TÉCNICO LISBOA

RC 17/18	LAB ASSIGNMENT	Number:	4
Web and Email Scenarios		Issue Date:	10 Oct 2017
A Web server in Python		Due Date:	20 Oct 2017

Preliminary Notes

The instructions in this document are applicable to Computers at the IST Labs, which are already configured with the required tools. 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. The instructions are suitable for machines running the following Operating Systems:

- Microsoft Windows from version 7 up to 10
- Apple macOS from versions 10.8 'Mountain Lion' up to 11.13 'High Sierra'
- Debian-based Linux, such as Ubuntu (recommended) from versions 12.04 'Precise' to 16.04 'Xenial Xerus'.

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

1 A Webserver in Python

In this lab, you will learn the basics of 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 a HTTP packet. 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, we will develop a simple Web server in Python that is capable of processing only one request at a time. You have available information about the Python's socket interface in `https://docs.python.org/2/library/socket.html`.

2 Setting up the Experimental Environment

2.1 Configuring Shared Folders in Vagrant

One simple way to share informaton between one virtual machine, designated as ***guest***, and the machine that hosts it, known as the ***host***, is by mapping one folder or directory of the ***host***'s file system to one folder on the ***guest***'s file system. In order to have a shared folder you need to write in your Vagrantfile one line similar to the following, where the first path is related to the ***host*** and second path refers to the ***guest***.

```
client_config.vm.synced_folder "data/share_web", "/home/vagrant/
    shared_web"
```

Two HTML files, two JPEG files and a template for your server will be provided in the course website. You should make them available inside your machine. In order to do that, create a new directory inside the directory named "data" and name it `shared_web` and place all those new files inside it. After that, you should have one hierarchy of directories similar to the one in Figure 1:

