Final Project (Part 4)

CS 3410 Systems Programming

Due Date: Wednesday May 3, 2017 at 6:00 PM

You may work in teams of *up to* **three (3)** members from your lab section

Background Information

We now begin the final phase of the project. Each group should have *at least* one member currently taking CSCI 2441 ("Database Systems and Team Projects"). If this is not the case, please reach out to the TAs to make the proper arrangements. After this part has been implemented, the system will be able to provide more advanced data analytics. All sensor readings will be stored in a persistent database *in addition to* the existing histogram structure. This separation will let us store data with differing degrees of precision in different mediums. Each storage medium has different trade-offs - querying the histogram in memory will be faster than executing a SQL `SELECT` on a database, although the full database is more flexible in what it supports.

The first step will be to integrate a SQLite database into the system. SQLite is a relational database engine (it is SQL compatible) that is linked into your program. SQLite is often used in embedded systems (every Android phone, for example) due to simple API and light footprint. SQLite has C language bindings, so you can access the database from your code. An introduction can be found here and the full API specification is also available. You can use any database schema you like, so long as all other requirements are satisfied.

All sensor readings should be stored in the database along with a timestamp signifying when the reading was obtained. We want to understand if the value of the environment sensor has any relationship to the heart rate values. We will calculate a linear regression between our two variables (environment sensor reading and heart rate) to see what kind of relationship (if any) exists. You will have to build (x, y) pairs of the two readings. Do this by grouping a heart rate and environment reading taken from the same time (or within some small tolerance) together. Since we are testing the effects of the environment on heart rate, the environment sensor readings will be our dependent variable.

You may have to collect some data in various environments to introduce some variety in your environment sensor readings.

Specification (Part 3)

The command line interface will expose the following new functions. The full list is reproduced below with new items **bolded**.

- `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. Print 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.

Complete Function Listing

- `date`
- `env`
- `exit`
- `hist X`
- `pause`
- `rate`
- `regression X`
- `reset`
- `resume`
- `show X`
- `stat X`

Useful Links

- SQLite Quickstart
- Using SQLite in C Programs
- Introduction to Regression Analysis
- Various Formulas

Final Demo

The demo for the final part will be comprehensive and more involved than the previous parts. You should be able to demonstrate functionality from this and all prior parts. You will show off the system working and your implementation of all the commands above. Additionally, you will explain how you constructed your communication protocol and how that protocol ensures data integrity and transmission reliability. You may want to prepare visuals (slides, diagrams, etc) to assist in communicating your solution. You should also be prepared to explain the algorithmic process behind other aspects of the project (outlier detection, data integrity checks, command parsing, real-time clock synchronization, and sample creation for regression analysis). All of these ideas should be expressed in your write-up document, as well.

You must also show a "state machine" diagram for the components of your code (on the Arduino and host computer). This diagram should show the various control flow paths of your code and how the different parts interact with each other. For example, you can draw an arrow to show one process in your host code sending a signal to the other in order to ensure both processes abort when the user runs the `exit` command. For more information about drawing these diagrams, see the Wikipedia page for UML State Machine.

This demo is expected to take 5-7 minutes of time with the possibility of an additional 1-2 minutes for questions. Each group member should "lead" an equal share of the demo. For example, in a group of three, each person should talk for approximately 2 minutes for a total demo length of 6 minutes. In addition, each group member should be able to explain how any part of the code works and why certain architectural decisions were made.

The full project rubric can be found on the following page.

Deliverables and Grading

- Push your code to GitHub (a link will be posted on Blackboard) before the deadline. The following things should be in a directory named `part4`:
  - One Arduino sketch
  - The C source code (and appropriate header files) of your host program. Be sure to include a `Makefile` so the grader can build your code. Do not forget, your code *MUST* compile with the following `gcc` flags: `-std=c99 -Wall -pedantic -Werror`. You will lose credit if your code compiles, but only without those extra flags.
  - One Fritzing[1] file that shows the final circuit you built.
  - A `pdf` document containing:
    - The complete, final protocol specification you're using
    - An outline of your database schema
    - A state machine diagram for all components of your code (both Arduino and host)
    - Descriptions of how you are computing the following: outlier detection, time synchronization between the Arduino and host, building sensor reading pairs for regression analysis, and anything else of interest in your codebase.
- You will be giving a demo of your circuits before class on the day of the deadline. Be sure to bring everything you need to show it off.
- Be sure to write clear and concise commit messages outlining what has been done.
- Write clean and simple code, using comments to explain what is not intuitive. If the grader cannot understand your code, you will lose credit on the assignment.
- Draw clean Fritzing schematics. If the grade cannot understand them, you will lose credit as well.
- Be sure your code compiles! If your sketches do not compile, you will receive **no credit**. It is better to submit a working sketch that only does a subset of the requirements than a broken one that attempts to do them all.

Table 1: Grading Rubric

Category	Percentage
Demo	60%
Code Quality (including `Makefile`)	15%
Write-Up & Protocol Design	10%
Compilation with `-Wall -pedantic -Werror`	10%
Schematic	5%

[1] http://fritzing.org/home/