

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

**STATEMENT OF ORIGINAL WORK: I HEREBY ATTEST THAT THIS REPORT IS
ENTIRELY MY OWN ORIGINAL WORK, EXCEPT FOR EXCERPTS THAT HAVE BEEN
EXPRESSLY CITED AND ATTRIBUTED TO THE ORIGINAL AUTHOR.**

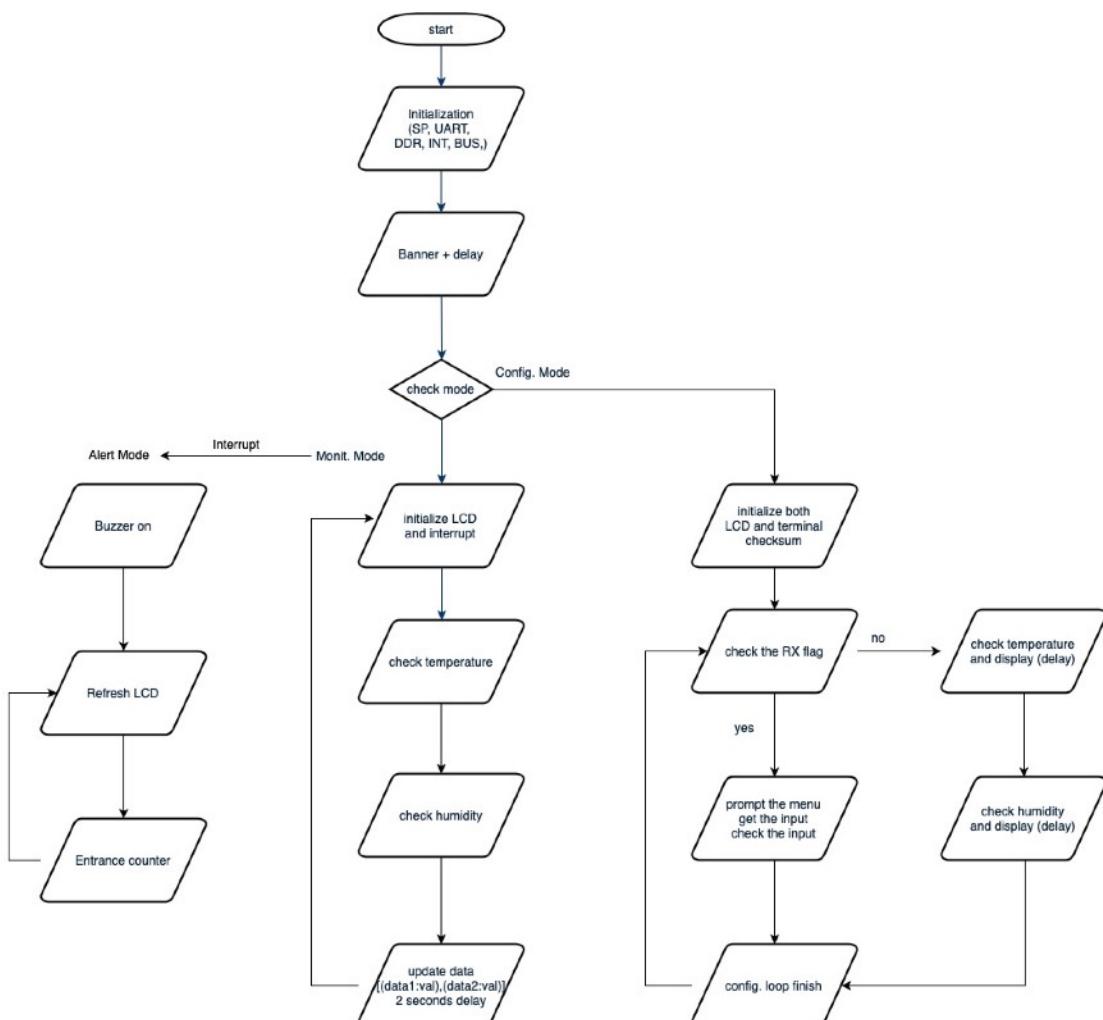
