Plans and Progress

Sunday, August 2, 2020 12:37 PM

Client side hardware Phase 1 – Preparing the Raspberry Pi for temperature readings

Install Raspberry Pi OS Lite using Installer App

Start up the Pi with Ethernet plugged in. (And the usb wireless dongle)

After logging in as the default user 'pi' perform the following commands:

```
sudo apt update
sudo apt upgrade
sudu raspberry-config
```

Within the raspi-config application, Enable SSH for remote access and Setup Wifi for network connectivity, then save, remove the Ethernet and reboot

Log in again (optionally run the 'ip addr' command to get the wireless network address confirming connectivity. This can then be used to SSH in from a workstation for further commands)

Install the GPIO pinout utility so that we can determine what to connect the temperature sensor to:

```
sudo apt install python3-gpiozero
```
`pinout` (To display GPIO pins)

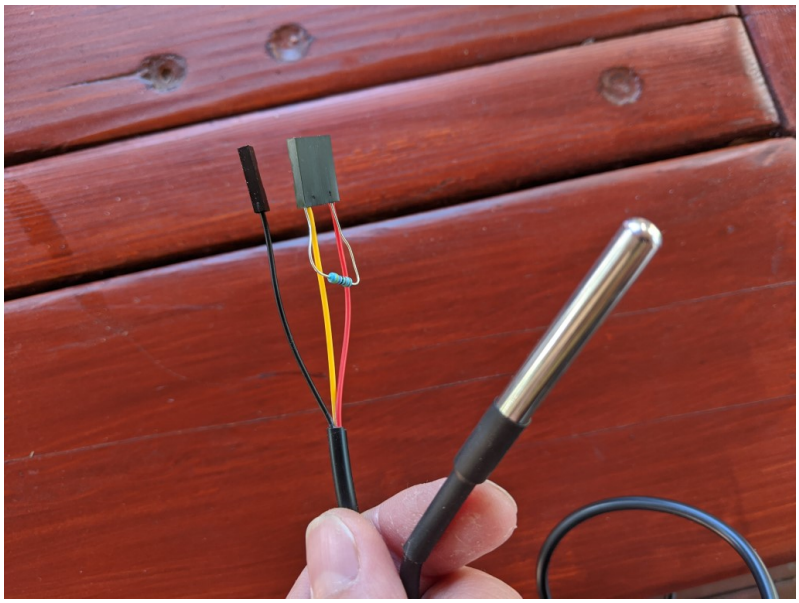
The pinout for the particular model of Pi should be displayed for example:

```
For further information, please refer to https://pinout.xyz/
```

No Shutdown to connect the temperature sensor.

shutdown

The water proof temperature probe we are using is based on a 3 wire DS18B20 Digital temperature sensor. The 3 wires are a ground, +3.3 Voltage in and a data wire. In order to use a DS18B20 with a raspberry pi one a 4.7k resistor is required to bridge the +3.3V and data. For more information on this see https://blog.robertelder.org/ds18b20-raspberry-pi-setup-pullup/



The Red wire goes to Pin 1 (3.3V) The Yellow Wire goes to Pin 7 (GPIO4) and the black wire goes to PIN6 (one of the grounds)

For testing purpose we will be measuring the temperature of a fish tank, with the temperature probe easily removable to verify change in temperature readings.

Once logged in we need to install software modules to communicate with both the GPIO circuit and of course the DS18B20 temperature probe

```
sudo modprobe w1-gpio
sudo modprobe w1-therm
sudo nano /etc/modules
```

-- Add the following lines the file and save with CTRL-X

```
w1-gpio
w1-therm
```

```
sudo nano /boot/config
```

-- Add the following line and save with CTRL-X

```
 dtoverlay=w1-gpio
```

Reboot again and the raspberry pi should now be ready for us to read the temperature.

