stored in a corresponding index is set equal to a variable which is cast to a char or an int as necessary. ASCII conversion must be performed since the values sent are chars.

There are 96 possible time intervals in the array and 5 possible BPM values. Hence, there are 480 possible values being stored. The hour value that is sent from the Arduino is multiplied by 4 to determine the firstIndex. If the minute value is between 0 and 15 then the first index is equal to the hour value multiplied by 4. If the minute value is between 15 and 30 then one is added to this value, if the minute value is between 30 and 45 then two is added to this value and if the minute value is between 45 and 60 then three is added to this value. There are 24 hours in a day and 4 time intervals for each hour, 24 * 4 = 96 which is why the first index can range between 0 and 96. For the second index, the BPM value sent from the Arduino that is read from the heart rate sensor is used. If the BPM is between 0 to 40 then the second index is 0, if it is between 41 and 81 then the second index is 1, if it is between 81 and 120 then the second index is 2, if it is between 121 and 160 then the second index is 3 and if it is above 160 then the second index is 4. This comprises 5 possible values for the second index: 0, 1, 2, 3 and 4.

These 480 possible values are mmapped by the mapWrite, mapSync and mapRead functions. mapWrite writes the 480 values into memory by using a loop with the 2D array that stores the histogram and mapSync writes these values into the filepath. mapRead is the function that is called initially to get previous histogram values from memory. There is a function that prints the histogram on demand with the ranges of possible values based upon the time interval indicated in the file path that the histogram is stored. Outlier readings are being detected when the array that stores the histogram has its values incremented. If the current BPM which is received from the Arduino is between 0 and 40, a messages is flashed on the screen of the Arduino that says “Warning: BPM is low.” In addition, if the current BPM is greater than 160, a message is flashed on the screen of the Arduino that says “Warning: BPM is high.” These warnings are flashed on the LCD by the C code sending a low or high command based upon the BPM values which is then sent to the Arduino to flash the corresponding warning.

Part 3

The communication protocol of our system was extended by adding an environment sensor that outputs readings for temperature and humidity. The SI7021 environment sensor was added and configured using the I2C communication protocol. A difference between this sensor and the heart rate sensor and real time clock is that it uses 3.3V rather than 5 so this had the be adjusted accordingly. In order to accommodate multiple processes (and to assure the simultaneous functionality of multiple sensors) running at once, our program needed to be forked. A child process and parent process were created.

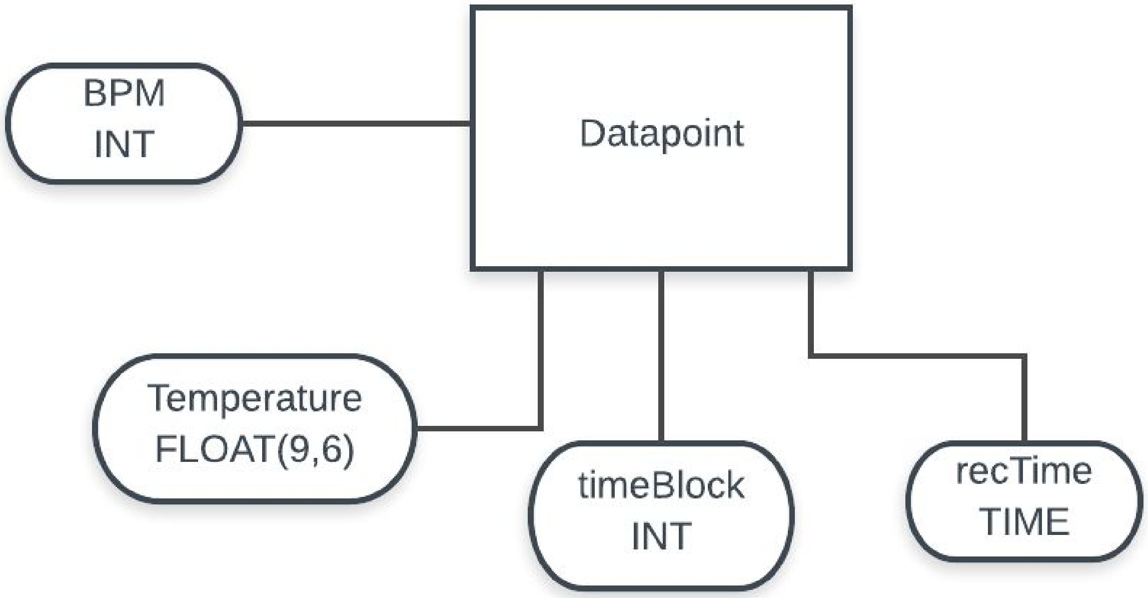
The parent process takes commands from the terminal. These commands can send data to the Arduino, print existing stored data to the console as well as receive information to the Arduino and call functions that use this received data. The list of expanded commands includes showX, pause, resume, rate, env, hist1, histX, reset and exit. The new and/or improved commands include rate, env, hist1, histX, reset and exit. The rate command is simply a modification of a functionality present in the previous implementation of the system. It sends a command for the Arduino to send the current heart rate which is then printed to the console.

