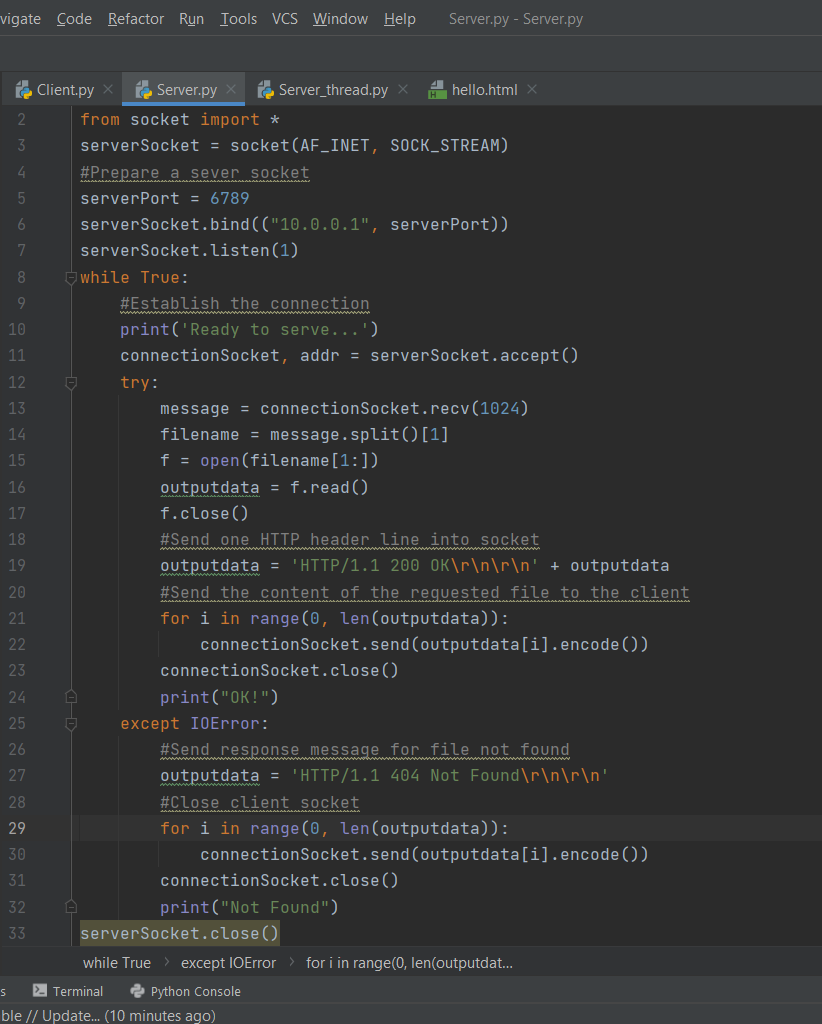
# **Assignment1**

## **Servidor que maneja 1 solicitud a la vez**

### **Código**



### **Explicación**

**serverSocket = socket(AF\_INET, SOCK\_STREAM)**: Aquí se crea un objeto de socket. AF\_INET se refiere a la familia de direcciones del protocolo de internet, mientras que SOCK\_STREAM significa que es un socket TCP.

**serverPort = 6789**: Aquí se define el puerto del servidor al que se conectará el cliente.

**serverSocket.bind(("10.0.0.1", serverPort))**: Aquí se vincula el socket del servidor a una dirección IP específica y un puerto.

**serverSocket.listen(1)**: Aquí el servidor comienza a escuchar las conexiones entrantes. El argumento '1' especifica el número máximo de conexiones en cola.

**while True:**: Este es un bucle infinito donde el servidor sigue aceptando nuevas conexiones y procesándolas.

**connectionSocket, addr = serverSocket.accept()**: Aquí el servidor acepta una conexión entrante y devuelve un nuevo socket y la dirección del cliente.

**message = connectionSocket.recv(1024)**: Aquí el servidor recibe datos del cliente. El argumento '1024' especifica el tamaño máximo de los datos que se pueden recibir a la vez.

