CSE 157 - Lab 1

Dallas Meyer

Abstract—This report addresses setting up the RaspberryPi, and its network interfaces (WiFi, Adhoc) and other programming.

Index Terms—RaspberryPi, IoT.

1 Securing and Getting to Know Your Pi

After getting the Raspberry Pi (RPi), there is still a slight set up required before one can program, browse the internet, etc.

1.1 Installing the Operating System

The Raspberry Pi may come with an outdated operating system (as in my case) or none at all, as such its best to flash the RPi's SD card with the newest OS, Raspbian in this case. This can be done by installing the "Raspberry Pi Imager" on a separate computer (https://www.raspberrypi.com/software/) and using that to flash the SD card. After that, simply plug it into the RPi.

Fig. 1. Raspberry Pi Imager program

1.2 Setting up the WiFi

After booting up the Raspberry Pi, you may get automatically prompted to connect to a WiFi network. Connecting to the WiFi can also be done through other methods. Such as through clicking the WiFi icon at the top right of the screen, and selecting one of the listed networks (Fig. 2).

Fig. 2. List of networks (https://raspberrytips.com/raspberry-pi-wifi-setup/)

Another method is through the basic-looking RPi Config tool, which can see accessed through the command: **sudo raspi-config**. This is useful for headless systems as it can be accessed through an SSH terminal connection.

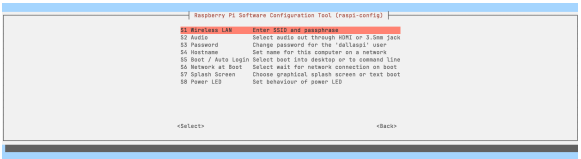

Fig. 3. Raspberry Pi Configuration Tool, showing the WiFi settings)

Another more manual way of setting up the WiFi, is through editing configuration files.

1.3 Changing the password

There are many methods of changing the password for the RPi, which is used for permissions, ssh, and other admin functions.

One way is through the command **passwd**, where self-explanatory input prompts are given.

The other way is to use the previously mentioned Raspberry Pi Configuration Tool.

The system preferences settings also has a password-reset option, which is more user-friendly since it can be easily located.

1.4 Headless mode - SSH

To use the RPi without a monitor & keyboard, one can SSH into it using another device.

To do this:

- ensure that SSH is enabled through the Raspberry Pi Configuration settings.
- Then on the RPi, run the command **ifconfig** to locate the IP address. The address can also be found by hovering the mouse over the wifi connection icon.
- On another computer, simply run the SSH command **ssh [RPi name]@[IP address]**

As for whether I'd find this functionality useful, I'd find myself doing this often. This is because setting up the PI can be a tedious with all the cables, with plugging them in and carrying them around. In addition, I found that terminal IDE/text-editors work really well, which I primarily use.

1.5 Network information and interfaces

The Raspberry Pi contains lots of network interfaces, allowing basic functionality such as WiFi and ad-hoc modes.

Running the command **ifconfig** will list lots of different network interfaces. This also will allow you to check their status, along with any addresses assigned. This is useful when needing to determine the IP addresses.

iwconfig will instead show the current network configurations: essid, mode, channel. This is helpful as it can indicate whether we are in ad-hoc mode and as such are sending out a signal.

In a application that uses WiFi, the RPi usually follows a client-server model, as it connects to a outside network (eg. web browsing). This requires existing infrastructure.

For ad-hoc applications, it allows unique infrastructure, also known for its peer to peer functionality. This is useful for when a WiFi network doesn't exist to allow devices to communicate wirelessly.

1.5.1 Comparing WiFi and Ad-hoc interface files

adhoc-interface file

```
auto lo
iface lo inet loopback
iface eth0 inet dhcp

auto wlan0
iface wlan0 inet static
address 192.168.1.1
netmask 255.255.255.0
wireless-channel 4
wireless-essid RPitest
wireless-mode ad-hoc
```

When comparing the interface files they have similarities and differences.