Objective

- 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

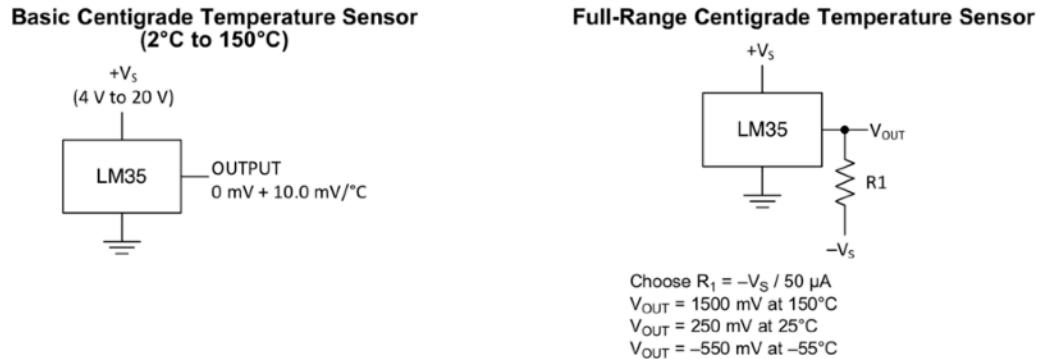
Embedded System

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.



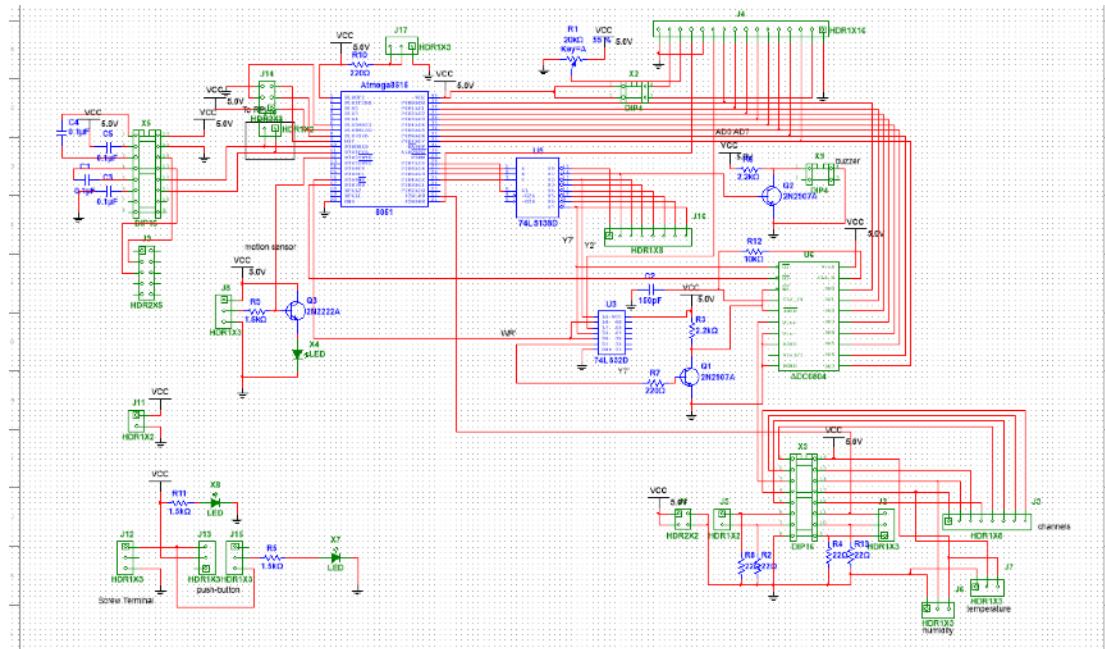
(Figure 1, Micro-controller flow chart)

