Dawson College – Electronics Engineering Technology
243-698-DW Computer Network Project

Project Report#2

Configuring and Monitoring System for Server Room

Student Name: Yihuan Zhang
Student Number: 1630783
Date created: March 21, 2018
Date submitted: March 28, 2019
Submitted to: Nick Markou

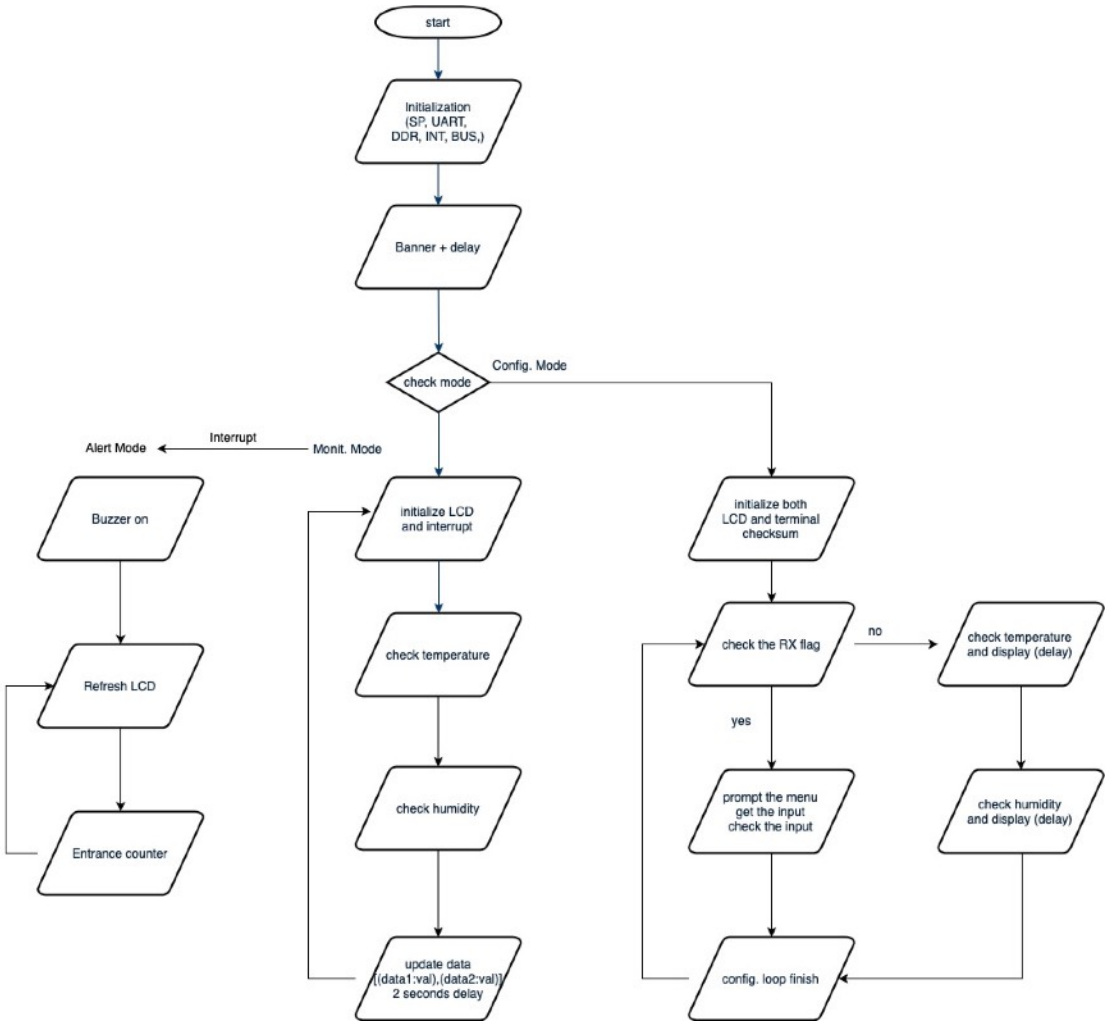
Objective

- To continue the Python program in serial communication;
- To log the data from the Python program using a bash script;
- To configure the execution schedule in the crontab;
- To mill the power supply and embedded system PCB boards;
- To finish the functions of web server.