The env command sends a command to the Arduino to query the temperature and humidity values from the environment sensor which are then printed to the console. The hist1 command simply prints the histogram of the current time block heart rate to the console. The histX command requests a time block number from the user in the console and prints a histogram for the heart rate values stored in virtual memory. The reset command clears all data from the backing file in virtual memory space in order to set the system up to read and store new values from the sensors. Finally, the exit command, kills the child and parent processes and exits the host program.

The child process simply requests the heart rate values from the Arduino every second. This process needed to be separate from the parent process so that the parent and child processes can wait on each other and there are no conflicts in sending and receiving data. Another distinction between the code for part3 and part2 is that two separate functions now exists. One for sending a command to the Arduino to request data from the heart rate sensor and a second one to request data from the environment sensor. The purpose of this is to allow the histogram for the heart rate values to be stored in the histogram as soon as this data is received and to differentiate the variable names for BPM, temperature, humidity and timing information. Further implementations however, would separate some of the corresponding code in the send command functions into separate functions and consolidate the send command functions into one.

Part 4

The first step to fulfilling the specifications of the final part of the system was adding a date command to show the current date on the Arduino. We used the existing code we wrote from earlier parts to fetch the data received from the Arduino. The Arduino requests the day, month and year from the real time clock and sends it back to the host computer. This data is then parsed and printed to the console in the format of a date. We needed to tweak a few things to resolve and ensure the parsing was correct. At first the date was being sent from the Arduino incorrectly. Using different debug statements, and once we found that the date was being sent correctly, we then printed the date to the LCD display. After adding the other new commands to our C code to send to the Arduino (regression X, stat X, others added with the date command), we had to outline a schema design for the database which can be seen below.

