Web-Based GPIO Control

A. Objectives

1. Understand about web.py
2. Learn about the packets passing between a web server and a client
3. Web-based GPIO control using Web.py
4. Review Python programming
5. Introduction to JavaScrip
6.

B. Time of Completion

This laboratory activity is designed for students with some understanding about Raspberry Pi and it is estimated to take about 4 hours to complete. You must have done Labs 1-3 before starting this lab.

C. Requirements

1. A Raspberry Pi 3 Model 3
2. 32 GByte MicroSD card → Give your MicroSD card to the lab instructor for a copy of Ubuntu.
3. USB adaptor to power up the Pi
4. LED and wires
5. Wireshark

D. Pre-Lab

Lear about Web.py. Have a general understanding about HTML and JavaScript. Make sure you review your previous Python programs.

E. Lab

Let's learn a little about Web.py [1]. Web.py is a web framework for Python. In our case we like to use Web.py as a way to run our web server. So basically, instead of starting, say apache2 service directly (as we learned in the previous lab) we start the Web.py service, which happens to be a Python-based program. Web.py is simple, powerful and it is in the public domain; you can use it for whatever purpose with absolutely no restrictions. When we invoke Web.py we can basically allow access to any particular port. For example, we can allow the user to access a web page by typing 192.168.1.75:8080. In this case port 8080 is open and is listening. Therefore, once the user access this address using the browser, the user can view the page. The best way to get this is to actually do an example.

1. Installing Web.py Service

First, using SSH log into your RPI. In your home directory on your RPi create a directory called my_webpi. Make sure you have a directory called `/home/ssuee/my_webpi`. Create a file called `example_code.py` and paste the following code in there. This file effectively acts as your web page (or htm) file! See Appendix A for full description of the code.

Make sure your web server is **not** active: `systemctl stop apache2`.

Now, let's enable the web.py service. First do `ps` and make sure there is no python process. See the figure below. If there is a python process stop the process using `kill -9` command.

Start web.py service by entering codepi.py form the `/home/ssuee/my_webpi` directory: `python example_code.py`

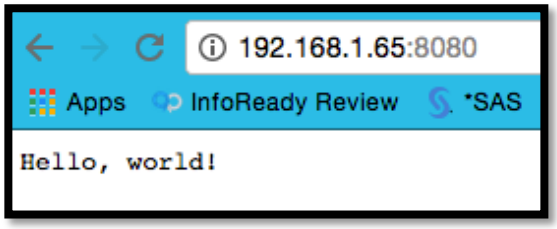
Note that you can use `python example_code.py &` in order to run the server on the background. Remember the process number or just use `ps` command to find the process number.

Upon enabling the web.py, you should see something like http://0.0.0.0:/8080. This is the port that is listening. In your browser point to the address above and your should see something like this:

Note that the IP address is referring to is the address of your RPI.

```
import web  #importing web.py module

#set URL structure – index is the html file we open
(index.html in the /templates directory)
urls = (
    '/', 'index'
)

# Upon requesting the page we GET the following
message:
class index:
    def GET(self):
        return "Hello, world!"

#Create an application specifying the url and a way
to tell web.py to start serving web pages:
if __name__ == "__main__":
    app = web.application(urls, globals())
    app.run()
```

```
ssuee@ssuee-desktop:~/my_webpi$ ps
  PID TTY          TIME CMD
 4458 pts/0    00:00:01 bash
 9811 pts/0    00:00:00 ps
ssuee@ssuee-desktop:~/my_webpi$ python ex_code.py
http://0.0.0.0:8080/
192.168.1.72:50760 - - [11/Nov/2016 13:02:24] "HTTP/1.1 GET /" - 200 OK
```



[1] For more information please see http://webpy.org/docs/0.3/tutorial

Answer the following questions:

1- How can you change the code so the page displays your name?
2- Using the returned response, which port is the client using to access the web.py on the RPi? What is the IP address of the client?
3- While your Web.py is enabled, on your RPi run the following command sudo nmap -sT -O 192.168.1.xx. You can do this from any other machine that supports nmap. Note that xx refers to the IP address of the RPi. Alternatively, you may want to open a new terminal on your PC and ssh into the RPi. Then use the second terminal and run the nmap command. Using the retuned result, verify that port 8080 is open or not. Explain your response.
4- Stop your Web.py and use the following command to restart it: `python example_code.py 1234`. Use nmap command to verify that port 1234 is in fact open. What is it called?
5- Assuming web.py is using port 1234, how can you access your web page? What exactly is the url you used?
6- Open your Wireshark on your client side (PC) and follow these steps:
    a. Capture the packets as you try to access the RPi using your browser.
    b. Search fro HTTP protocols in your captured packets.
    c. Click on one and then click on Analys→Follow→TCP Stream. Which port is being accessed on the RPi? Which TCP stream is this?
    d. Note that in the case below I got TCP stream 8!
    e. Go to Statistics→Flow Graph. You should get something similar to the figure below. Note that you can ignore the segments reassembled PDU (Protocol Data Unit) [2].
    f. Looking at your packets, when the client receives the response, does the client send an ACK to the server to confirm that the message was received? How do you know?