Results

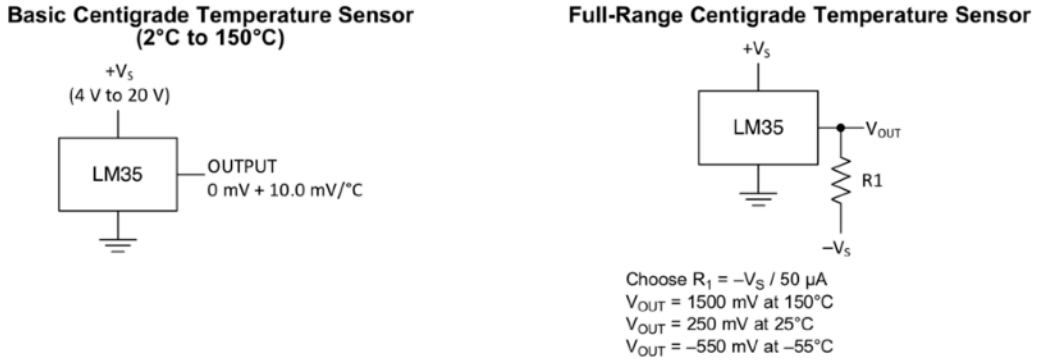
Embedded System

The embedded part has been modified during the second stage of my project, in order to adapt the serial communication with RPi. In “Monit.” mode, the system is configured only to send the data every 2 seconds. I removed the menu and simplified the terminal interface in this mode so that it would be easier for python program to grep the data and there would also be less chance of error occurring. This “Monit.” mode is designed for python API (see the next section for further description), correspondingly the other one, “Config.” Mode, still kept the menu in serial communication as well as having data showed on LCD.



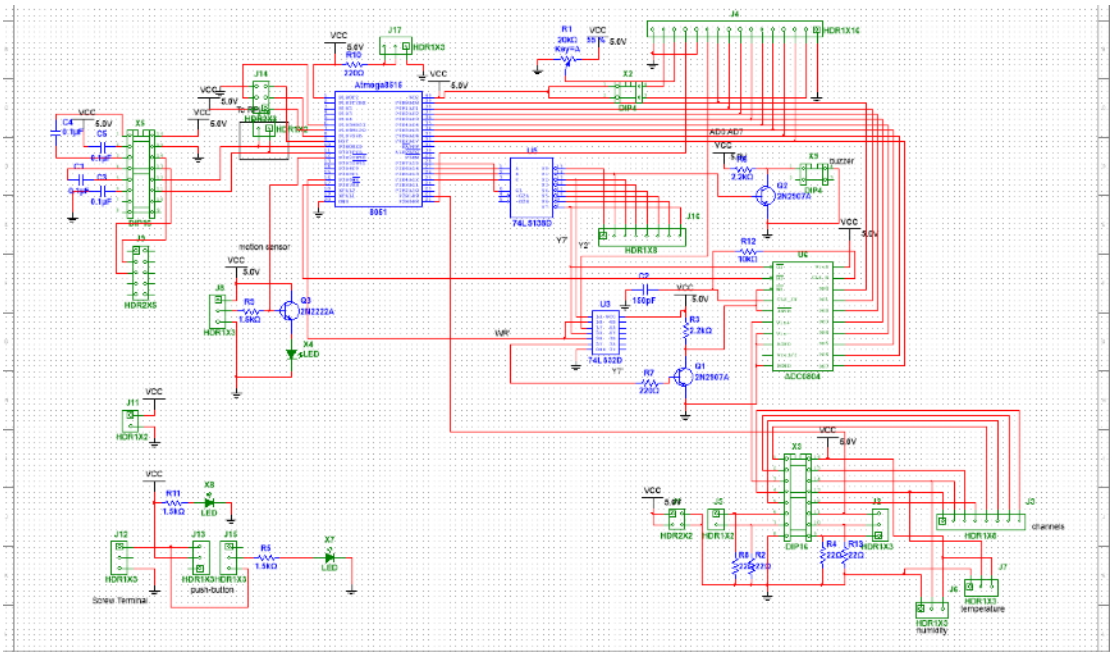
(Figure 1, Micro-controller flow chart)

The LM35 has been chosen as the the temperature sensor. MCP9701A and other thermistors were considered, but were aborted due their non-linear thermal resistance and additional components. In general, the LM35 is an analog temperature sensor using package TO-220(3) that provides a linear scale factor from -55℃ to 150℃, without calibration.