They both utilize the wlan0 interface. This makes sense as it is responsible for wireless connections, and ethernet (eth0) is not being used.

One notable difference is the modes and wireless settings, where obviously "wireless-mode ad-hoc" is set for the ad-hoc interface. A wireless-channel, and wireless-essid has also been set here since Ad-hoc "creates" a network.

On the other hand, WiFi interface has the network ssid and password set accordingly to the existing nearby WiFi network (eg. "Eduroam").

2 Setting up the Pi for development

When setting up the RPi, I ensured that it was updated through the command **sudo apt-get update** so that it can install the latest libraries.

I also set up a Git repository so that in case anything happens to the RPi, work can be saved. In my case, my RPi stopped booting up completely, requiring me to start this lab over again. Fortunately I haven't started programming in this stage.

The RPi's Rasbpian OS already included lots of things thats needed to start development, as such not much configuration was needed besides the aforementioned network and SSH setup.

2.1 Choosing an IDE

For an IDE, I choose to use NeoVim. This is because NeoVim is lightweight, and can be easily installed and used in the existing terminal. In addition, I'm familiar with NeoVim and enjoy that fact that everything can be done via the keyboard, making it useful during SSH.

In addition its very customizable and convenient. As I have a NeoVim/LazyVim configuration that sets up a fully fledged IDE – through one simple "git clone". It can be configured to include other debugging tools as well.

To install NeoVim, I had to use the RPi's "snap store", as the default Raspbian library had a outdated version of NeoVim. This was a slight headache as the installation took a couple minutes as the installation speeds were really slow.

3 Creating a Weather Report Program

For the "programming" portion of the lab, a python script is created here that is a "weather report" program.

In summary, the script takes UCSC's latitude and longitude coordinates and formats that into a Open Meteo website link. From this link is a list of max temperatures – acquired through a GET request. These contents are then pasted into a output text file. This text file also will have my name, and current time and date as a header.

I will go into further detail on this program.

3.1 Code

```
# Generates a report for the weather
import time
import requests
from datetime import datetime

time.sleep(20)
url =
    "https://api.open-meteo.com/v1/forecast
?latitude=37&longitude=-122.06&
daily=temperature_2m_max&timezone=America
%2FLos_Angeles"

x = requests.get(url)

name = "Dallas Meyer\n"

now = datetime.now()
time = now.strftime("%I:%M %p\n")
date = now.strftime("%m-%d-%Y\n\n")


payload = str(x.content)[2:-2]
with open("report.txt", "w") as file:
    file.write(name)
    file.write(time)
    file.write(date)
    file.write(payload)
```

3.2 Description

At the start of the code, I imported the required libraries for the time and GET request functions.

The sleep function is used here. So that when running the program at boot it gives time for the WiFi to be fully set up.

Many variables are set, such as the URL, name, time, and date. These will be written into the output file.

The api.open-meteo.com URL has the longitude and latitude inputted, and asks for the recent maximum 2m daily temperatures at that location. This makes gathering info relatively simple.

"x" is a special object (requests.Response object) that receives the file from the URL GET request.

The "payload" variable simply trims/formats and stores the content from the URL.

The output file "report.txt" is opened and written into using the variables we had set earlier, almost as if line-by-line.

3.3 Example output file

```
Dallas Meyer
01:22 AM
10-08-2023

{"latitude":36.997498,
"longitude":-122.076935,
"generationtime_ms":0.029087066650390625,
"utc_offset_seconds":-25200,
"timezone":"America/Los_Angeles",
"timezone_abbreviation
":"PDT","elevation":263.0,"daily_units":
{"time":"iso8601",
"temperature_2m_max":"\xc2\xb0C"},
"daily":{"time":["2023-10-08",
"2023-10-09","2023-10-10","2023-10-11",
"2023-10-12","2023-10-13","2023-10■14"],
"temperature_2m_max":
[21.7,16.0,20.2,20.4,23.8,22.6,23.4]}
```

Example output after running the generate_report.py program.