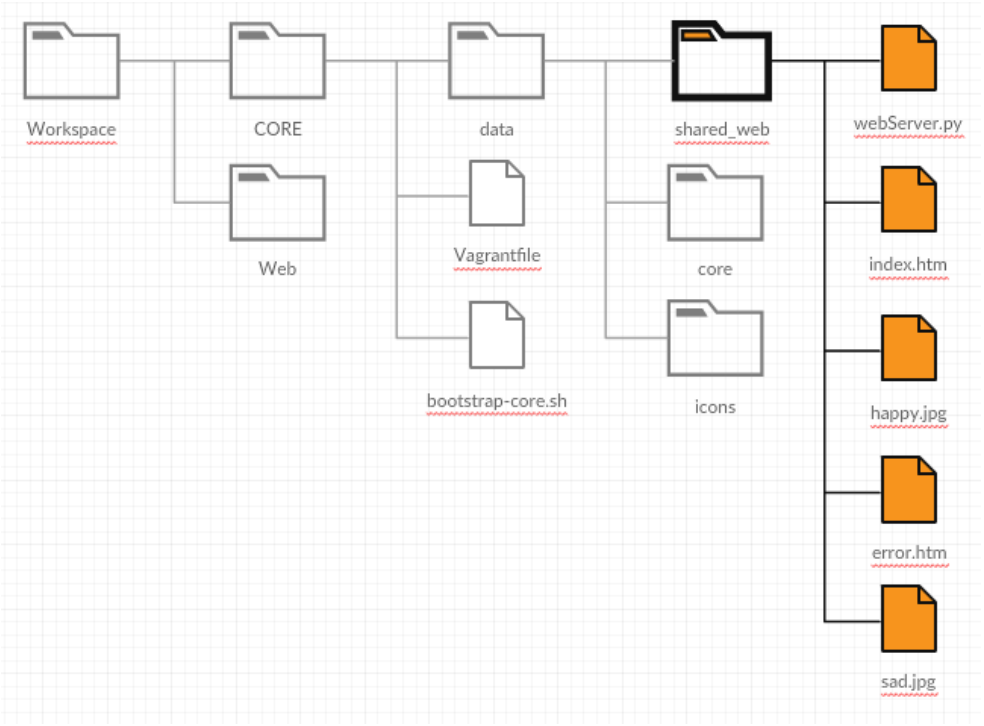

Figure 1: Directory Tree of our workspace

Now, change your Vagrantfile by adding the line that allows to share this new directory with your C.O.R.E guest machine. Your Vagrantfile should be similar to the following:

```
## -*- mode: ruby -*-
# vi: set ft=ruby :

# CORE emulator VM

Vagrant.configure("2") do |config|

  config.ssh.insert_key = false
  config.ssh.forward_x11 = true
```

```
  config.vbguest.auto_update = true

  #CORE VM configuration
  config.vm.define :core do |core_config|
    core_config.vm.box = "ubuntu/trusty64"
    core_config.vm.hostname = "core"
    core_config.vm.network :private_network, ip: "192.168.56.200"
    core_config.vm.synced_folder "data/core/configs", "/home/
       vagrant/.core/configs"
    core_config.vm.synced_folder "data/shared_web", "/home/vagrant
       /shared_web"
    core_config.vm.synced_folder "data/core/myservices", "/home/
       vagrant/.core/myservices"
    core_config.vm.provider "virtualbox" do |vb|
      vb.name = "core"
      opts = ["modifyvm", :id, "--natdnshostresolver1", "on"]
      vb.customize opts
      vb.memory = "2048"
    end
    core_config.vm.provision :shell, path: "bootstrap-core.sh"
  end
end
```

After this, all the modifications made to your files will be automatically accessible inside your virtual machine.

3 Developing a Webserver in Python

Specifically, your Web server will create a connection socket when contacted by a client (for example a browser), 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 browser.

If a browser 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. Your job is to complete the code, run your server, and then test your server by sending requests from a browser running on 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.

RC 17/18	LAB ASSIGNMENT	Number:	4
Web and Email Scenarios		Issue Date:	10 Oct 2017
A Web server in Python		Due Date:	20 Oct 2017

```
from socket import *
import sys # In order to terminate the program

HOST, PORT = '', 8000
serverSocket = socket(AF_INET, SOCK_STREAM)
#Bind your socket
#Fill in start #Fill in end
while True:
    #Listen and wait for request
    #Fill in start #Fill in end
    #Establish the connection
    print('Ready to serve in port %s...' % PORT)
    connectionSocket, addr =  #Fill in start #Fill in end
    try:
        #Read the message sent
        message = #Fill in start #Fill in end
        print(message)
        #Parsing request
        filename = message.split()[1]
        if filename[1:]=="":
            filename="/index.htm"
        f = open(filename[1:], 'rb')
        outputdata = f.read()
        #Send response message
        #Fill in star #Fill in end
        #Send the content of the requested file to the client
        for i in range(0, len(outputdata)):
            connectionSocket.send(outputdata[i])
        connectionSocket.send("\r\n")
        #Close client socket
        connectionSocket.close()
    except IOError:
        filename = "/error.htm"
        f = open(filename[1:], 'rb')
        outputdata = f.read()
        #Send response message for file not found
        connectionSocket.send("HTTP/1.1 404 Not Found\r\n\r\n")
        #Send the content of error file to the client
        #Fill in start #Fill in end
        #Close client socket
        #Fill in start #Fill in end
serverSocket.close()
sys.exit()#Terminate the program
```

RC 17/18	LAB ASSIGNMENT	Number:	4
Web and Email Scenarios		Issue Date:	10 Oct 2017
A Web server in Python		Due Date:	20 Oct 2017

3.1 Starting your server with C.O.R.E

Please confirm that you have X11 server running (in Linux, it should already be running, but in case you use Windows, or macOS, you need to confirm that XMing or XQuartz, respectively are running. Now that we have everything configured, start the machine with `vagrant up`, and when system is ready, open the graphical user interface of C.O.R.E.:

```
:~$ vagrant up
...
...
:~$ vagrant ssh -c gui-core -- -X
Connecting to core "core-daemon"... connected!
```

In C.O.R.E. we need to create a simple network between two machines. The network will consist of one PC and one Server connected by a router, as illustrated in Figure 2.