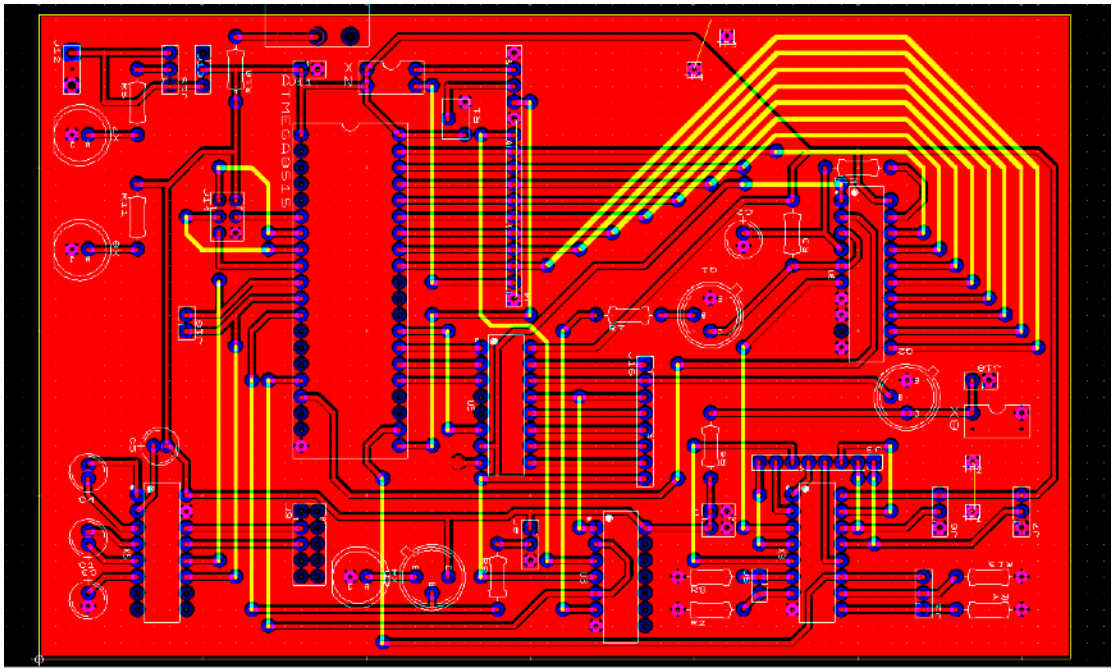
(Figure 2, circuitries of the LM35)

The embedded system uses “Basic Centigrade Temperature Sensor” configuration to monitor the server room environment with the range between 2℃ and 150℃.



(Figure 3, the wiring diagram of the embedded system)

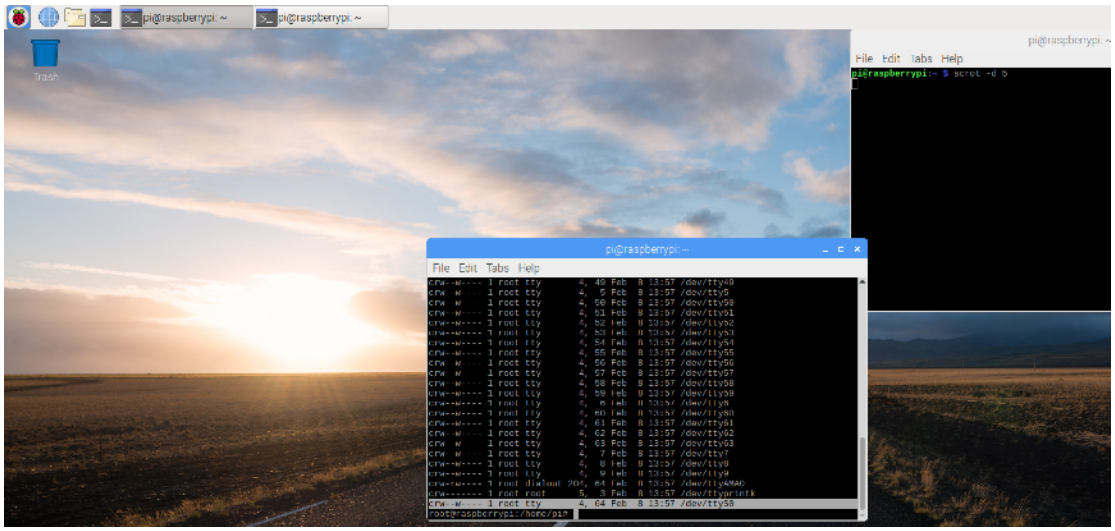
The wiring diagram has all the footprints assigned, certain components such as LCD and sensors are substituted with headers, two LCDs are added to indicate the power status. Also, the on-board SPI header offers the interface to MKII programmer, which allows the AtmelStudio writing the program into the Micro-controller directly. Address Bus and analog MUX channels are reserved for future expansion and troubleshooting through the headers.



