



RC 19/20	LAB ASSIGNMENT	Guide.Part #:	1.3
Application Layer		Issue Date:	07 Oct 2019
A Web Server and Web Client in Python		Due Date:	15 Oct 2019
Author: Prof. Rui Cruz		Revision:	4.0

1 Introduction

The objective of this Lab experiment is to use the multi-node environment (servers, network) created in **Part 1** of the Lab Assignment, to learn the basics of network socket programming for TCP connections, using Python.

Preliminary Notes

The instructions in this document are applicable to Computers at the IST Labs.

Nevertheless, one nice feature of the software stack we are going to use is that it is portable to many platforms including **YOUR OWN** personal computers, running the following Operating Systems:

- Microsoft Windows from version 10 up
- Apple macOS from versions 10.13 'High Sierra' up
- Debian-based Linux, such as Ubuntu (recommended) from versions 12.04 'Precise' up.

It is not recommended to apply this setup to a virtual machine (nested virtualization), although possible, as the configuration requires access to a hypervisor environment (recommended Virtualbox) in the host system.

Note: Avoid copying text strings from the command line examples or configurations in this document, as pasting them into your system or files may introduce/modify some characters, leading to errors or inadequate results.

2 The Web Server and Web Client in Python

In this lab, you will learn about network socket programming for TCP connections in Python: how to create a **socket**, bind it to a specific address and port, as well as send and receive HTTP packets. You will also learn some basics of HTTP header format.

Socket is the name given to the **abstraction of an interface used to communicate** between processes. In the context of computer networks, we communicate with processes in different machines (virtual and/or physical). The structures that implement this abstraction are usually managed by the operating system and are manipulated by programmers using an Application Program Interface (API).

For our purpose, you will develop a simple Web server and a web Client in Python. You have available information about the Python's socket interface in <https://docs.python.org/2/library/socket.html> for Python 2 or <https://docs.python.org/3.7/library/socket.html> for Python 3.

RC 19/20	LAB ASSIGNMENT	Guide.Part #:	1.3
Application Layer		Issue Date:	07 Oct 2019
A Web Server and Web Client in Python		Due Date:	15 Oct 2019
Author: Prof. Rui Cruz		Revision:	4.0

2.1 The Web Server

The Web Server to develop is capable of processing only one request at a time. Your web server should accept and parse the HTTP request, get the requested file from the server's file system, create an HTTP response message consisting of the requested file preceded by header lines, and then send the response directly to the client. If the requested file is not present in the server, the server should send an HTTP **"404 Not Found"** message back to the Web Client.

2.2 The Web Client

The HTTP Web Client to develop can be used to test your Web Server. Your client will connect to the server using a TCP connection, send an HTTP request to the server, and display the server response as an output. You can assume that the HTTP request sent is a **GET** method. The client should take *command line arguments* specifying the server IP address or host name, the port at which the server is listening, and the path at which the requested object is stored at the server.

3 Setting up the Experimental Environment

To start, go to the project directory created **Part 1** of this Lab Assignment, named, for example, **weblab**.

Download from the course website the file `RC_19_20_LAB_01_P3_support_files.zip` and uncompress the content to the **weblab** project folder (answer Yes if the system asks to replace existing files).

The **weblab** project folder should then have a structure similar to the following:

```
.
|--- Vagrantfile
|--- bootstrap-client.sh
|--- client
|    |--- webclient.py
|--- html
|    |--- index.html
|    |--- webserver.py
```

A file named `webserver.py` will be created in the "html" folder. That folder should also contain the "index.html" file from previous lab. A file named `webclient.py` will be created in the "client" folder.

In that **weblab** folder verify that the `Vagrantfile` contains a structure similar to the following (adapt addresses, if needed, to suit the environment in your host system):