**filename = message.split()[1]**: Aquí se extrae el nombre del archivo de la solicitud del cliente.

**f = open(filename[1:])**: Aquí se abre el archivo solicitado.

**outputdata = f.read()**: Aquí se lee el contenido del archivo.

**outputdata = 'HTTP/1.1 200 OK\r\n\r\n' + outputdata**: Aquí se prepara la respuesta HTTP para enviar al cliente.

**for i in range(0, len(outputdata)):**: Aquí se envía la respuesta al cliente carácter por carácter.

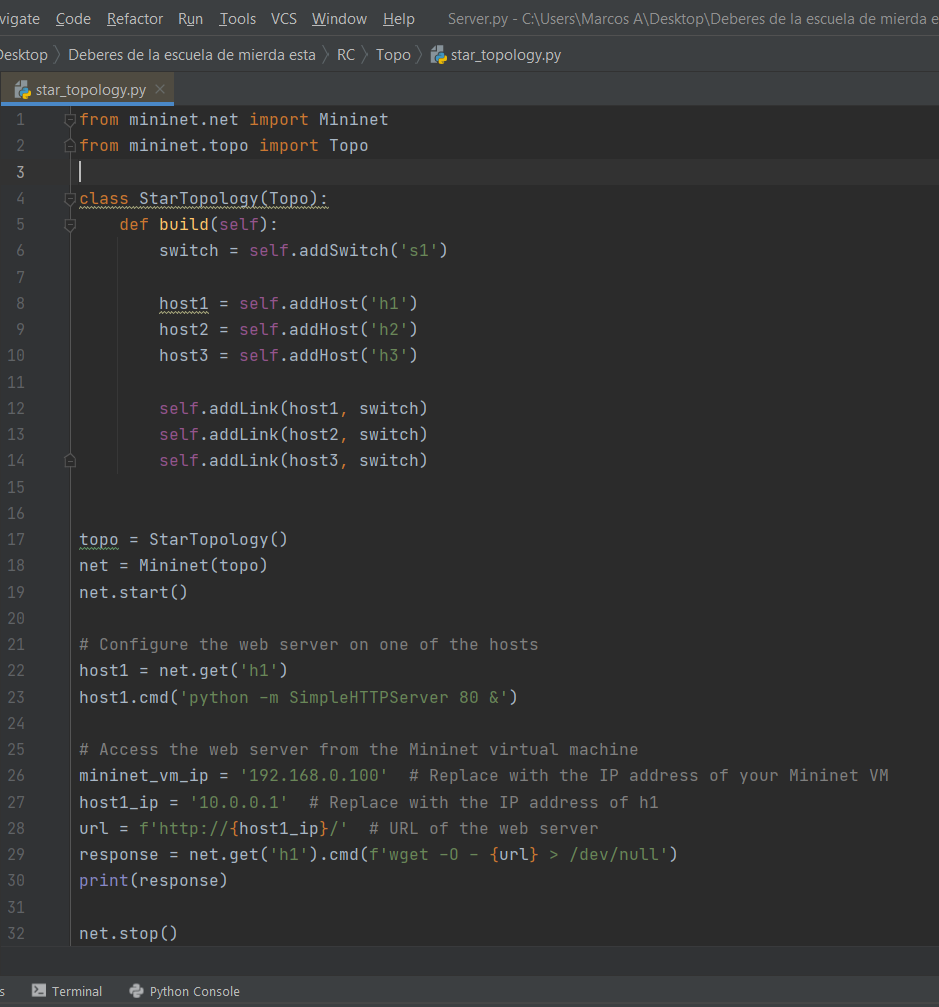
**connectionSocket.close()**: Aquí se cierra la conexión con el cliente.