(Figure 4, the PCB diagram of the embedded system)

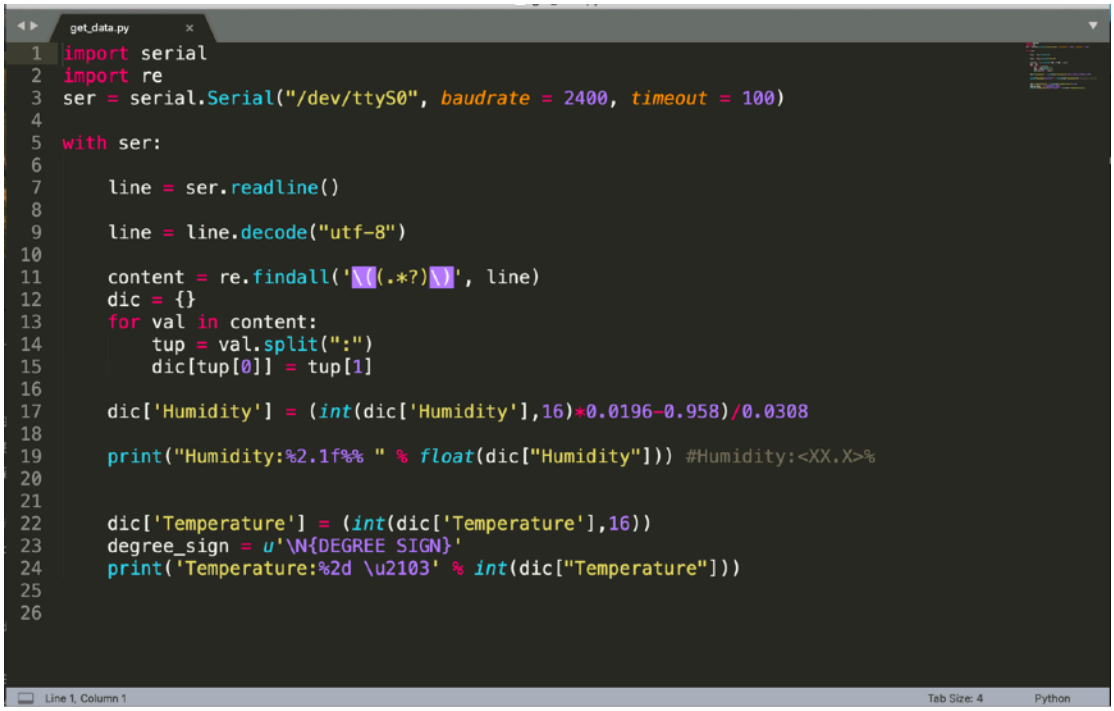
The PCB is designed as a 6X9 inches outline double-side board. The power plane on the bottom side is used as ground. Top side is mainly used for vertical routes and data address. The test points (TP1, TP2, TP3, TP4) are in the ground net which interconnect each other (isolated ground). Top left are the power section, the Vcc can be either supplied from the screw terminal (J12) or from the header (J11). In addition, the other one can be used as the power source for other circuitry if there is power already connected in.

Raspberry Pi



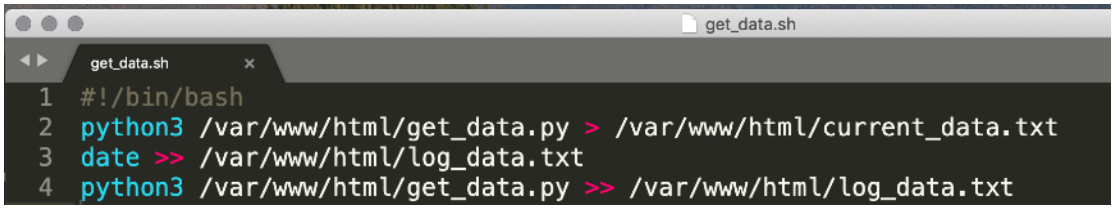
(Figure 5, the ttyS0 for serial communication)

Since in this mode the micro controller is periodically sending data in a form of [(Humidity:XX),(Temperature:XX)], the Python program is written to capture the keywords (diction) through the GPIO serial port ttyS0 and to convert the raw data (in hex) into decimal (Celsium & percentage).



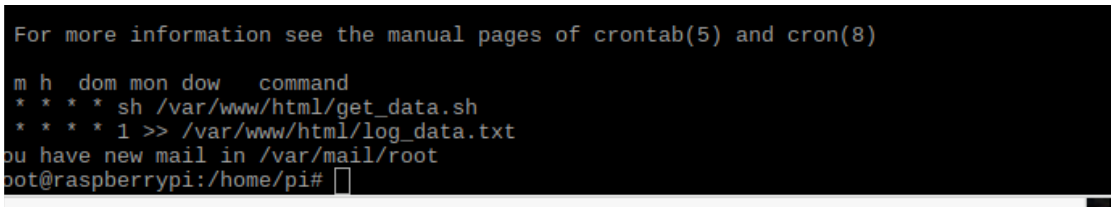
(Figure 6, the python program “get_data.py”)

On the Raspberry Pi, the python program “get_data” is not only used to establish the serial communication, but also reads the data. The RPi stores ASC II characters in a variable “line”, then transfers into another variable “content”. The “diction” splits the two words, converts the variables into integers before displaying them. Therefore, whenever this program is executed, it will take the next latest data from embedded system through the ttyS0 and print it to the terminal in form of “Humidity: XX.X% /n Temperature XX°C”.