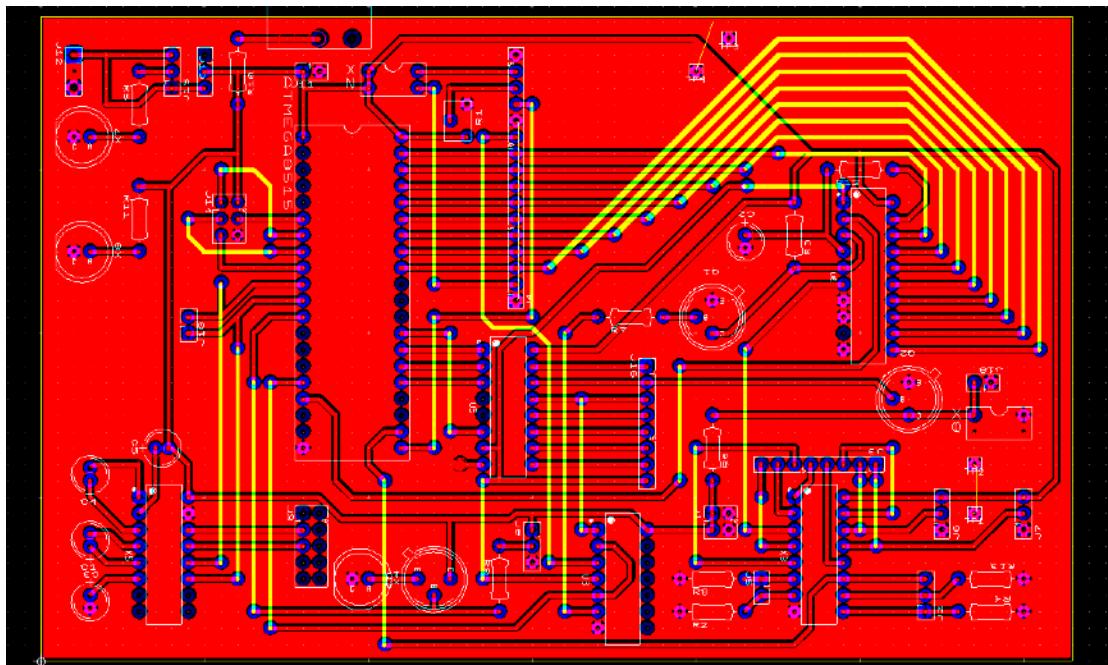
The LM35 has been chosen as the the temperature sensor. MCP9701A and other thermistors were considered, but were aborted due their non-linear thermal resistance and additional components. In general, the LM35 is an analog temperature sensor using package TO-220(3) that provides a linear scale factor from -55°C to 150°C, 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°C and 150°C.

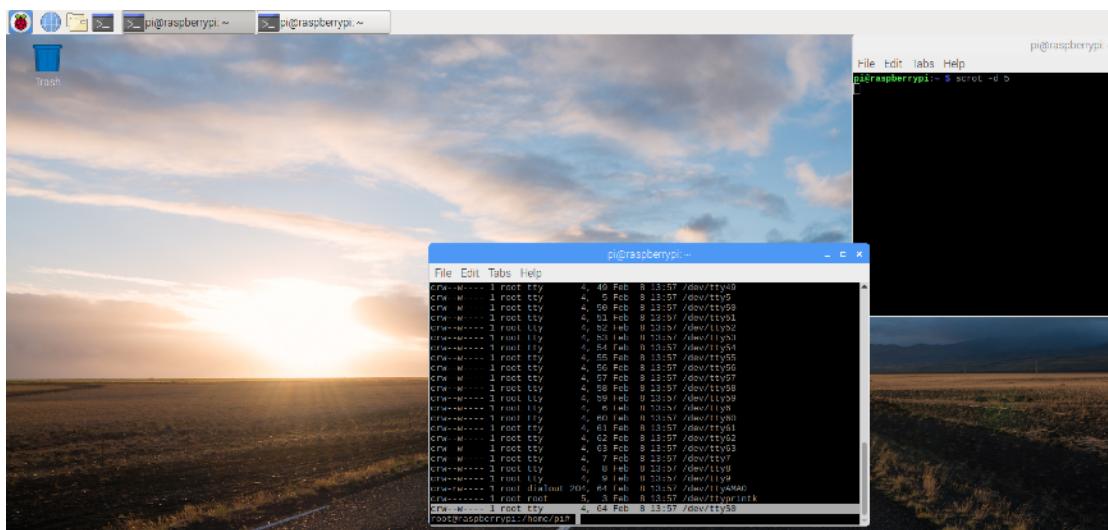




(Figure 4, the PCB diagram of the embedded system)

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 (Celsius & percentage).

```

1 import serial
2 import re
3 ser = serial.Serial("/dev/ttyS0", baudrate = 2400, timeout = 100)
4
5 with ser:
6
7     line = ser.readline()
8
9     line = line.decode("utf-8")
10
11    content = re.findall('(\n(.*)\n)', line)
12    dic = {}
13    for val in content:
14        tup = val.split(":")
15        dic[tup[0]] = tup[1]
16
17    dic['Humidity'] = (int(dic['Humidity'],16)*0.0196-0.958)/0.0308
18
19    print("Humidity:%2.1f% " % float(dic["Humidity"])) #Humidity:<XX.X>%
20
21
22    dic['Temperature'] = (int(dic['Temperature'],16))
23    degree_sign = u'\N{DEGREE SIGN}'
24    print('Temperature:%2d \u00b0' % int(dic["Temperature"]))
25
26