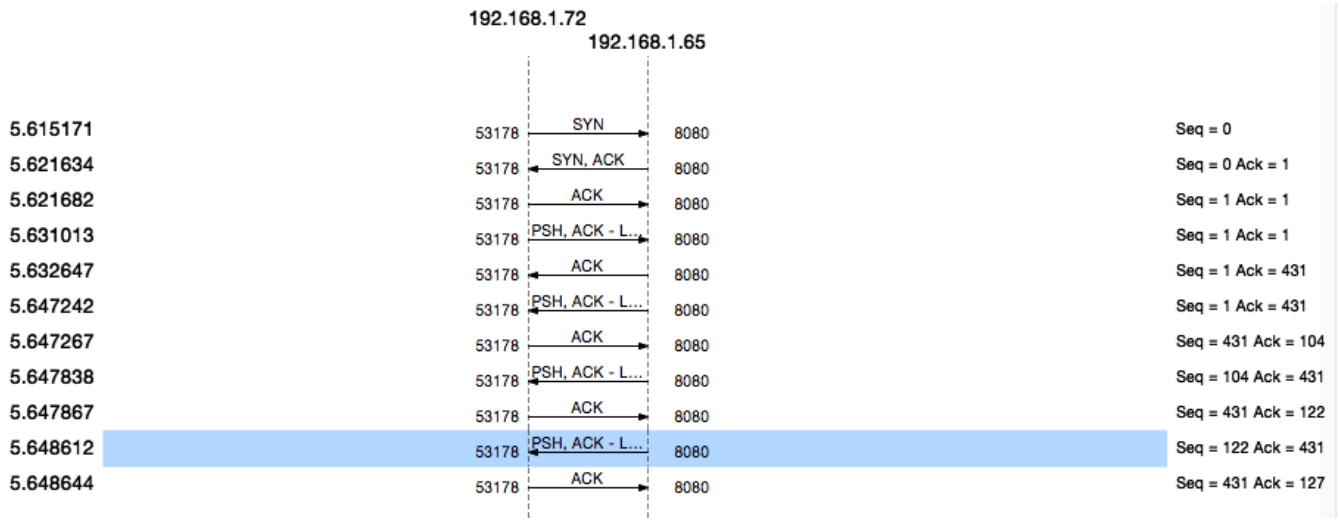
[2] Read for more information: https://www.wireshark.org/lists/wireshark-users/200805/msg00206.html

Note that Frame 52 @ Time 5.6486 is highlighted in both figures.

tcp.stream eq 8 Expression... +

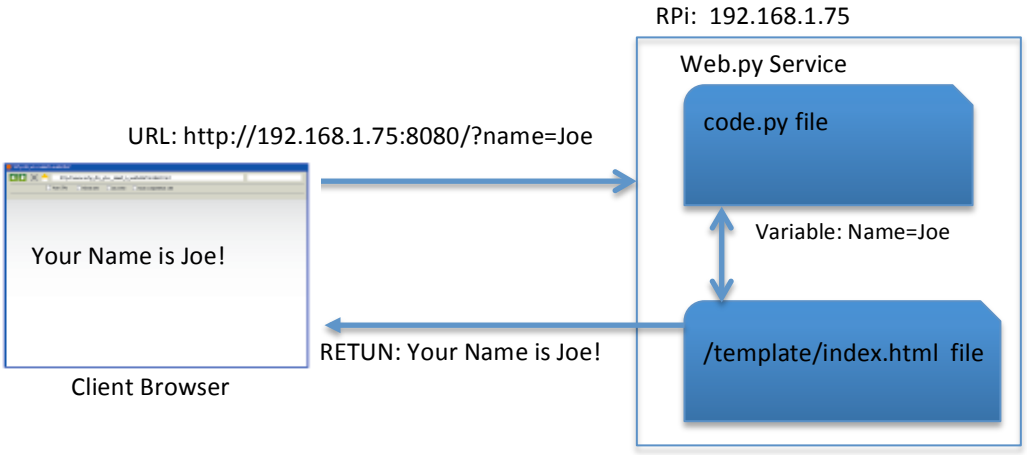
No.	Time	Source	Destination	Protocol	Length	Info
34	5.615171	192.168.1.72	192.168.1.65	TCP	78	53178 → 8080 [SYN] Seq=0 Win=65535 L…
38	5.621634	192.168.1.65	192.168.1.72	TCP	74	8080 → 53178 [SYN, ACK] Seq=0 Ack=1 …
39	5.621682	192.168.1.72	192.168.1.65	TCP	66	53178 → 8080 [ACK] Seq=1 Ack=1 Win=1…
42	5.631013	192.168.1.72	192.168.1.65	HTTP	496	GET / HTTP/1.1
43	5.632647	192.168.1.65	192.168.1.72	TCP	66	8080 → 53178 [ACK] Seq=1 Ack=431 Win…
48	5.647242	192.168.1.65	192.168.1.72	TCP	169	[TCP segment of a reassembled PDU]
49	5.647267	192.168.1.72	192.168.1.65	TCP	66	53178 → 8080 [ACK] Seq=431 Ack=104 W…
50	5.647838	192.168.1.65	192.168.1.72	TCP	84	[TCP segment of a reassembled PDU]
51	5.647867	192.168.1.72	192.168.1.65	TCP	66	53178 → 8080 [ACK] Seq=431 Ack=122 W…
52	5.648612	192.168.1.65	192.168.1.72	HTTP	71	HTTP/1.1 200 OK
53	5.648644	192.168.1.72	192.168.1.65	TCP	66	53178 → 8080 [ACK] Seq=431 Ack=127 W…

```
GET / HTTP/1.1
Host: 192.168.1.65:8080
Connection: keep-alive
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_5) AppleWebKit/
537.36 (KHTML, like Gecko) Chrome/54.0.2840.71 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/
*;q=0.8
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8,es;q=0.6,fr;q=0.4

HTTP/1.1 200 OK
Transfer-Encoding: chunked
Date: Fri, 11 Nov 2016 22:04:51 GMT
Server: localhost

d
Hello, world!
0
```