**except IOError:**: Esta es una excepción que se lanza cuando el archivo solicitado no se encuentra.

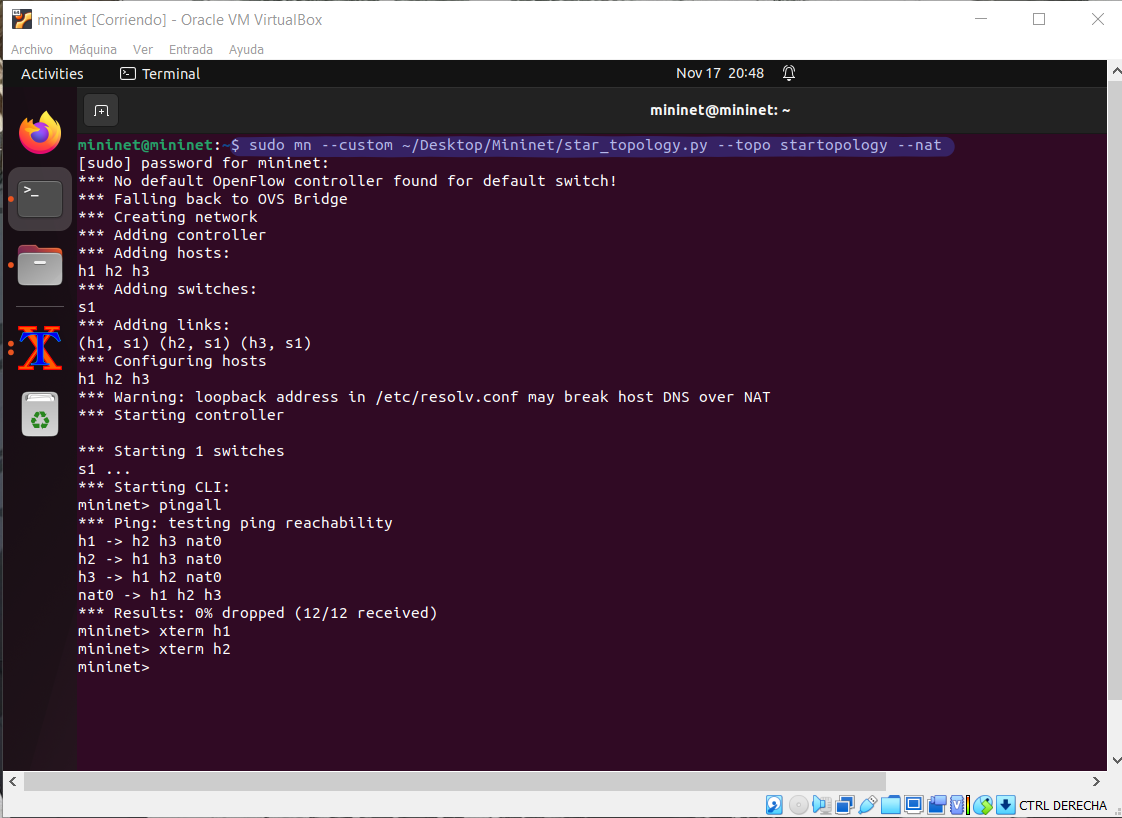
**outputdata = 'HTTP/1.1 404 Not Found\r\n\r\n'**: Aquí se prepara la respuesta HTTP 404 para enviar al cliente si el archivo no se encuentra.