We should be able to see the device with:

```
pi@tank1:~ $ ls /sys/bus/w1/devices/
28-030297945948  w1_bus_master1
```

The directory starting with 28- is a unique identifer for the probe. If we had multiple temperature probes connected we would have multiple directory references. We can check its raw output with the following:

```
pi@tank1:~ $ cat /sys/bus/w1/devices/28-030297945948/w1_slave
33 01 55 05 7f a5 a5 66 4b : crc=4b YES
33 01 55 05 7f a5 a5 66 4b t=19187
```

As we can see the t=19187 indicates 19 or so degrees celcius

Client side hardware Phase 1 – Writing code to read temperature

We will be using Python to interact with the temperature sensor as research indicates this should be fairly easy and Python can also interact with Round Robin Databases using a rrdtool plugin.

After setting up a git repository for this project we want to clone it on the raspberry pi.

git clone https://github.com/SamuelPaulAshton/2020SP2Group18A3.git

Now setup Gits user details for commits and pushes (If this hasn't previously been done)

```
cd 2020SP2Group18A3
git config --global user.email "s3742249@student.rmit.edu.au"
git config --global user.name "Sam Ashton"
```

Now we will create our Python file:

```
nano temperaturereader.py
```

And enter the following code:

<<<<temperature.py>>>>>

Now test the python file:

```
pi@tank1:~/2020SP2Group18A3 $ python temperatureread.py
18.937
19.0
19.0
19.0
```

Now we have working base, ad this file and commit to the repo

```
pi@tank1:~/2020SP2Group18A3 $ git add temperatureread.py
pi@tank1:~/2020SP2Group18A3 $ git commit -m "First commit, python script to
read and output temperature every 1 second"
[master fd14e34] First commit, python script to read and output temperature
every 1 second
1 file changed, 39 insertions(+)
create mode 100644 temperatureread.py
pi@tank1:~/2020SP2Group18A3 $ git push
Username for 'https://github.com': SamuelPaulAshton
```

```
Password for 'https://SamuelPaulAshton@github.com':
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 883 bytes | 88.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/SamuelPaulAshton/2020SP2Group18A3.git
   383e754..fd14e34  master -> master
```

Client side hardware Phase 1 – Implementing a Round Robin Database to store temperature data

Install rrdtool:
Sudo apt install rrdtool

First we want to create the round robin database:

rrdtool create tempdb.rrd --start now DS:temp:GAUGE:600:-273:5000 RRA:AVERAGE:0.5:1:1200

We can manually add values to this but we will go straight to getting our Python script to do it.

Nano temperatueread.py

Add `import rrdtool` to the top

In our While loop add the following lines to update the tempdb.rrd each time the temperature is read.

```
value="N:" + str(temp_c)
rrdtool.update("tempdb.rrd", value)
```

Every one second is a bit of overkill so additionally, modify the sleep time to 300 seconds (5 minutes) and we can probably remove the `print(temp_c)` line aswell.

```
time.sleep(300)
```

We can now run our python script (as a backround process) and wait 10 or so minuted for a couple of updates

```
pi@tank1:~/2020SP2Group18A3 $ python temperatureread.py &
[1] 24490
```

You can see that the script has a process ID of 24490.

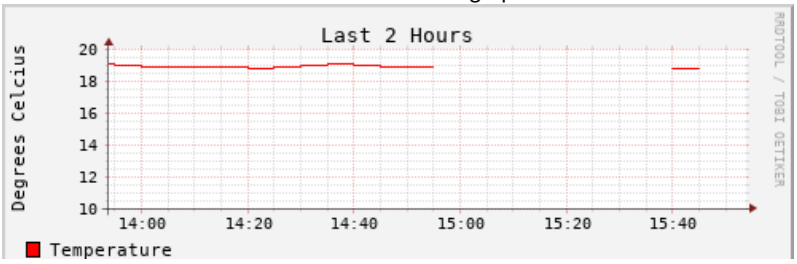
Generating graphs for display can be done from the command line with the following commands:

