# ONLINE Final Project (Part 3)

CS 3410 Systems Programming

## Project: Simulated Weather Analyzer

**Due Date:** Monday April 24, 2019 at 11:59 PM

This is an individual assignment.

### Background Information

We will now integrate the user interaction from part 1 and the data gathering from part 2. For part 3, we will have two processes running: one in the background (doing sensor gathering and updating the mmapped data), and one in the foreground (obtinaing input from the user and printing visualizations of the data to screen). They will exchange information through two communication pipes: the sensor_pipe, which conveys user messages to the background process; and the data_pipe, which returns the gathered data from the background to the foreground process.

### Specification (Part 3)

You will use the provided template and expand it to satisfy the speciffication. The running programm will do some setting up and will then fork into twp processes running the foreground (child) and background (parent) tasks.

The command line interface will expose the following functions:

- `blink X`: Causes the Arduino to rapidly blink X times (the frequency is fixed).
- `pause`: Pause the background process from requesting data and causes the Arduino to set the LED to LOW.
- `resume`: Resume the background process requesting data and causes the Arduino to alternate the LED HIGH and LOW every second. Note that this causes the background process to request data from the Arduino (by sending the command linked to a REQUEST).
- `env`: Request data package as usual and additionally print the returned packet to stdout. Note that, in addition to the normal request done during the *resume* stage, this casues the printing of the reply in the terminal.
- `hist t`: prints the whole 2D histogram for temperature in the terminal
- `hist p`: prints the whole 2D histogram for pressure in the terminal
- `hist h`: prints the whole 2D histogram for humidity in the terminal
- `record`: prints the contents of the record mmapped memory (so far) to the terminal.
- `exit`: Exits the bacckground program.

The commands `pause`, `resume`, `blink X`, and `env` all cause messages to be sent to the Arduino. The remaining commands extract information from the (shared) mmapped data (histograms and the record) and print it to the terminal.

The commands `pause`, `resume`, `blink X`, and `env` cause the child to forward the comand to the parent using the signal_pipe.

The background process will perform the same tasks as before (requesting data from the Arduino once a second) but will first check for user commands. We will use the **select-and-read sequence** shown in pipe3.c of lecture 18 to wait for 1 second for user input arriving from the child process through the signal_pipe. If a command is read, then it issues the appropriate command to the Arduino and continues, otherwise it just

continues. Upon continuing, if the background process is not paused and a full second has passed since the last request, it requests a new data package from the Arduino and updtates the histograms and the record.

### 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 `part3`:
- Inside part3, you will have the exact same structure as for part2: the folders `sensorsoftware` and one `hostsoftware` with the same contents.
- All your work will happen on the host side where **you ONLY need to provide a new version of host.c**. You only need to fill out the template that we provided by **filling every section marked with a TODO**.
- Note that the resulting histograms and records should be identical to part 2, which will help you verify the correctness of your code.
- 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.
- Be sure your code compiles! If your code does not compile, you will receive **no credit**. It is better to submit a working program that only does a subset of the requirements than a broken one that attempts to do them all.
- You must create a **one page Write-Up** explaining how the system works in a way that is easily followed by a casual reader. You may use state-diagrams or flowcharts to illustrate the steps followed.

Table 1: Grading Rubric

| Category | Percentage |
|---|---|
| Demo (works perfectly) | 80% |
| Code Quality | 10% |
| Write-Up | 10% |

An example sequence of command output is shown in figure 1:

```
pfrank@ubuntu:~/cs3410/s-20/onlineProject/part3/part3_solution/hostsoftware$ make clean
rm -f *.o host *.bin
pfrank@ubuntu:~/cs3410/s-20/onlineProject/part3/part3_solution/hostsoftware$ make run
gcc -Werror -Wextra -Wall -pedantic -std=c99 -g -O0   -c -o host.o host.c
gcc   host.o   -o host
./host
Initializing Host-side Processes
Beginning Sensor Reading
->pause
[main_loop_cli] writing to signal_pipe: pause
->resume
[main_loop_cli] writing to signal_pipe: resume
->env
        Arduino Reply (consumed: 16):   :055, 187, 041, 001, 202001020400
->env
        Arduino Reply (consumed: 16):   :128, 120, 122, 000, 202001020600
->env
        Arduino Reply (consumed: 16):   :173, 071, 211, 001, 202001020800
->record
  029, 087, 185, 000, 202001012300,
  038, 127, 117, 001, 202001020000,
  018, 130, 078, 000, 202001020100,
  035, 173, 021, 001, 202001020200,
  051, 178, 021, 000, 202001020300,
  055, 187, 041, 001, 202001020400,
  098, 133, 063, 000, 202001020500,
  128, 120, 122, 000, 202001020600,
  169, 096, 165, 001, 202001020700,
  173, 071, 211, 001, 202001020800,
  200, 084, 235, 001, 202001020900,
  207, 085, 219, 001, 202001021000,
```

```
->hist p
Hour|   16   32   48   64   80   96  112  128  114  160  176  192  208  224  240  256
-------------------------------------------------------------------------------------
  0 |    0    0    1    0    0    0    0    0    0    0    0    0    0    0    0    0
  1 |    0    1    0    0    0    0    0    0    0    0    0    0    0    0    0    0
  2 |    0    0    1    0    0    0    0    0    0    0    0    0    0    0    0    0
  3 |    0    0    0    1    0    0    0    0    0    0    0    0    0    0    0    0
  4 |    0    0    0    1    0    0    0    0    0    0    0    0    0    0    0    0
  5 |    0    0    0    0    0    0    1    0    0    0    0    0    0    0    0    0
  6 |    0    0    0    0    0    0    0    0    1    0    0    0    0    0    0    0
  7 |    0    0    0    0    0    0    0    0    0    0    1    0    0    0    0    0
  8 |    0    0    0    0    0    0    0    0    0    0    1    0    0    0    0    0
  9 |    0    0    0    0    0    0    0    0    0    0    0    0    1    0    0    0
 10 |    0    0    0    0    0    0    0    0    0    0    0    0    1    0    0    0
 11 |    0    0    0    0    0    0    0    0    0    0    0    0    0    0    1    0
 12 |    0    0    0    0    0    0    0    0    0    0    0    0    0    1    0    0
 13 |    0    0    0    0    0    0    0    0    0    0    0    0    0    1    0    0
 14 |    0    0    0    0    0    0    0    0    0    0    0    0    1    0    0    0
 15 |    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
 16 |    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
 17 |    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
 18 |    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
 19 |    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
 20 |    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
 21 |    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
 22 |    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
 23 |    0    1    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    -------------------------------------------------------------------------------

->exit
[main_loop_cli] writing to signal_pipe: exit
[main_loop_data] Got: exit!
```