**serverSocket.close()**: Finalmente, se cierra el socket del servidor.

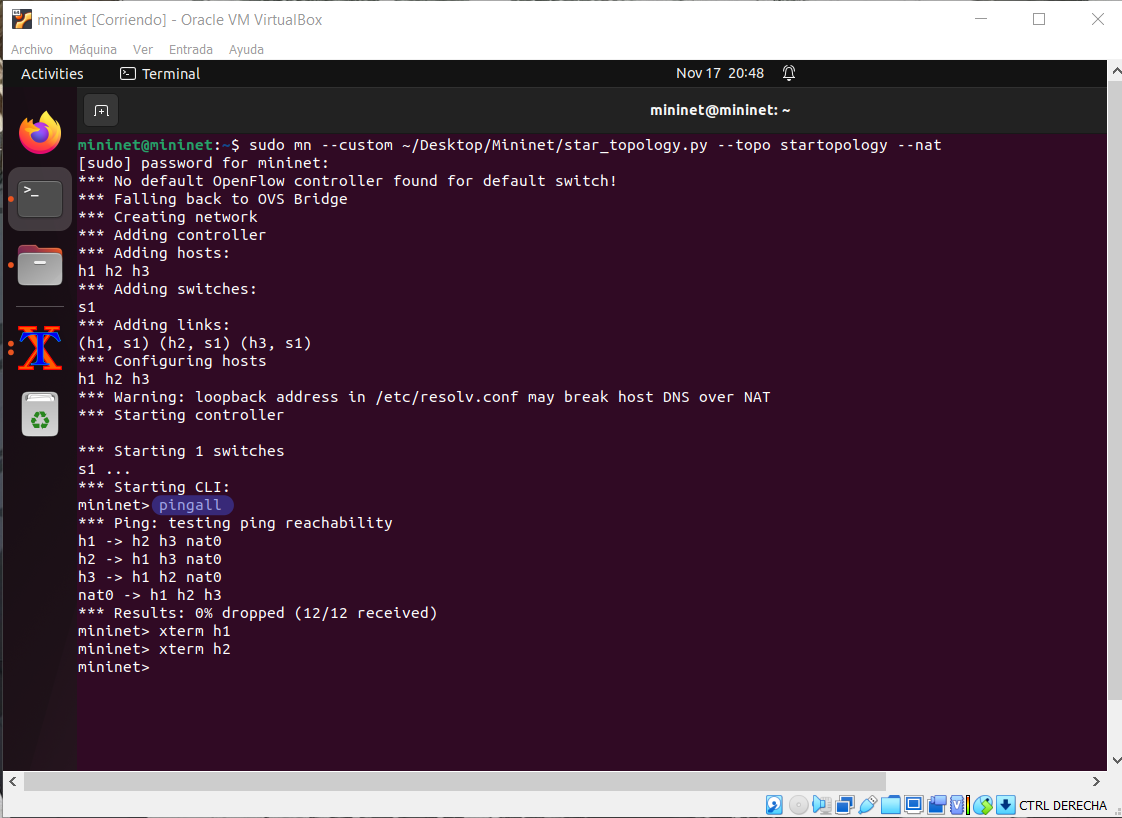
### **Despliegue en Mininet**

Topología utilizada

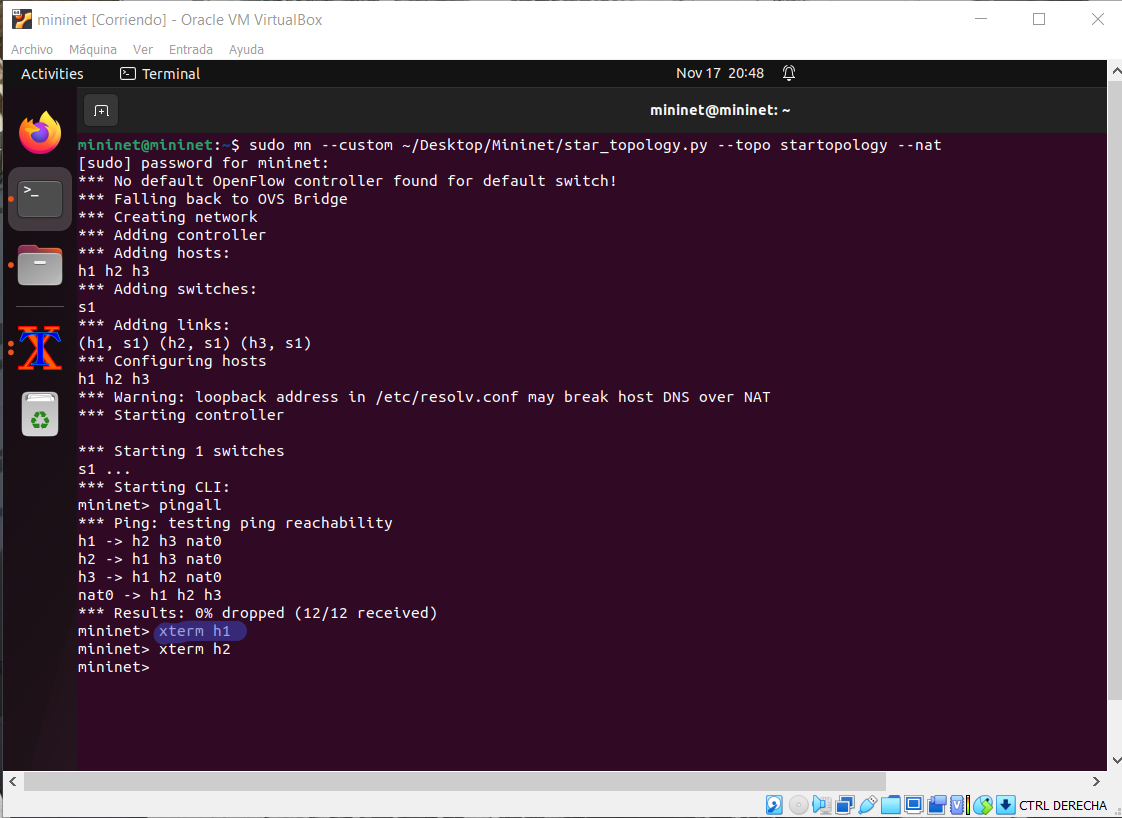
1)Primeramente, levantamos la topología creada previamente en un archivo .py inspeccionando la ruta donde este se encuentra con el comando: **sudo mn --custom ~/Desktop/Mininet/star\_topology.py --topo startopology –nat**