RC 19/20	LAB ASSIGNMENT	Guide.Part #:	1.3
Application Layer		Issue Date:	07 Oct 2019
A Web Server and Web Client in Python		Due Date:	15 Oct 2019
Author: Prof. Rui Cruz		Revision:	4.0

```
# -*- mode: ruby -*-
# vi: set ft=ruby :
Vagrant.configure(2) do |config|

  config.vm.define "webserver" do |webserver_config|
    webserver_config.vm.box = "ubuntu/trusty64"
    webserver_config.vm.hostname = "webserver"
    webserver_config.vm.network "private_network", ip: "
      192.168.56.21"
    webserver_config.vm.network "forwarded_port", guest: 80, host:
      8080
    if Vagrant::Util::Platform.windows? then
      webserver_config.vm.synced_folder "html", "/home/vagrant/
        html",
        id: "vagrant-root", owner: "vagrant", group: "vagrant",
        mount_options: ["dmode=775,fmode=664"]
    else
      webserver_config.vm.synced_folder "html", "/home/vagrant/
        html"
    end
    webserver_config.vm.provider "virtualbox" do |vb|
      vb.name = "webserver"
      .....
    end # of vb
  end # of webserver_config

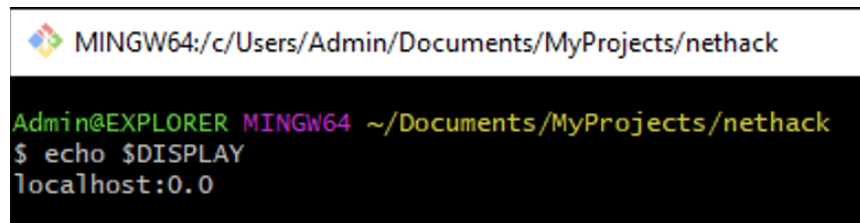
  config.vm.define "client" do |client_config|
    client_config.vm.box = "ubuntu/trusty64"
    client_config.vm.hostname = "client"
    client_config.vm.network "private_network", ip: "192.168.56.51"
    "
    client_config.vm.synced_folder "client", "/home/vagrant/client
      "
    client_config.vm.provider "virtualbox" do |vb|
      # Change the VM name/ID in the Hypervisor
      vb.name = "client"
      .....
    end # of vb
    client_config.vm.provision "shell", path: "bootstrap_client.sh
      "
  end # of client_config
end # of config
```

For Microsoft Windows hosts, you need to start “**VcXsrv**” before spin off the lab environment. The “**VcXsrv**” application will start a **X-Windows** server that will forward

RC 19/20	LAB ASSIGNMENT	Guide.Part #:	1.3
Application Layer		Issue Date:	07 Oct 2019
A Web Server and Web Client in Python		Due Date:	15 Oct 2019
Author: Prof. Rui Cruz		Revision:	4.0

application GUIs running inside the VM to the host system.

Verify also that you have an environment Variable with name **DISPLAY** and with **Variable value** of localhost:0.0, as illustrated in Figure 1.



```

MINGW64:/c/Users/Admin/Documents/MyProjects/nethack
Admin@EXPLORER MINGW64 ~/Documents/MyProjects/nethack
$ echo $DISPLAY
localhost:0.0

```

Figure 1: Verify DISPLAY Variable

For the experiments in this lab you will use **Wireshark** (A network Sniffer and Protocol dissector tool, provisioned and configured in the VM) to capture the packets in the network interface in order for you to analyse them. For that purpose, you will need to have **three Terminal windows opened**. On the first window you will start Wireshark with the following command:

```
$ vagrant ssh nethack -c wireshark-gtk -- -X
```

In other Terminal windows start SSH connection with both machines:

```
$ vagrant ssh client
```

```
$ vagrant ssh webserver
```

In order to have a shared folder you need to ensure in your Vagrantfile that you have lines similar to the following, where the first path is related to the **host** and the second path refers to the **guest**.

```
config.vm.synced_folder "html", "/home/vagrant/html"
```

Having this type of share for both VMs, you ensure that whatever you put on the “html” or “client” folders in your **host** will appear and be available inside the VMs in the “user” **home** folder as you can verify:

```

vagrant@client:~$ cd client
vagrant@client:~/client$ ll client
total xx
-rw----- 1 vagrant vagrant 2202 Oct 16 23:27 webclient.py

```

RC 19/20	LAB ASSIGNMENT	Guide.Part #:	1.3
Application Layer		Issue Date:	07 Oct 2019
A Web Server and Web Client in Python		Due Date:	15 Oct 2019
Author: Prof. Rui Cruz		Revision:	4.0

```

vagrant@webserver:~$ cd html
vagrant@webserver:~/html$ ll
total xx
-rw-r--r-- 1 vagrant vagrant 420 Oct 26 2016 index.html
-rw----- 1 vagrant vagrant 2202 Oct 16 23:27 webserver.py

```

4 Developing a Web Server in Python

Specifically, your Web Server will create a connection socket when contacted by a Web Client that will receive the HTTP request from this connection, will parse the request to determine the specific file being requested, will get the requested file from the server's file system, will create an HTTP response message consisting of the requested file preceded by header lines, and will send the response over the TCP connection to the requesting client.

If a client requests a file that is not present in your server, your server should return a **“404 Not Found”** error message and an error page should be sent.

The skeleton code for your server is as follows (and should be already there in the “html” folder. Your job is to complete the code, run your server, and then test your server by sending requests from other hosts.