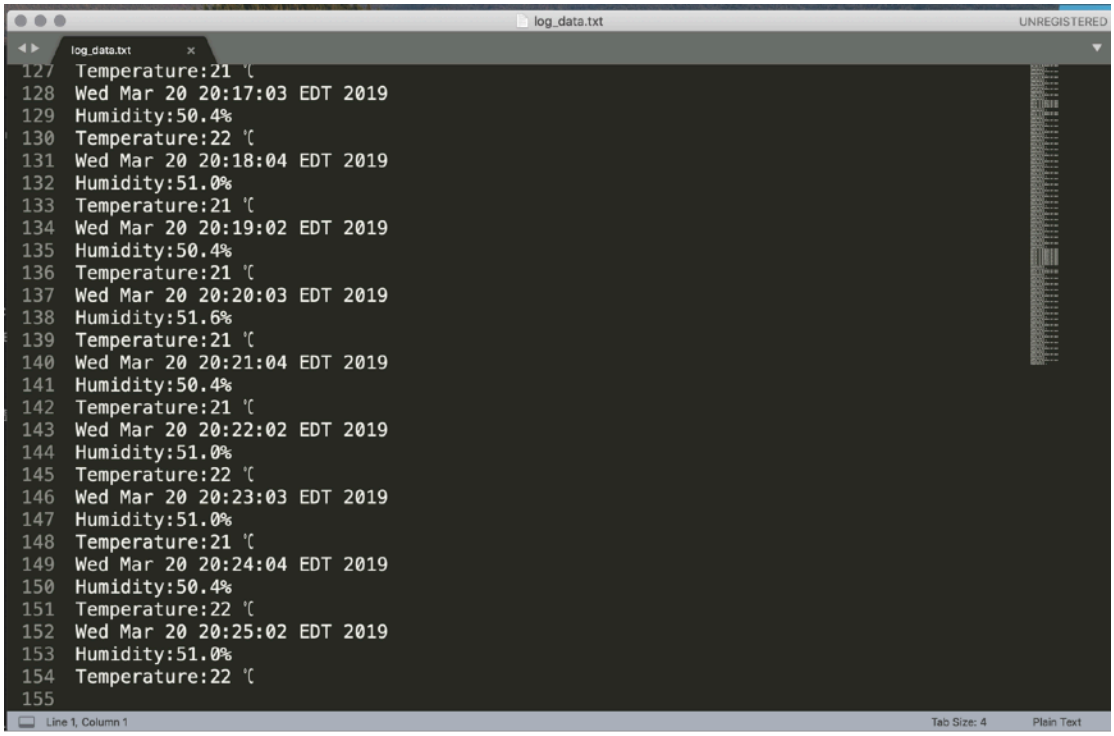
(Figure 7, the bash script “get_data.py”)

In order to log the output data and save it into a log file, a bash script is created and runs periodically. The script “get_data.sh” calls the “get_data.py” program and save it into “current_data.txt” file (overwriting), as well as log it into “log_data.txt” with a time stamp.



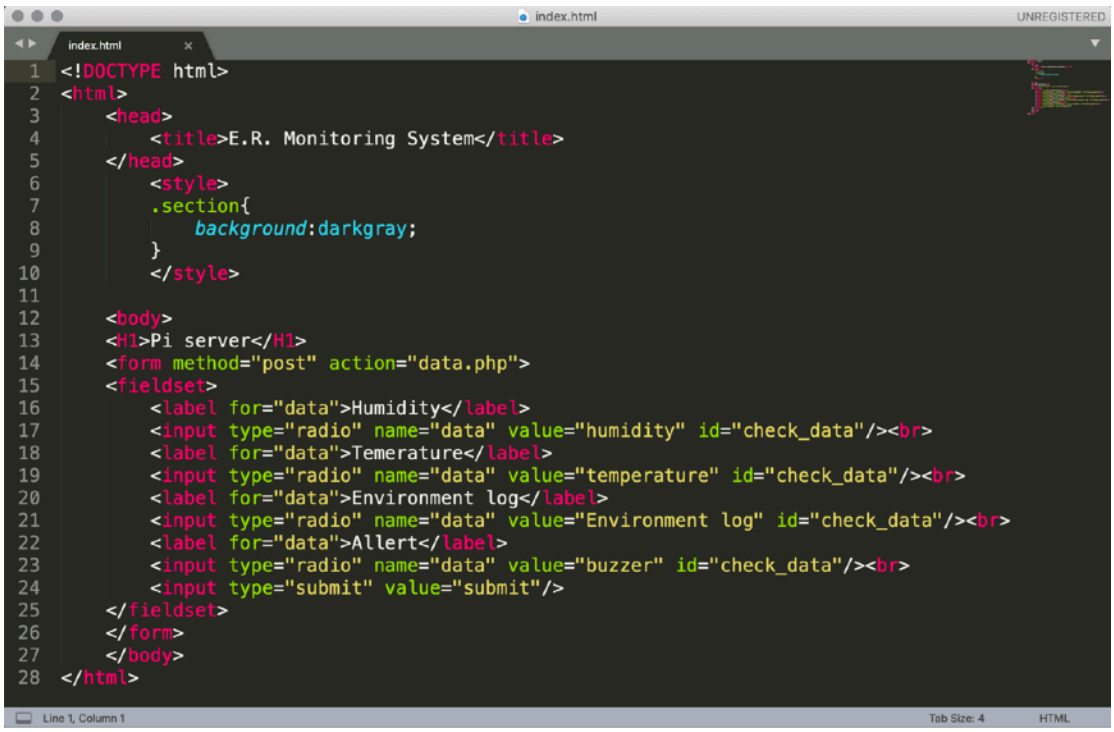
(Figure 8, the crontab configuration)

