RC-I 2024-2025	TRABALHO DE LABORATÓRIO	Número:	3
CAMADA DE APLICAÇÃO		Data:	
Um servidor Web e Cliente Web em Python		Prazo:	

1. Introdução

O objectivo desta tarefa de laboratório é utilizar o ambiente multi-nós (servidores, rede) e aprender os conceitos básicos de programação de sockets de rede para conexões TCP em Python.

2. Servidor Web e Cliente Web em Python

Neste laboratório, aprenderá sobre programação de socket de rede para conexões TCP em Python: como criar um socket, vinculá-lo a um endereço e porta específicos, bem como enviar e receber pacotes HTTP. Também aprenderá alguns conceitos básicos do formato de cabeçalho HTTP.

Socket é o nome dado à abstracção de uma interface utilizada para comunicação entre processos. No contexto de redes de computadores, nos comunicamos com processos em diferentes máquinas (virtuais e/ou físicas). As estruturas que implementam essa abstracção são geralmente geridas pelo sistema operativo e são manipuladas por programadores utilizando uma *Application Program Interface* (API).

Para nosso propósito, desenvolverá um servidor Web simples e um cliente Web em Python. Tem informações disponíveis sobre a interface de socket do Python em https://docs.python.org/2/library/socket.html para **Python 2** ou https://docs.python.org/3.7/library/socket.html para **Python 3**.

2.1. O Servidor Web

O Servidor Web é capaz de processar apenas um pedido por vez. Seu servidor web deve aceitar e analisar o pedido HTTP, obter o ficheiro solicitado do sistema de ficheiros do servidor, criar uma mensagem de resposta HTTP consistindo do ficheiro pedido precedido por linhas de cabeçalho e, em seguida, enviar a resposta directamente ao cliente. Se o ficheiro pedido não estiver presente no servidor, o servidor deve enviar uma mensagem HTTP “404 Not Found” de volta ao cliente Web.

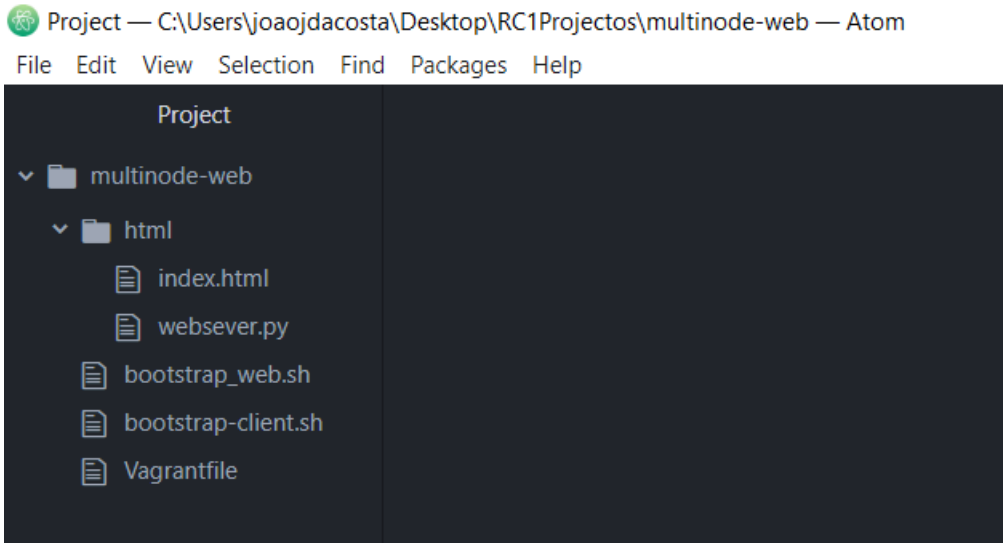
2.2. O Cliente Web

O Cliente Web HTTP pode ser utilizado para testar seu Servidor Web. Seu cliente se conectará ao servidor utilizando uma conexão TCP, enviará um pedido HTTP ao servidor e apresentará a resposta do servidor como uma saída. Pode assumir que o pedido HTTP enviado é um método GET. O cliente deve receber argumentos de linha de comando a especificar o endereço IP do servidor ou o nome do host, a porta na qual o servidor está a escutar e o caminho no qual o objecto pedido é armazenado no servidor.

3. Configuração do Ambiente de Experiências

Para começar, vá para o directório do projecto criado para LAB#1 e LAB#2, nomeado, por exemplo, multinode. Dentro da pasta do projecto cria a pasta "multinode-webserver". Faça download do ficheiro `RCI2025-Lab3_support_files.zip` e descompacte o conteúdo para a pasta do projecto `multinode-webserver`.