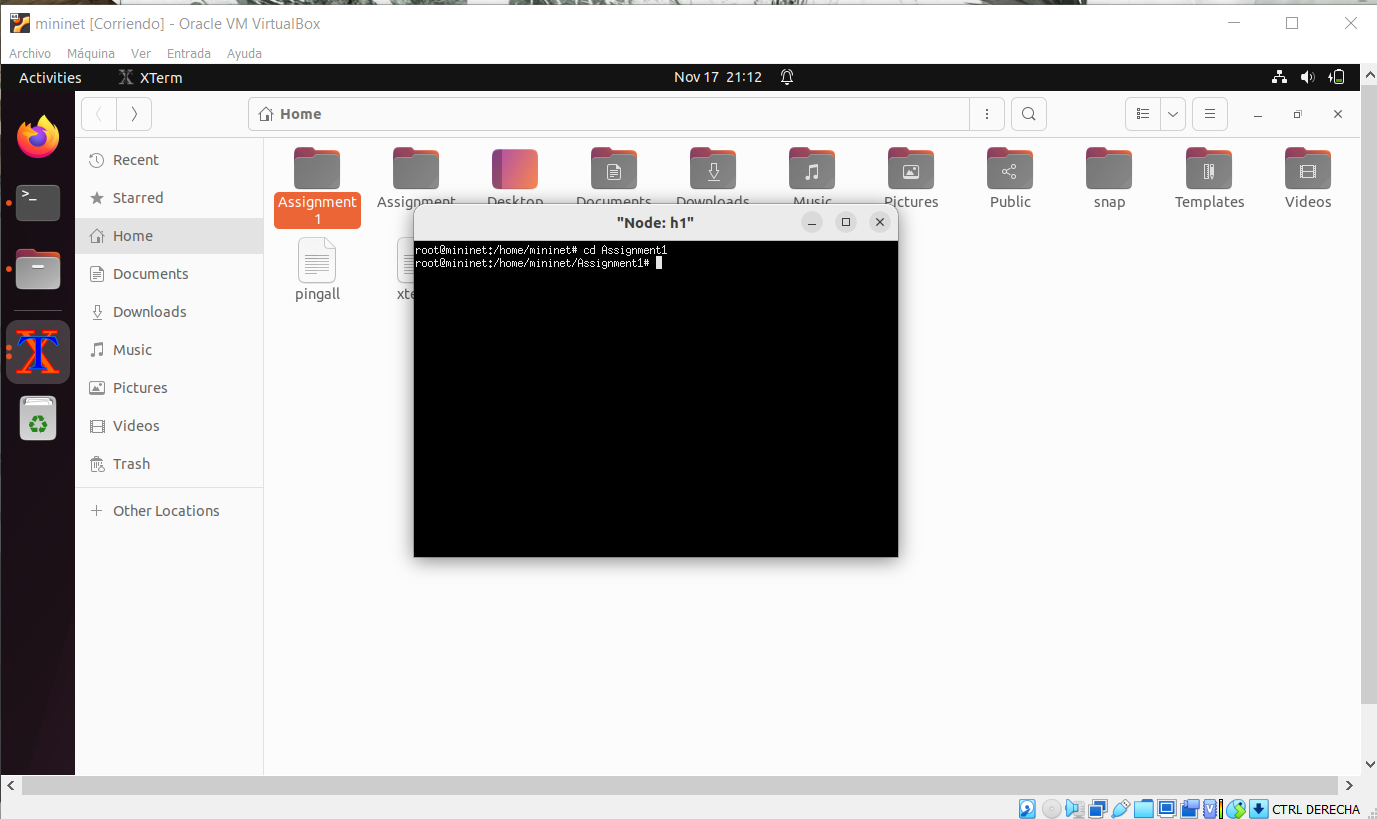
2)Opcionalmente ejecutamos el comando **pingall** para verificar que se levantó el servicio correctamente