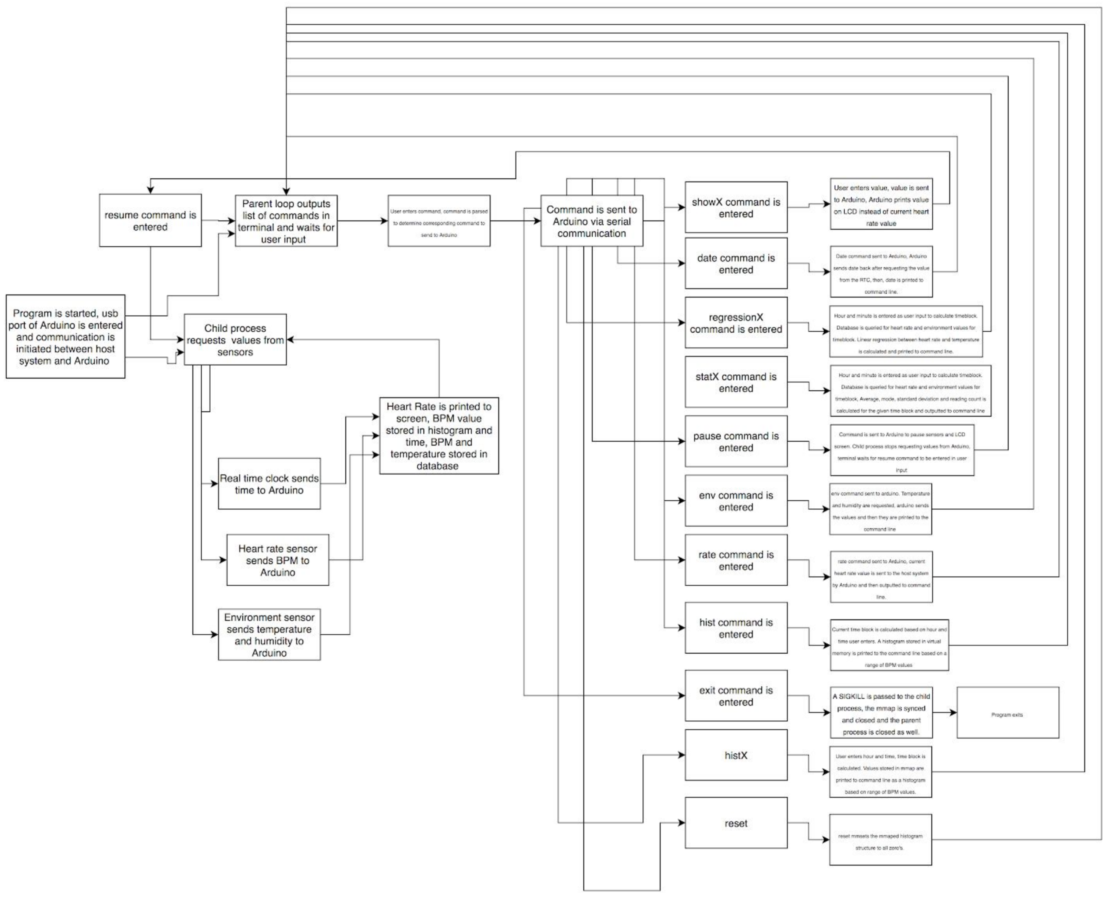


The database is composed of 4 attributes that stores the BPM as an int, the temperature as a float, the time block as an int and the recorded time as a time data type. Our function dbInit() opens the database to ensure successful connection and closes it if the connection was not successful. The data being inserted is the BPM, temperature at a specific time block and time every second.

Two additional functions were implemented called statX and regressionX. The function statX calculates the reading count, average, mode and standard deviation of BPM and Temperature. The other function regressionX calculates a linear regression between heart rate and temperature for a given time block. These functions query the data required to calculate these values from the database and provide useful statistics for the values monitored via our system.

To conclude, the state machine which represents the functionality of our final system can be seen below.

Text



resume command is entered
Parent loop outputs list of commands in terminal and waits for user input
User enters command, command is parsed to determine corresponding command to send to Arduino
Command is sent to Arduino via serial communication
Program is started, usb port of Arduino is entered and communication is initiated between host system and Arduino
Child process requests values from sensors
Heart Rate is printed to screen, BPM value stored in histogram and time, BPM and temperature stored in database
Real time clock sends time to Arduino
Heart rate sensor sends BPM to Arduino
Environment sensor sends temperature and humidity to Arduino
showX command is entered
User enters value, value is sent to Arduino, Arduino prints value on LCD instead of current heart rate value
date command is entered
Date command sent to Arduino, Arduino sends date back after requesting the value from the RTC, then, date is printed to command line.
regressionX command is entered
Hour and minute is entered as user input to calculate timeblock. Database is queried for heart rate and environment values for timeblock. Linear regression between heart rate and temperature is calculated and printed to command line.
statX command is entered
Hour and minute is entered as user input to calculate timeblock. Database is queried for heart rate and environment values for timeblock, Average, mode, standard deviation and reading count is calculated for the given time block and outputted to command line
pause command is entered
Command is sent to Arduino to pause sensors and LCD screen. Child process stops requesting values from Arduino, terminal waits for resume command to be entered in user input
env command is entered
env command sent to arduino. Temperature and humidity are requested, arduino sends the values and then they are printed to the command line
rate command is entered
rate command sent to Arduino, current heart rate value is sent to the host system by Arduino and then outputted to command line.
hist command is entered
Current time block is calculated based on hour and time user enters. A histogram stored in virtual memory is printed to the command line based on a range of BPM values
exit command is entered
A SIGKILL is passed to the child process, the mmap is synced and closed and the parent process is closed as well.
Program exits
histX
User enters hour and time, time block is calculated. Values stored in mmap are printed to command line as a histogram based on range of BPM values.
reset
reset mmsets the mmaped histogram structure to all zero's.