A pasta do projecto multinode-webserver deve ter uma estrutura semelhante à seguinte:



Um ficheiro chamado `webserver.py` será criado na pasta "html". Essa pasta também deve conter o ficheiro "`index.html`" do LAB#2.

Na pasta `multinode-webserver`, edite o `Vagrantfile` de modo que fique semelhante ao seguinte:

Vagrantfile

```
# -*- mode: ruby -*-
# vi: set ft=ruby :
Vagrant.configure(2) do |config|
  config.ssh.insert_key = false
  config.vbguest.auto_update = true
  config.ssh.forward_x11 = true

  config.vm.define "webserver" do |web_config|
    web_config.vm.box = "ubuntu/trusty64"
    web_config.vm.hostname = "webserver"
    web_config.vm.network "private_network", ip: "192.168.56.21"
    #web_config.vm.network "forwarded_port", guest: 80, host: 8080
    web_config.vm.synced_folder "html", "/home/vagrant/html"
    web_config.vm.provider "virtualbox" do |vb|
      vb.name = "webserver"
      opts = ["modifyvm", :id, "--natdnshostresolver1", "on"]
      vb.customize opts
      vb.memory = "256"
    end # of vb
    # web_config.vm.provision "shell", path: "bootstrap_web.sh"
  end # of web_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: "192.168.56.11"
    client_config.vm.provider "virtualbox" do |vb|
      vb.name = "client"
      opts = ["modifyvm", :id, "--natdnshostresolver1", "on"]
      vb.customize opts
      vb.memory = "256"
    end # of vb
    client_config.vm.provision "shell", path: "bootstrap-client.sh"
  end # of client_config

end # of config
```

Em outras janelas do Terminal, inicie a conexão `ssh` com ambas as máquinas:

```
$ vagrant ssh client
```

```
$ vagrant ssh webserver
```

3. Utilização de Pasta Partilhada no Vagrant

Uma maneira simples de partilhar informações entre uma máquina virtual, designada como **guest**, e a máquina que a hospeda, conhecida como **host**, é mapeando uma pasta ou directório do sistema de ficheiros do host para uma pasta no sistema de ficheiro do guest.

Para ter uma pasta partilhada, precisa garantir que no seu Vagrantfile tenha pelo menos uma linha semelhante à seguinte, onde o primeiro caminho está relacionado ao **host** e o segundo caminho se refere ao **guest**.

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

Tendo esse tipo de partilha para ambas as VMs, garante que tudo o que colocar na pasta “html” do seu host aparecerá e estará disponível dentro da VM na pasta home “user”.

4. Desenvolvimento de um Servidor Web em Python

Especificamente, seu servidor Web criará um socket de conexão quando contactado por um cliente Web que receberá a solicitação HTTP dessa conexão, analisará a solicitação para determinar o ficheiro específico que está a ser solicitado, obterá o ficheiro solicitado do sistema de ficheiros do servidor, criará uma mensagem de resposta HTTP consistindo no ficheiro solicitado precedido por linhas de cabeçalho e enviará a resposta pela conexão TCP ao cliente solicitante.

Se um cliente solicitar um ficheiro que não está presente no seu servidor, o servidor deverá retornar uma mensagem de erro “404 Not Found” e uma página de erro deverá ser enviada.

O esqueleto do código para seu servidor é o seguinte (e já deve estar lá na pasta “html”). Seu trabalho é completar o código, executar seu servidor e então testá-lo enviando solicitações de outros hosts.

Os espaços onde precisa preencher o código são marcados com `#Preencha o início` e `#Preencha o fim`. Cada espaço pode exigir uma ou mais linhas de código.

websever.py

```
"""
Lab3 - Servidor e Cliente Web

Docente: João Costa (joaojdacosta@gmail.com)
Outubro, 2024.
"""

from socket import *  # importa o módulo socket
import sys # Por forma a terminar o programa

# Cria um socket do servidor TCP
#(AF_INET é utilizado para o protocolo IPv4)
#(SOCK_STREAM é utilizado para o TCP)
serverSocket = socket(AF_INET, SOCK_STREAM)

# Atribui o número de porta
serverPort = 6789

# TODO #1:  Vincular o socket ao endereço e porta do servidor
# Preencha o início

# Preencha o fim