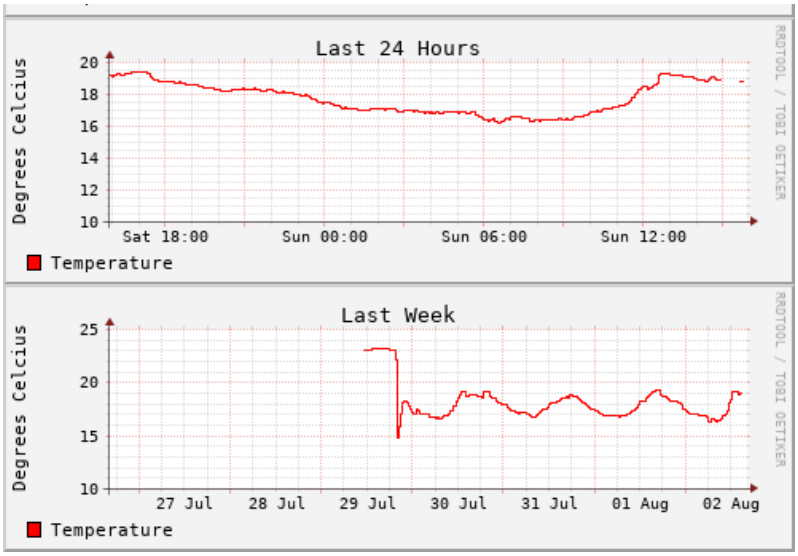
```
rrdtool graph last2hours.png --end now --start end-2h
DEF:Temperature=tempdb.rrd:temp:AVERAGE
LINE1:Temperature#FF0000:"Temperature" -v "Degrees Celcius" -t "Last 2
Hours"

rrdtool graph last24hours.png --end now --start end-86400s
DEF:Temperature=tempdb.rrd:temp:AVERAGE
LINE1:Temperature#FF0000:"Temperature" -v "Degrees Celcius" -t "Last 24
Hours"

rrdtool graph lastweek.png --end now --start end-1w
DEF:Temperature=tempdb.rrd:temp:AVERAGE
LINE1:Temperature#FF0000:"Temperature" -v "Degrees Celcius" -t "Last Week"
```

This 3 commands will each create a PNG with a graph of the data as such:





We should be able to add the recreation of these to our script so we have graphs up to date every 5 minutes. To do this add the following into the while loop in our Python script:

```
rrdtool.graph("last2hours.png", "--start", "-2h",
"DEF:Temperature=tempdb.rrd:temp:AVERAGE","LINE1:Temperature#FF0000:Temperatu
re", "-v","Degrees Celcius", "-t", "Last 2 Hours")
rrdtool.graph("last24hours.png", "--start", "-24h",
"DEF:Temperature=tempdb.rrd:temp:AVERAGE","LINE1:Temperature#FF0000:Temperatu
re", "-v","Degrees Celcius", "-t", "Last 24 Hours")
rdtool.graph("lastweek.png", "--start", "-1w",
"DEF:Temperature=tempdb.rrd:temp:AVERAGE","LINE1:Temperature#FF0000:Temperatu
re","-v","Degrees Celcius", "-t", "Last Week")
```

Rerunning this script now will result in the 3 PNG files being updated every 5 minutes.

Client side hardware Phase 1 A web interface for monitoring the values directly on the client device

First we need a local web server, either Apache or Nginx will do – it doesn't really matter but we will use Nginx for no particular reason

sudo apt install nginx

Now we will link the nginx webroot location to our project folder for easier access to the PNG files

cd /var/www
sudo rm html
sudo ln –s ~/2020SP2Group18A3

Now we create a basic index.html to display the images

```
<!DOCTYPE html>
<html>
<head>
<title>Aquaponics Health Dashboard Prototype</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Aquaponics Dashboard Prototype</h1>
<img src="last2hours.png">
<img src="last24hours.png">
<img src="lastweek.png">
```

```
</body>
</html>
```