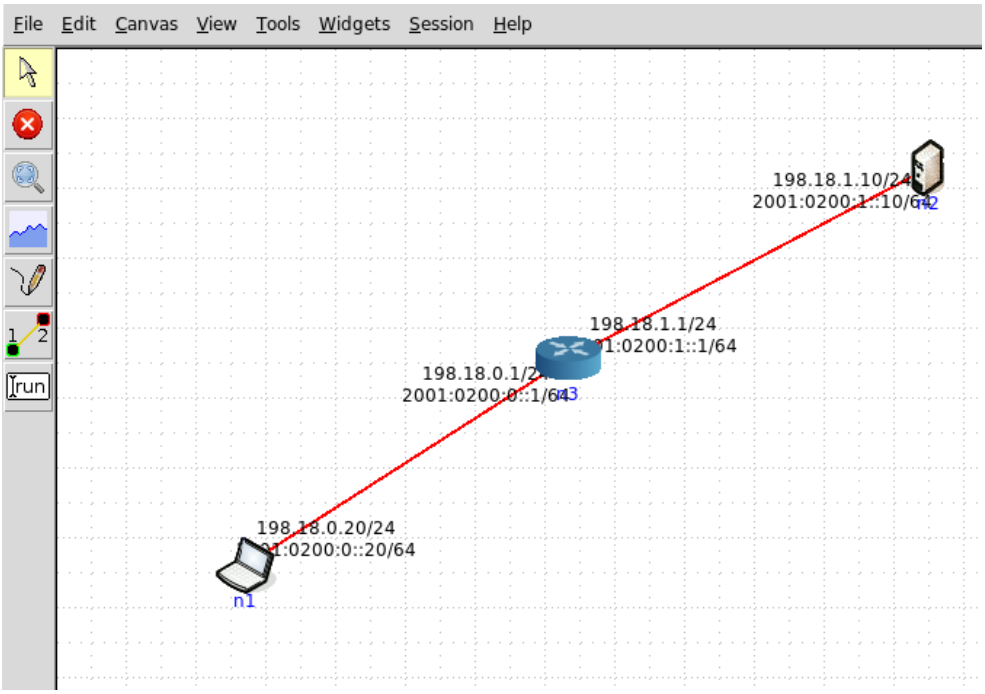

Figure 2: Simple network created on C.O.R.E.

Open a Terminal (with bash) for each one of your end systems and open a wireshark window in one of them. In your server, go inside your shared folder and run your server:

```
:~$ cd web_shared
:~$ python webServer.py
Ready to serve in port 8000...
```

Now in the command line for your client run a telnet connection to your server and test it using a simple connect request and a request for a file that does not exist.

```
:~$ telnet 198.18.1.10 8000
Trying 198.18.1.10 ...
Connected to 198.18.1.10.
Escape character is '^]'.
GET / HTTP/1.1
```

Analyse the wireshark capture. To ensure your server can be used as a normal Web server, we can use an already implemented Web client, for example a browser, to see if it works properly.

3.2 Starting your server outside C.O.R.E.

So close all core windows and connect to the machine:

```
:~$ vagrant ssh
```

Now, inside your `shared_web` folder, start your server:

```
:~$ cd web_shared
:~$ python webServer.py
Ready to serve in port 8000...
```

After this, you can use your browser to access your web server (at $192.168.56.200:8000$) and see the requests made by analyzing the output of your server.

4 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
==> default: Attempting graceful shutdown of VM...
:~$ vagrant global-status
```

Confirm that the statuses of the VMs is 'powered off'.

In Lab computers, also do the following (answering "y"), in order to ensure the system is cleaned from your experiments. Note however that the configuration files remain stored in your private area in AFS:

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

Confirm that there are no VMs listed.

5 Report the results

The experiences in this lab are for evaluation. In order to report the results you achieved, proceed as follows. You will hand in the complete server code along with the screen shots of your client browser, verifying that you actually receive the contents of the HTML file from the server.

5.1 Lab Report

Create a Google Doc Lab Report using one of the Google Docs ”Science Report” Template. In the report, explain what you did and include the relevant screen shots with captions, the text results of commands or wireshark captures of the experiments, explaining them.

Submission of your work:

Filename: Your report MUST be named **RC1718-LAB04-Report-IDofStudent** where “**IDofStudent**” must have the format “ist123456”.

Python code: Your Python code file MUST be named **RC1718-LAB04-WebServer-IDofStudent.py** where “**IDofStudent**” must have the format “ist123456”.

Submission Form: Your work is submitted in a Google Form available from the Assignment published in Google Classroom.

Due Date: Please respect the due date of the assignment, considering the time limit of 23:59h.

WARNING. Incorrect submissions will not be considered for evaluation.