In crontab file, the command is scheduled as being executed every minute.



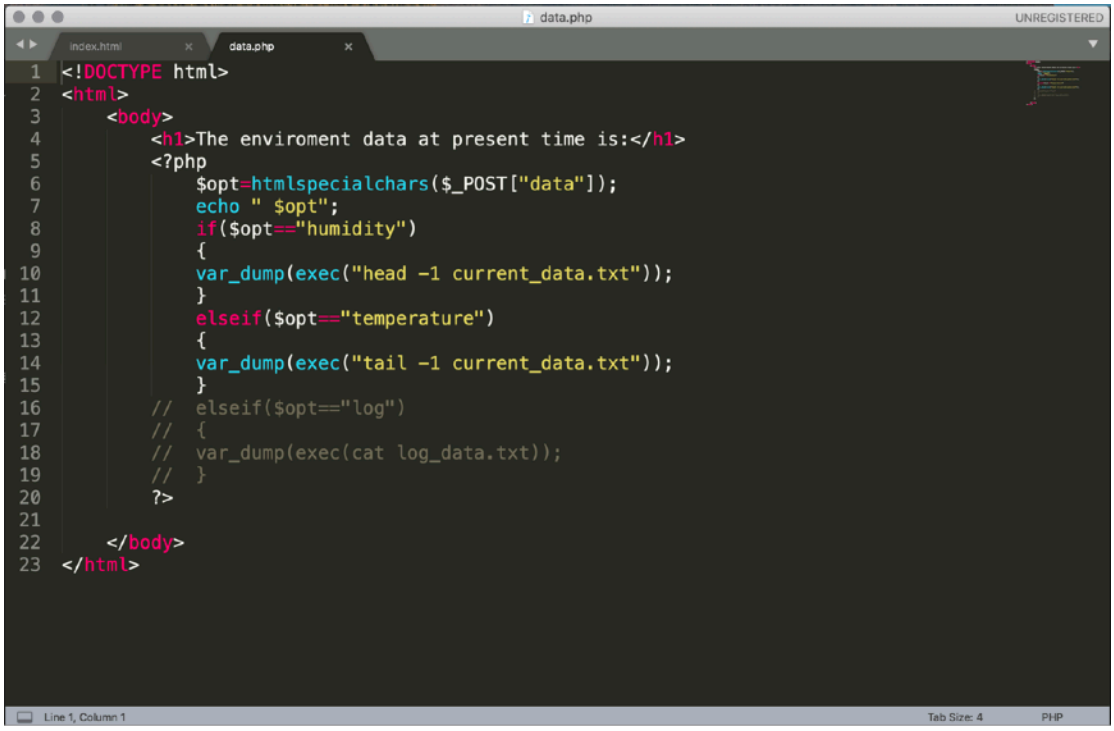
(Figure 9, the results in “log_data.txt”)

as the file records both temperature and humidity data every minute, the change of server room environment is shown in time domain.



(Figure 10, the “index.html”)

As for the prototype of the web server, the front-end submits a form to the back-end (“data.php”), the PHP takes actions whether to retrieve the data, log or sends an instruction to enter into the alert mode (activating buzzer).