2. Interacting Web.py Service with HTML pages

We now want to use the web.py such that when the service is enabled, a particular web page is called. This way we can write our web page using HTML code as opposed to Python. Thus, when the client accesses the web.py service, the client is directed to an HTML file. For this section we use files in Appendix B.



So, let's start. In your RPI's home directory create a directory called my_webpi. Under my_webpi create a sub-directory called templates. Using the files in Appendix B create codepi.py and index.html place them in the correct directory as shown here:

```
ssuee@ssuee-desktop:~/my_webpi$ tree
.
├── codepi.py
├── codepi.pyc
└── templates
    └── index.html

1 directory, 3 files
```

An easy way to copy and paste the files is using nano text editor. This text editor allows you to copy and paste while your are accessing the RPi through SSH. Nano is probably much easier than vi text editor. Use `nano index.html` for example.

Once you have all the files in the correct directories make sure you web server is not active: `systemctl stop apache2`

Using ***pip*** utility install web.py as shown below:

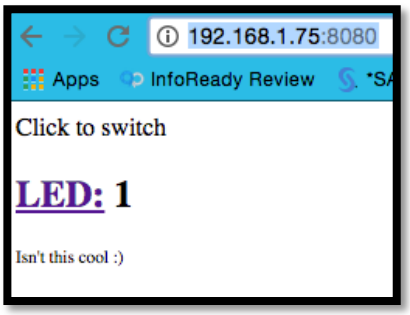
```
ssuee@ssuee-desktop:~/my_webpi$ pip install web.py
Collecting web.py
  Downloading web.py-0.38.tar.gz (91kB)
    100% |████████████████████████████████| 92kB 447kB/s
Building wheels for collected packages: web.py
  Running setup.py bdist_wheel for web.py ... done
  Stored in directory: /home/ssuee/.cache/pip/wheels/6d/80/5f/15b1b743a43cbed7f8
bbd9d4833e8530af9cf7209fc246fb07
Successfully built web.py
Installing collected packages: web.py
Successfully installed web.py-0.38
```

Note that **pip** is yet another free and powerful utility similar to the **apt-get** utility that offers package management command line ability. Similar to the **apt-get** utility it works with **Ubuntu's APT** (Advanced Packaging Tool) library to perform installation of new software packages, removing existing software packages, etc.

Start web.py service by entering codepi.py form the `/home/ssuee/my_webpi` directory: `python codepi.py`

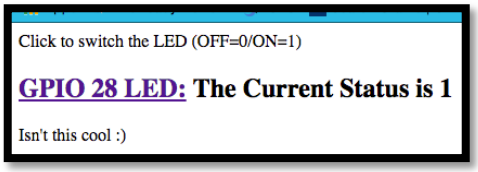
Note that you can use `python codepi.py &` in order to run the server on the background. Remember the process number or just use **ps** command to find the process number.

In the URL of your web browser type http://192.168.1.75:8080/ and you should see something like the figure below. Note that we assume 192.168.1.75 is the IP address of your RPI.