```

(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”.

```

#!/bin/bash
python3 /var/www/html/get_data.py > /var/www/html/current_data.txt
date >> /var/www/html/log_data.txt
python3 /var/www/html/get_data.py >> /var/www/html/log_data.txt

```

(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.

```

For more information see the manual pages of crontab(5) and cron(8)

m h  dom mon dow   command
* * * * * sh /var/www/html/get_data.sh
* * * * 1 >> /var/www/html/log_data.txt
You have new mail in /var/mail/root
root@raspberrypi:/home/pi# 

```

(Figure 8, the crontab configuration)

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

```

log_data.txt
127 Temperature:21 °C
128 Wed Mar 20 20:17:03 EDT 2019
129 Humidity:50.4%
130 Temperature:22 °C
131 Wed Mar 20 20:18:04 EDT 2019
132 Humidity:51.0%
133 Temperature:21 °C
134 Wed Mar 20 20:19:02 EDT 2019
135 Humidity:50.4%
136 Temperature:21 °C
137 Wed Mar 20 20:20:03 EDT 2019
138 Humidity:51.6%
139 Temperature:21 °C
140 Wed Mar 20 20:21:04 EDT 2019
141 Humidity:50.4%
142 Temperature:21 °C
143 Wed Mar 20 20:22:02 EDT 2019
144 Humidity:51.0%
145 Temperature:22 °C
146 Wed Mar 20 20:23:03 EDT 2019
147 Humidity:51.0%
148 Temperature:21 °C
149 Wed Mar 20 20:24:04 EDT 2019
150 Humidity:50.4%
151 Temperature:22 °C
152 Wed Mar 20 20:25:02 EDT 2019
153 Humidity:51.0%
154 Temperature:22 °C
155

```

(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.

```

index.html
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>E.R. Monitoring System</title>
5   </head>
6   <style>
7     .section{
8       background:darkgray;
9     }
10  </style>
11
12 <body>
13 <H1>Pi server</H1>
14 <form method="post" action="data.php">
15   <fieldset>
16     <label for="data">Humidity</label>
17     <input type="radio" name="data" value="humidity" id="check_data"/><br>
18     <label for="data">Temerature</label>
19     <input type="radio" name="data" value="temperature" id="check_data"/><br>
20     <label for="data">Environment log</label>
21     <input type="radio" name="data" value="Environment log" id="check_data"/><br>
22     <label for="data">Allert</label>
23     <input type="radio" name="data" value="buzzer" id="check_data"/><br>
24     <input type="submit" value="submit"/>
25   </fieldset>
26 </form>
27 </body>
28 </html>

```

(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).

The screenshot shows a code editor window with two tabs: 'index.html' and 'data.php'. The 'data.php' tab is active, displaying the following PHP code:

```
1 <!DOCTYPE html>
2 <html>
3     <body>
4         <h1>The environment data at present time is:</h1>
5         <?php
6             $opt=htmlspecialchars($_POST["data"]);
7             echo " $opt";
8             if($opt=="humidity")
9             {
10                 var_dump(exec("head -1 current_data.txt"));
11             }
12             elseif($opt=="temperature")
13             {
14                 var_dump(exec("tail -1 current_data.txt"));
15             }
16             // elseif($opt=="log")
17             //
18             // var_dump(exec(cat log_data.txt));
19             //
20         ?>
21
22     </body>
23 </html>
```

The code is designed to handle POST requests for 'data'. It filters the 'data' parameter using `htmlspecialchars`. Depending on the value of '\$opt', it either outputs the contents of 'current_data.txt' or 'log_data.txt' using `var_dump` and `exec`.

(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.