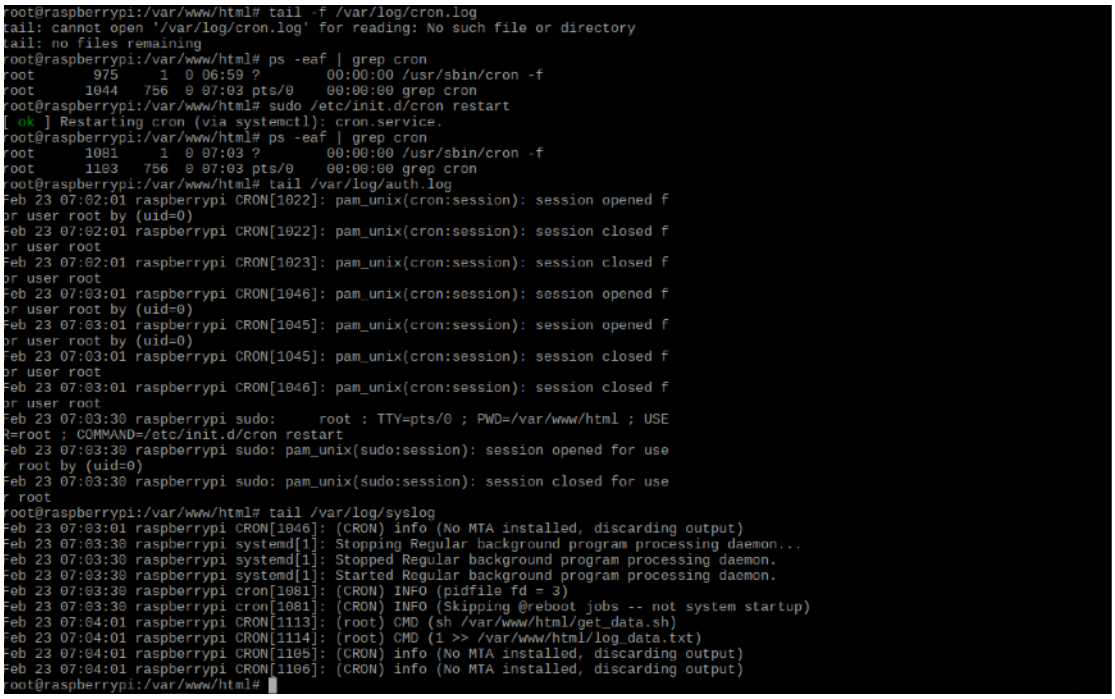


(Figure 11, the “data.php”)

After receiving requests from HTML, the back-end uses “htmlspecialchars” to filter out any HTML keywords or tags in order to prevent any malicious commands. Then the “data” stores into a variable “$opt”, different actions under different conditions are taken according to its value.

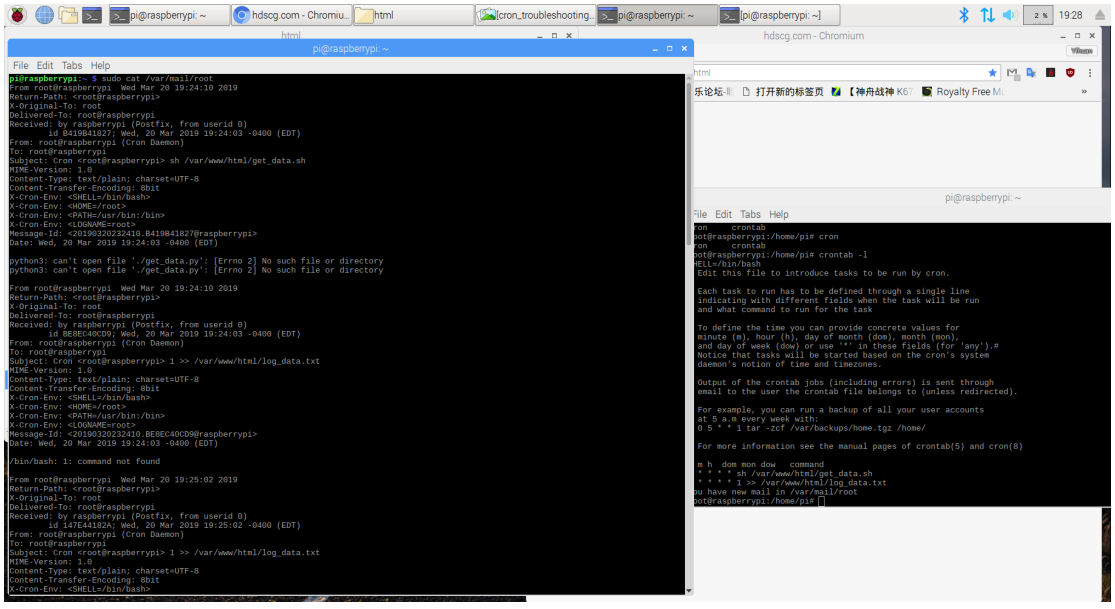
Troubleshooting

Troubleshooting for this part is mainly about python program and RPi configuration. The use of “PySerial” library, “diction”, “split()”, hex-to-int convert, special characters (formatting) are referred to “Stack OverFlow”, “W3School” and “Python Software Foundation” (Dr. Andrew N. Harrington, Chris Liechti, etc).