```

websever.py

```
# TODO #2: Escutar, no máximo, 1 conexão por vez
# Preencha o início

# Preencha o fim

# O servidor deve estar Up, em execução e a escuta
# por novas conexões

while True:
    print('O servidor está pronto para receber')

    # TODO #3: Configurar uma nova conexão do cliente
    # Preencha o início

    # Preencha o fim

    # Se ocorrer uma excepção durante a execução da cláusula try
    # o resto da cláusula é ignorado
    # Se o tipo de exceção corresponder à palavra após except
    # a cláusula except é executada
    try:
        # TODO #4: Receber a mensagem de solicitação do cliente
        message = #Preencha o início    #Preencha o fim }@>
```

```
        # Extrae o caminho do objecto solicitado da mensagem
        # O caminho é a segunda parte do cabeçalho HTTP,
        # identificado por [1]
        filename = message.split()[1]
        # O caminho extraído da solicitação HTTP inclui
        # um caractere '\', lemos o caminho do
        # segundo caractere
        f = open(filename[1:])
        # Armazena todo o conteúdo do ficheiro solicitado
        # em um buffer temporário
        outputdata = f.read()
        # TODO #5: Enviar a linha de cabeçalho de resposta HTTP
        # para o socket de conexão
        # Preencha o início

        # Preencha o fim

        # Envia o conteúdo do ficheiro solicitado
        # para o socket de conexão
        for i in range(0, len(outputdata)):
            connectionSocket.send(outputdata[i].encode())
        connectionSocket.send("\r\n".encode())

```

```
        # Fecha o socket de conexão do cliente
        connectionSocket.close()

    except IOError:
            # TODO #6: Enviar mensagem de resposta HTTP
            # para ficheiro não encontrado
            connectionSocket.send("HTTP/1.1 404 Not Found\r\n\r\n".encode())
            # Preencha o início

            # Preencha o fim

            # TODO #7: Fechar o socket de conexão do cliente
            # Preencha o início

            # Preencha o fim

serverSocket.close()
sys.exit()# Encerra o programa após enviar os dados correspondentes.

```

Quando tiver o código completo, abra um Terminal (com bash) para cada sistema. Abra também uma janela do Wireshark da máquina Cliente e selecione a interface para capturar.

Na máquina do servidor Web, entre na sua pasta partilhada e execute seu servidor com:

```
vagrant@webserver:~/html$ python websever.py
O servidor está pronto para receber
```

Agora, na linha de comando para sua máquina Cliente, execute uma conexão `telnet` com seu Servidor Web e teste-a solicitando um ficheiro, usando HTTP. Os endereços IP e o número da Porta podem ser diferentes. Adapte o comando ao seu caso.

```
$ 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
```

Analise a captura do Wireshark e tente identificar as mensagens trocadas entre o Cliente e o Servidor Web.

5. Desenvolvimento de um Cliente Web em Python

Inspirado no código do Servidor Web, desenvolva o código para o simples Cliente Web.

Nomeie o ficheiro "`webclient.py`" e coloque-o dentro da pasta "`html`" (o ficheiro aparecerá dentro da máquina cliente).

Para iniciar o Cliente Web, entre na pasta partilhada e execute o cliente (como no exemplo a seguir, usando o endereço IP e a porta do servidor Web):

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

A sintaxe do commando é `webclient.py server_host server_port filename`.

Analise a captura do Wireshark e tente identificar as mensagens trocadas entre o Cliente Web e o Servidor Web.

6. Finalização das experiências

Para parar as Máquinas Virtuais e verificar o estado global de todos os ambientes Vagrant activos no sistema, podemos emitir os seguintes comandos:

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

Confirme se o status das VMs é '`powered off`'. Para evitar que essas máquinas instanciadas utilizem recursos, pode destruí-las, pois agora elas podem ser recriadas com o comando simples `vagrant up`. Confirme se não há VMs listadas.

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

6. Envio dos resultados das experiências

As experiências que executar neste LAB produzirão resultados que precisa relatar ou dos quais será questionado sobre a execução. Para relatar os resultados que alcançou, proceda da seguinte forma:

6.1. Procedimento geral

Deve enviar via email um memorando contendo todo o material requerido para aferir a execução do laboratório (screen-shots, saída de linha de comando, código desenvolvido, etc.).

6.2. Procedimento específico

Para esta Tarefa LAB, fornecerá o código Python que desenvolveu para o servidor Web e o Cliente Web bem como todos os scripts, via GitHub. Além disso, também fornecerá resultados da análise de pacote realizada. Além disso, algumas capturas de tela também são necessárias, como segue:

1. Envie o `Vagrantfile` modificado;
2. Envie o código Python para o Servidor Web;
3. Envie o código Python para o Cliente Web;
4. Anexar, no memorando, a captura de tela do resultado do comando:
   `telnet 192.168.56.21 6789`
5. Anexar, no memorando, a captura de tela dos resultados do comando:
   `python webclient.py 192.168.56.21 6789 /index.html`
6. Relate, no memorando, sua análise dos pacotes TCP trocados usando "`telnet`";
7. Relate, no memorando, sua análise dos pacotes TCP trocados usando "`Cliente Web`";

Bom trabalho!