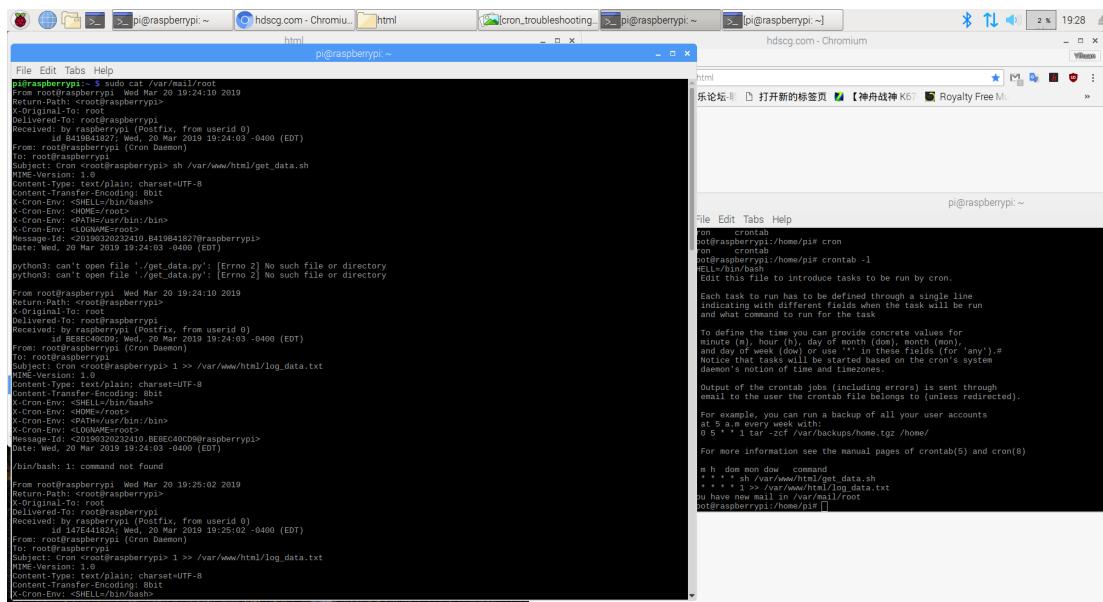
Troubleshooting

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).

```
root@raspberrypi:/var/www/html# tail -f /var/log/cron.log
tail: cannot open '/var/log/cron.log' for reading: No such file or directory
tail: no files remaining
root@raspberrypi:/var/www/html# ps -ef | grep cron
root      975     1  0 06:59 ?        00:00:00 /usr/sbin/cron -f
root     1044   756  0 07:03 pts/0    00:00:00 grep cron
root@raspberrypi:/var/www/html# sudo /etc/init.d/cron restart
[ ok ] Restarting cron (via systemctl): cron.service.
root@raspberrypi:/var/www/html# ps -ef | grep cron
root      1081     1  0 07:03 ?        00:00:00 /usr/sbin/cron -f
root     1183   756  0 07:03 pts/0    00:00:00 grep cron
root@raspberrypi:/var/www/html# tail /var/log/auth.log
Feb 23 07:02:01 raspberrypi CRON[1022]: pam_unix(cron:session): session opened f
or user root by (uid=0)
Feb 23 07:02:01 raspberrypi CRON[1022]: pam_unix(cron:session): session closed f
or user root
Feb 23 07:02:01 raspberrypi CRON[1023]: pam_unix(cron:session): session closed f
or user root
Feb 23 07:03:01 raspberrypi CRON[1046]: pam_unix(cron:session): session opened f
or user root by (uid=0)
Feb 23 07:03:01 raspberrypi CRON[1045]: pam_unix(cron:session): session opened f
or user root by (uid=0)
Feb 23 07:03:01 raspberrypi CRON[1045]: pam_unix(cron:session): session closed f
or user root
Feb 23 07:03:01 raspberrypi CRON[1046]: pam_unix(cron:session): session closed f
or user root
Feb 23 07:03:01 raspberrypi CRON[1046]: pam_unix(cron:session): session closed f
or user root
Feb 23 07:03:30 raspberrypi sudo:      root : TTY:pts/0 ; PWD=/var/www/html ; USE
-troot ; COMMAND=/etc/init.d/cron restart
Feb 23 07:03:30 raspberrypi sudo: pam_unix(sudo:session): session opened for use
by root by (uid=0)
Feb 23 07:03:30 raspberrypi sudo: pam_unix(sudo:session): session closed for use
by root
root@raspberrypi:/var/www/html# tail /var/log/syslog
Feb 23 07:03:01 raspberrypi CRON[1046]: (CRON) info (No MTA installed, discarding output)
Feb 23 07:03:30 raspberrypi systemd[1]: Stopping Regular background program processing daemon...
Feb 23 07:03:30 raspberrypi systemd[1]: Stopped Regular background program processing daemon.
Feb 23 07:03:30 raspberrypi cron[1081]: (CRON) INFO (pidfile fd = 3)
Feb 23 07:03:30 raspberrypi cron[1081]: (CRON) INFO (Skipping @reboot jobs -- not system startup)
Feb 23 07:04:01 raspberrypi CRON[1113]: (root) CMD (sh /var/www/html/get_data.sh)
Feb 23 07:04:01 raspberrypi CRON[1114]: (root) CMD (1 > /var/www/html/log_data.txt)
Feb 23 07:04:01 raspberrypi CRON[1105]: (CRON) info (No MTA installed, discarding output)
Feb 23 07:04:01 raspberrypi CRON[1106]: (CRON) info (No MTA installed, discarding output)
root@raspberrypi:/var/www/html#
```