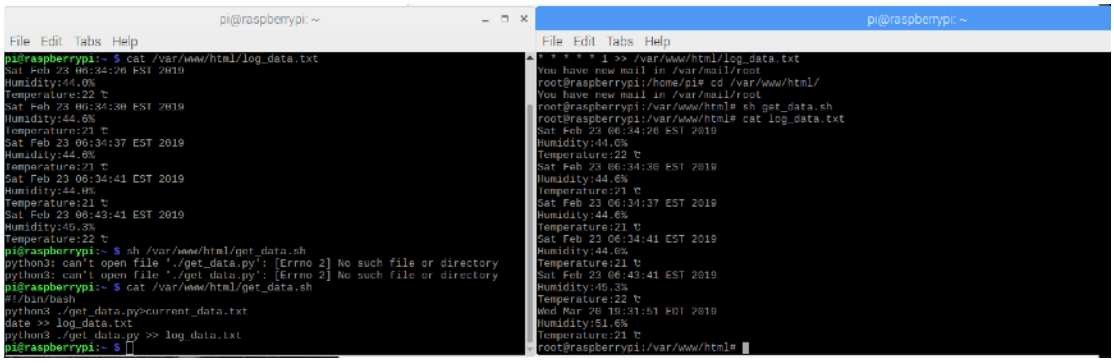
(Figure 10, the “auth.log” and “syslog”)

At the beginning, the crontab did not work. The bash script do records the data into “log_data.txt” if it’s executed, therefore I was assuming the problem occurred because of the crontab (the lack of MTA in the latest message of syslog).



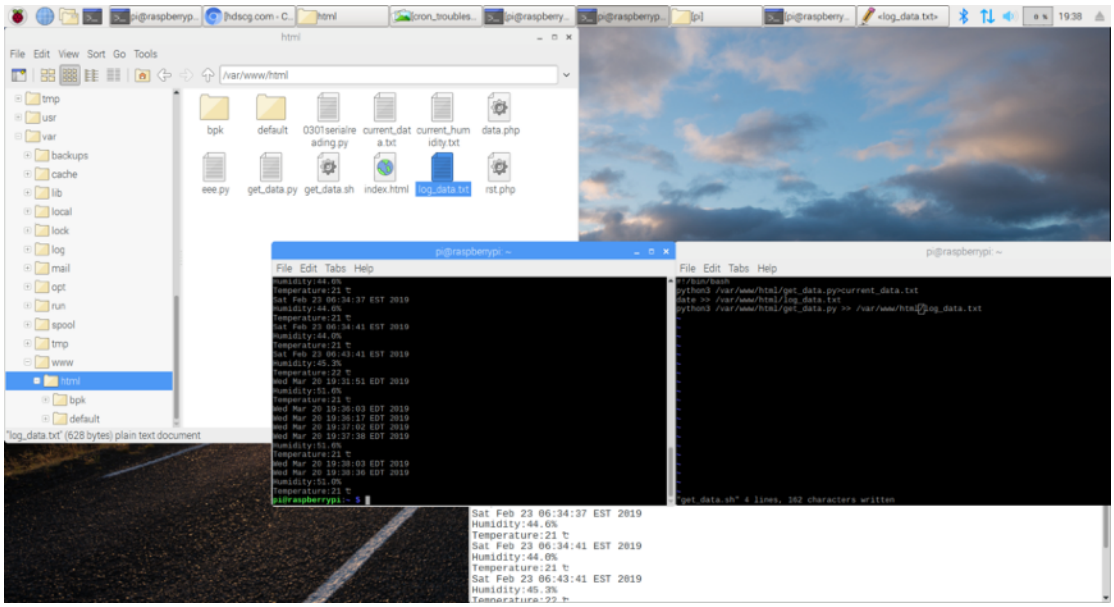
(Figure 11, the “var/mail/root”)

After installing the MTA, I noticed the log message“python3: can’t open file ‘./get_data.py’: [ERROR 2] No such file or directory” in the “var/mail/root” file. I then went back to the bash script again.



(Figure 12, the error message)

I executed the program again, found the root reason of the problem: the “get_data.sh” could not find the log files and python program when crontab is executing it outside of the directory.



(Figure 13, the result of troubleshooting)

So soon as I changed the relative path to the absolute path in the “get_data.sh” script, the crontab started executing it, and the “get_data.log” started updating the data.