The places where you need to fill in code are marked with **Fill in start** and **Fill in end**. Each place may require one or more lines of code.

```

1 # Import socket module
2 from socket import *
3 import sys # In order to terminate the program
4
5 # Create a TCP server socket
6  #(AF_INET is used for IPv4 protocols)
7  #(SOCK_STREAM is used for TCP)
8
9 serverSocket = socket(AF_INET, SOCK_STREAM)
10
11 # Assign a port number
12 serverPort = 6789
13
14 # Bind the socket to server address and server port
15 #Fill in start
16
17 #Fill in end
18
19 # Listen to at most 1 connection at a time

```

RC 19/20	LAB ASSIGNMENT	Guide.Part #:	1.3
Application Layer		Issue Date:	07 Oct 2019
A Web Server and Web Client in Python		Due Date:	15 Oct 2019
Author: Prof. Rui Cruz		Revision:	4.0

```

20 #Fill in start
21
22 #Fill in end
23
24 # Server should be up and running and listening
25 # to the incoming connections
26
27 while True:
28     print('The server is ready to receive')
29
30     # Set up a new connection from the client
31     #Fill in start
32
33     #Fill in end
34
35     # If an exception occurs during the execution of try clause
36     # the rest of the clause is skipped
37     # If the exception type matches the word after except
38     # the except clause is executed
39     try:
40         # Receives the request message from the client
41         message = #Fill in start #Fill in end
42         # Extract path of requested object from message
43         # The Path is the second part of HTTP header,
44         # identif. by [1]
45         filename = message.split()[1]
46         # The extracted path of the HTTP request includes
47         # a character '\', we read the path from
48         # the second character
49         f = open(filename[1:])
50         # Store the entire content of the requested file
51         # in a temporary buffer
52         outputdata = f.read()
53         # Send the HTTP response header line
54         # to the connection socket
55         #Fill in start
56
57         #Fill in end
58
59         # Send the content of the requested file
60         # to the connection socket
61         for i in range(0, len(outputdata)):
62             connectionSocket.send(outputdata[i].encode())
63         connectionSocket.send("\r\n".encode())
64

```

RC 19/20	LAB ASSIGNMENT	Guide.Part #:	1.3
Application Layer		Issue Date:	07 Oct 2019
A Web Server and Web Client in Python		Due Date:	15 Oct 2019
Author: Prof. Rui Cruz		Revision:	4.0

```

65         # Close the client connection socket
66         connectionSocket.close()
67
68     except IOError:
69         # Send HTTP response message for file not found
70         connectionSocket.send("HTTP/1.1 404 Not Found\r\n\r\n"
71                                n".encode())
72         #Fill in start
73         #Fill in end
74         # Close the client connection socket
75         #Fill in start
76         #Fill in end
77
78
79 serverSocket.close()
80 sys.exit() #Terminate the program after sending the corresponding
           data

```

When you will have the complete code, open a Terminal (with Git-Bash) for each system (webserver and client). Open also a wireshark window from the Client machine and select the interface to capture.

In the webserver machine, go inside the shared folder and run your server with:

```

vagrant@webserver:~/html$ python webserver.py
The server is ready to receive

```

Now in the command line for your Client machine run a **telnet** connection to your Web Server and test it by requesting a file, using HTTP. The IP addresses and Port number may be different. Adapt the command to your case.

```

:~$ telnet 192.168.56.21 6789
Trying 192.168.56.21 ...
Connected to 192.168.56.21.
Escape character is '^]'.
GET /index.html HTTP/1.1

```

Analyse the Wireshark capture and try to identify the messages exchanged between the Client and the Web Server.

5 Developing a Web Client in Python

Inspired by the code for the Web Server, develop the code for the simple Web Client.

RC 19/20	LAB ASSIGNMENT	Guide.Part #:	1.3
Application Layer		Issue Date:	07 Oct 2019
A Web Server and Web Client in Python		Due Date:	15 Oct 2019
Author: Prof. Rui Cruz		Revision:	4.0

A skeleton file name “webclient.py” is already inside the “client” folder. The places where you need to fill in code are marked with **Fill in start** and **Fill in end**. Each place may require one or more lines of code.

To start the Web Client go inside that shared folder and run the client (like the following example, using the IP address and Port of the Web server):

```
:~$~/client python webclient.py 192.168.56.21 6789 /index.html
```

The syntax of the command is `client.py server_host server_port filename`.

Analyse the Wireshark capture and try to identify the messages exchanged between the Web Client and the Web Server.

