# Final Project (Part 2)

CS 3410 Systems Programming

**Due Date:** Monday April 10, 2017 at 6:00 PM

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

### Background Information

One goal of this project is to detect when the current heart rate is an outlier. We will accomplish 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, your heart rate will be lower during the night (when you're asleep) than it will be during your daily 8 AM gym trip. Therefore, we will actually store a separate histogram for different parts of the day, in order to more accurately compare heart rate readings. 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 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 will use a function called `mmap` to load the contents of a file into our program's virtual memory space. Use the `open` function to obtain a file descriptor for the backing file and pass the result to `mmap` to load it into memory. Be sure to use `munmap` and `close` on the mapping and file descriptor, respectively. For more information, run `man 2 mmap` on any UNIX system.

Finally, we will now integrate an $I^2C$ real-time clock into our system. Recall that we have worked with this device previously in the course. We will use this device to provide precise timing information for our heart rate readings. When the Arduino starts up, it should synchronize timing information with the host program over the serial connection. Take note that a timestamp is larger than a single byte, so you will need some way to ensure that the correct number of bytes are read (and in the proper order). These are the first step towards building a communication protocol, an idea we will expand upon further in the rest of the course.

### Specification (Part 2)

In this part, we will focus on building one layer of data storage on the host machine and the communication protocol to and from the Arduino. Your program should perform the following:

- Integrate the $I^2C$ real-time clock into the system. You will need to ensure the RTC and host computer's are synchronized. For example, you should be able to translate a value from the RTC into one that makes sense in context of the host computer's clock.
- Build the histogram structure described above into a `mmap`'d file.
- The host program should request a heart rate value from the Arduino every second or so. The Arduino should send the heart rate reading (in beats per minute) as well as more precise timing information from the RTC back. The resulting information should be stored in the histogram
- If the host program detects the current reading is an outlier (by some method you determine), send a command to the Arduino to flash a warning message on your output device. Otherwise, follow the behavior from part 1 (show the current heart rate).
- Every 10 seconds, the host program should print out a graphical representation of the current time's histogram structure.

### Tips and Tricks

- Really think about the communication protocol. How do you design one that is reliable, correct, **and** robust. This is difficult to do, especially when you are sending messages longer than one byte (which you must do here). Draw out the different types of messages, possibly separated into requests and responses.
- Additional background information on communication protocols will be posted closer to the deadline. You should still start before that time, however, since you should still attempt to design your own protocol first. If you wait until then to start, you will find you won't have much time and much misery will follow.
- Since the histogram is stored in a `mmap`'d file, you should be able to open it and see its raw bytes in any text editor. This may be useful for debugging purposes.
- You may notice that the command line interface from part 1 is not required here, although many other parts are. I would recommend keeping that code in your program and extending it for debugging purposes (like reading the current value from the RTC and displaying a bucket in the histogram structure). For example, you may want to run the "debug console" if passed a `--debug` flag on the command line. For more information on reading command line arguments, see getopt.

### 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 `part2`:
  - 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 circuit you built.
  - A document explaining the following things:
    - The schema for your communication protocol (how are messages structured and how are you ensuring reliability, correctness, and robustness).
    - How are you detecting outlier readings
- 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 | 50% |
| Write-Up & Protocol Design | 20% |
| Code Quality (including `Makefile`) | 15% |
| Compilation with `-Wall -pedantic -Werror` | 10% |
| Schematic | 5% |

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