(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.

```

pi@raspberrypi:~ $ cat /var/www/html/log_data.txt
Sat Feb 23 06:34:26 EST 2019
Humidity:44.0%
Temperature:21 °C
Sat Feb 23 06:34:38 EST 2019
Humidity:44.0%
Temperature:21 °C
Sat Feb 23 06:34:37 EST 2019
Humidity:44.0%
Temperature:21 °C
Sat Feb 23 06:34:41 EST 2019
Humidity:45.0%
Temperature:21 °C
Sat Feb 23 06:34:43 EST 2019
Humidity:45.0%
Temperature:21 °C
pi@raspberrypi:~ $ sh /var/www/html/get_data.sh
python3: can't open file './get_data.py': [Errno 2] No such file or directory
pi@raspberrypi:~ $ cat /var/www/html/get_data.sh
#!/bin/bash
python3 ./get_data.py>current_data.txt
date >> log_data.txt
python3 ./get_data.py >> log_data.txt
pi@raspberrypi:~ [ ]

```

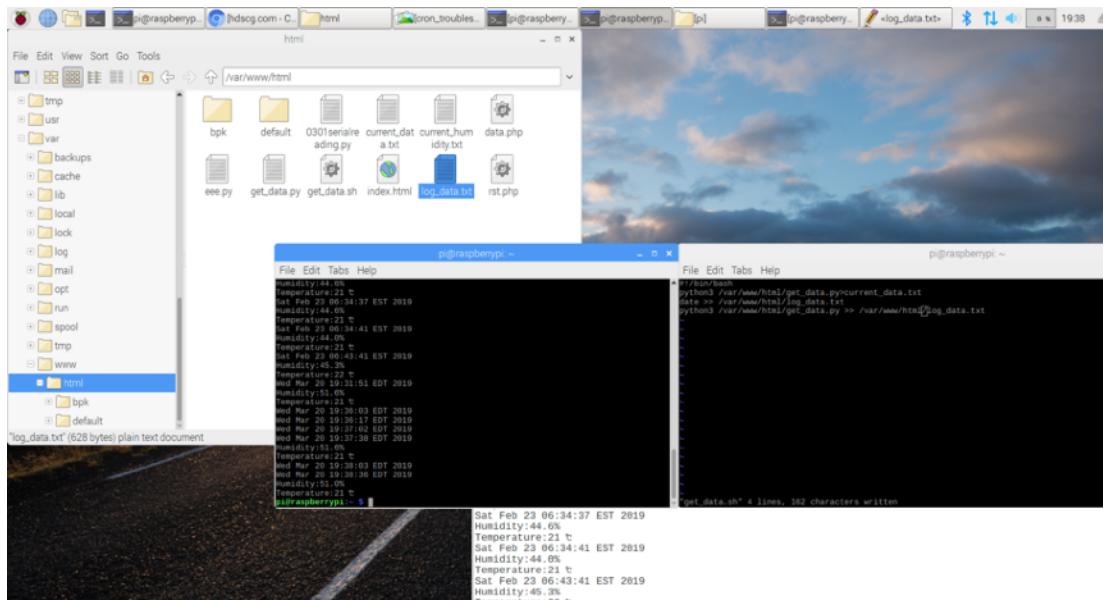
```

pi@raspberrypi:~ $ 
pi@raspberrypi:~ $ ** * * 1 >> /var/www/html/log_data.txt
You have new mail in /var/mail/root
root@raspberrypi:/home/pi# cd /var/www/html/
You have new mail in /var/mail/root
root@raspberrypi:/var/www/html# sh get_data.sh
root@raspberrypi:/var/www/html# cat log_data.txt
Sat Feb 23 06:34:26 EST 2019
Humidity:44.0%
Temperature:21 °C
Sat Feb 23 06:34:38 EST 2019
Humidity:44.0%
Temperature:21 °C
Sat Feb 23 06:34:37 EST 2019
Humidity:44.0%
Temperature:21 °C
Sat Feb 23 06:34:41 EST 2019
Humidity:44.0%
Temperature:21 °C
Sat Feb 23 06:34:43 EST 2019
Humidity:45.0%
Temperature:21 °C
Wed Mar 20 19:31:51 EDT 2019
Humidity:51.0%
Temperature:21 °C
root@raspberrypi:/var/www/html# 

```

(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.