6 Finishing your Experiments

In order to stop the Virtual Machines and to verify the global state of all active Vagrant environments on the system, we can issue the following commands:

```
:~$ vagrant halt
==> webserver: Attempting graceful shutdown of VM...
==> client: Attempting graceful shutdown of VM...
:~$ vagrant global-status
```

Confirm that the statuses of the VMs is 'powered off'. In order to prevent those instantiated machines to use resources, you can destroy them, as they can now be recreated with that simple command `vagrant up`. Confirm that there are no VMs listed.

```
:~$ vagrant destroy
client: Are you sure you want to destroy the 'default' VM? [y/N]
==> client: Destroying VM and associated drives...
:~$ vagrant global-status
```

7 Submitting the Results of the Experiments

The experiments that you execute in this LAB will either produce results that you need to report or from which you will be asked questions about the execution. In order to report the results you achieved, proceed as follows:

7.1 General Procedure

The procedure for submission is quite simple:

RC 19/20	LAB ASSIGNMENT	Guide.Part #:	1.3
Application Layer		Issue Date:	07 Oct 2019
A Web Server and Web Client in Python		Due Date:	15 Oct 2019
Author: Prof. Rui Cruz		Revision:	4.0

1. In your Google Classroom for this course, you will find an Assignment for reporting this specific Lab experiment (with a Due Date);
2. The Assignment provides a Link to the TSUGI CLOUD Web Form with Exercise Questions;
3. When you are prepared with the requested materials (screen-shots, command line outputs, developed code, etc.) you will submit the items into the respective Exercise Form questions of the Assignment;
4. In the same Exercise Form of the Assignment you may be asked to comment your submissions;
5. Please note that in some types of Exercises you may also be asked to evaluate (anonymously) and provide feedback to the answers from some of your classmates (this peer-grading feedback counts for your grading).
6. When finished answering the Exercise Form, click the button **Done** in the top left of the Form; You will be returned to the Google Classroom Assignment;
7. In the Assignment, do not forget to confirm (click the button) **MARK AS DONE** or **TURN IN** when the assignment is completed;

7.2 Specific Procedure

For this LAB Assignment you will provide the Python code you have developed for both the Web server and the Web Client. Additionally, you will also provide results from using the Wireshark Tool, as well as from interacting with the Web Server. When answering a question below related with the analysis of packets, you should hand in a dissection of the packet(s) captured. To export a packet summary/data, in Wireshark use **File** → **Export Packet Dissections** → **As Plain text**, choose **Selected packet only**, choose **Packet summary line**, and select the minimum amount of packet detail that you need to answer the question.

Additionally some screenshots are also needed, as follows:

1. Submit the Python code for the Web Server in the TSUGI Form where asked;
2. Submit the Python code for the Web Client in the TSUGI Form where asked;
3. Capture a screenshot of the result of command (adjust the addresses and port to your system):
telnet 192.168.56.21 6789 to submit in the TSUGI Form where asked;
4. Capture a screenshot of the results of command (adjust the addresses and port to your system):
python webclient.py 192.168.56.21 6789 /index.html
to submit in the TSUGI Form where asked;

RC 19/20	LAB ASSIGNMENT	Guide.Part #:	1.3
Application Layer		Issue Date:	07 Oct 2019
A Web Server and Web Client in Python		Due Date:	15 Oct 2019
Author: Prof. Rui Cruz		Revision:	4.0

5. Using <http://draw.io> draw a UML sequence diagram illustrating the sequence of packets exchanged between the client and webserver when using telnet. For each packet, indicate the source and destination port numbers. Export the diagram in PNG format, to submit in the TSUGI Form where asked;.
6. Capture a screenshot of the Wireshark window showing the packets related with the first interaction (via Telnet), to submit in the TSUGI Form where asked;.
7. Report (content of a small txt file including the packets summary exported from Wireshark) **your analysis**, (i.e., what you observed) of those TCP packets exchanged while using “telnet”, to submit in the TSUGI Form where asked;.
8. Using <http://draw.io> draw a UML sequence diagram illustrating the sequence of packets exchanged between the client and webserver when using the “Web Client” developed. For each packet, indicate the source and destination port numbers. Export the diagram in PNG format, to submit in the TSUGI Form where asked;.
9. Capture a screenshot of the Wireshark window showing the packets related with the using the “Web Client” developed, to submit in the TSUGI Form where asked;.
10. Report (content of a small txt file including the packets summary exported from Wireshark) **your analysis**, (i.e., what you observed) of those TCP packets exchanged while using the “Web Client” developed, to submit in the TSUGI Form where asked;.

WARNING. Submissions MUST BE MADE in the TSUGI CLOUD Web Form through Classroom. No other type of submission will be considered for evaluation.