Click on LED and note that the LED status shown on your browser changes. See what happens to the URL every time you click on LED in the browser.

Change the `index.html` file so the web site displays the following:



Physically connect an LED to GPIO channel number 28 (as indicated in `codepi.py` file). Make sure it toggles as you click on the text on your web page.

In HTML coding, it is not possible to add logic (e.g., IF statement). Therefore, we typically use JavaScript coding to add logic to our HTML code. Add the following JavaScript code to the end of your `index.html` file (before `</body>`). Note that in this case we are passing the value of `$status[0]` to the IF statement so we can change the displayed text! JavaScript is a powerful way to create dynamic HTML codes!

```
<script type="text/javascript">
var d = $status[0]
if (d < 1)
{
  document.write("<b>The LED is OFF</b>")
}
else
{
  document.write("<b>The LED is ON</b>")
}
</script>
```

Answer the following questions:

1- Doe the status value really shows the actual status of the LED?

2- In your `index.html` file what happens if you change the values in <style> section?

3- Which line in `codepi.py` passes the control to `index.html` file?

PROGRAMING EXERCISE: Modify `codepi.py` and `index.html` files such that the user can control TWO LEDs on your PI remotely. Show that LEDs can be controlled. Be prepared to demonstrate this in the lab. Make sure you have everything you need!

F. Submissions

- Complete the programming exercise.
- Answer all the questions in Section E.1
- Answer all the questions in Section E.2

F. Credits

Special thanks to online resources and all SSU students who assisted putting together this lab.

E. References

- Web control using Web.py: http://www.paulschow.com/2013/06/raspberry-pi-internet-led-control.html
- A vey simple example using web.py to control: http://blog.vikramank.com/2014/08/controlling-raspberry-pi-mobile/
- Web control using PHP and Python: http://www.reuk.co.uk/wordpress/raspberry-pi/simple-raspberry-pi-relay-control-over-the-internet/
- Web server control and Arduino: http://startingelectronics.org/tutorials/arduino/ethernet-shield-web-server-tutorial/CSS-introduction/
- Learn about HTML code: http://www.w3schools.com/html/html_comments.asp
- Web.py tutorial: http://webpy.org/
- Examples of Javascript: http://stackoverflow.com/questions/17439921/if-then-statement-in-html5

Appendix A

```
#!/usr/bin/env python
# first line points to path for python

# This is a web.py application example
# When we run code.py the web.py service begins: In ter terminal type the following:
# python code.py 1234
# To see the response open a browser and in its URL type the following:
# http://localhost:1234
# Depending on the code in class index we get different responses.

import web  #importing web.py module

#This tells web.py to look for templates in your templates directory. Then change
render = web.template.render('templates/')

#set URL structure – index is the html file we open (index.html in the /templates directory)
urls = (
    '/', 'index'
)

# Upon requesting the page we GET the following message:
class index:
    def GET(self):
        return "Hello, world!"

#Create an application specifying the url and a way to tell web.py to start serving web
pages:
if __name__ == "__main__":
    app = web.application(urls, globals())
    app.run()
```

```
#!/usr/bin/env python
# first line points to path for python

# we import the web.py library
import web

import RPi.GPIO as GPIO

#dont bug me with warnings
GPIO.setwarnings(False)

# to use Raspberry Pi bcm numbers
GPIO.setmode(GPIO.BCM)

# set up GPIO output channels
GPIO.setup(28, GPIO.OUT) #LED 1

#make a status global variable
global status

#fill it with zeroes
status = [0]

#templates are in templates folder
render = web.template.render('templates/')

urls = ('/','index')
app = web.application(urls,globals())

class index:

 def __init__(self):
  self.hello = "snakes on a pie!"
 def GET(self):
  getInput = web.input(turn="")
  command = str(getInput.turn)

#control commands
  if command == "8":
   if status[0] == 0:
    #toggle LED 1
    status[0] = 1
    GPIO.output(28, GPIO.HIGH)
    return render.index(status)
   elif status[0] == 1:
    status[0] = 0
    GPIO.output(28, GPIO.LOW)
    return render.index(status)
   else:
   return render.index(status)


   #default
  else:
   #has to start by visiting /?turn=on
   return render.index(status)

if __name__ == "__main__":
        app.run()
```

codepi.py: Toggling PIN 28

```
$def with (status)
<!DOCTYPE html>
<html>
<body>

<style>
 h1 {font-size:150%;}
 h2 {font-size:150%;}
 p  {font-size:100%;}
 s1 {font-size:100%;}
</style>

Click to switch
<h2><a
href="http://192.168.1.75:8080/?turn=8">
LED:</a> $status[0]</h2>

<s1> <p>Isn't this cool :)</p></s1>
</body>
</html>
```

Index.html: Assumes IP address of the PI is 192.168.1.75