3.4 More details on the Request API and HTTP

When sending a request and receiving from a server, this is considered a client-server model. HTTP serves a protocol that allows servers and clients to communicate with each other. HTTP has many different methods, such as a POST which sends data to a server.

When ever you access a website, your web browser sends a HTTP (Hypertext Transfer Protocol) GET request to the server order to retrieve the contents/file of the webpage. The server/website then sees this requests and returns that result. This connection is based on the TCP connection, usually using the default server port 80.

Python has a built-in library called "requests" used to implement HTTP. As described earlier in the code, "requests.get(url)" sends a GET request to the specified URL and stores that into a object. The sent request may look like Fig. 4 below

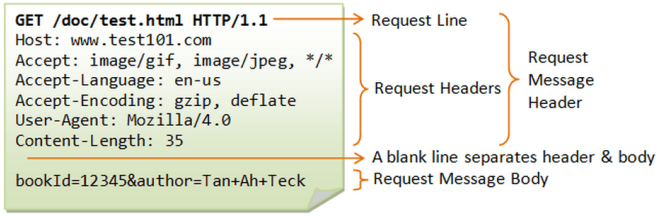

Fig. 4. HTTP GET request example (lifewire.com)

4 Booting Programs in the Start Sequence

4.1 Running the Weather Report program on Boot via Systemd

When trying to have the program run on boot, I settled on the systemd method. Since the program requires internet (to send the GET request), systemd allows you to run the program after the network interfaces have been started/setup.

I didn't have much luck using other methods such as init.d, as I found that it doesn't guarantee a network connection.

```
[Unit]
Description= Weather Report
After=network-online.target
Wants=network.online.target

[Service]
ExecStart=/usr/bin/python3
    /home/dallaspi/Desktop/cse-157
/lab1/generate_report.py

WorkingDirectory=/home/dallaspi/Desktop/
cse-157/lab1

User=dallaspi

[Install]
WantedBy=multi-user.target
```

lab1script.service file

In the [Unit] section, the "After" & "Wants" specifies to run the script after the network is online.

The [Service] section specifies what to execute. It executes the python file, along with where the working directory is.

The [Install] section sets parameters related to installation. The "wantedby" specifies the

runlevels. I found that this was included in many systemd guides.

4.1.1 Including Programs in Boot Sequence

When including a program in the boot sequence, one useful feature is that it allows you to run files or programs without needing to set up a keyboard, monitor, ssh connection, etc.

This behaves similarly to an embedded systems where a program is installed on it, and does just one job without needing any necessary inputs from the user. Thus the Raspberry Pi can be used for many other IoT applications and create smart devices (eg. temperature sensors).

The disadvantage with this is that it does increase the boot loading times, and does require a bit of tinkering to set up. In addition if a infinite loop is implemented by accident, it could make it very difficult to get the RPi to boot correctly.

5 Conclusion

Overall, the Raspberry Pi does require a bit of setting up. Although its versatile as many things (eg. network) can be set up through the command-line or GUI. In addition, it can be used as a stand-alone computer, or as a IoT device that you can SSH into.

For network connections, the Raspberry Pi can act as a client and send connections/requests to another server. Likewise it can act as the server or work in a peer to peer network (ad-hoc) without any other existing infrastructure.

The raspberry pi's linux-based operating system, also allows it to install and run many different applications and programs. This makes it very useful when wanting to program on it as one can install an IDE.

Having linux also makes it easier to set up programs during boot, as there are many different methods to use (eg. bash.rc, unit.d systemd).

It is indeed a nifty little machine that I have used many times in the past, such as for a motion-sensing security camera.

As for the python programming, its pre-included libraries makes it easy to implement a program. Such as one that gathers information from a website, or communicate through HTTP or perhaps any other TCP protocols.