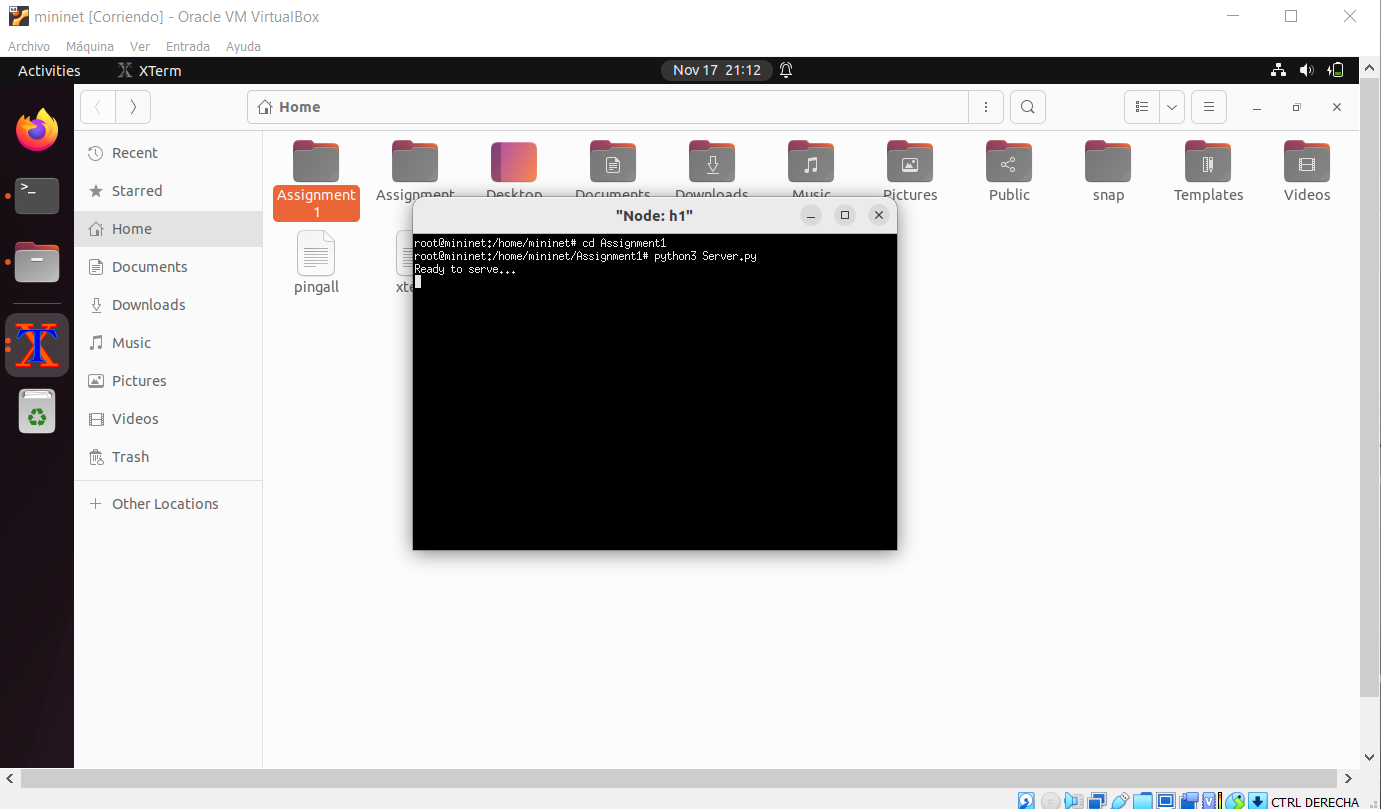
3)Levantar un servicio web, para ello tomamos en que host se levantará el servicio web (h1) y para acceder a él ponemos en la consola el comando x**term h1**



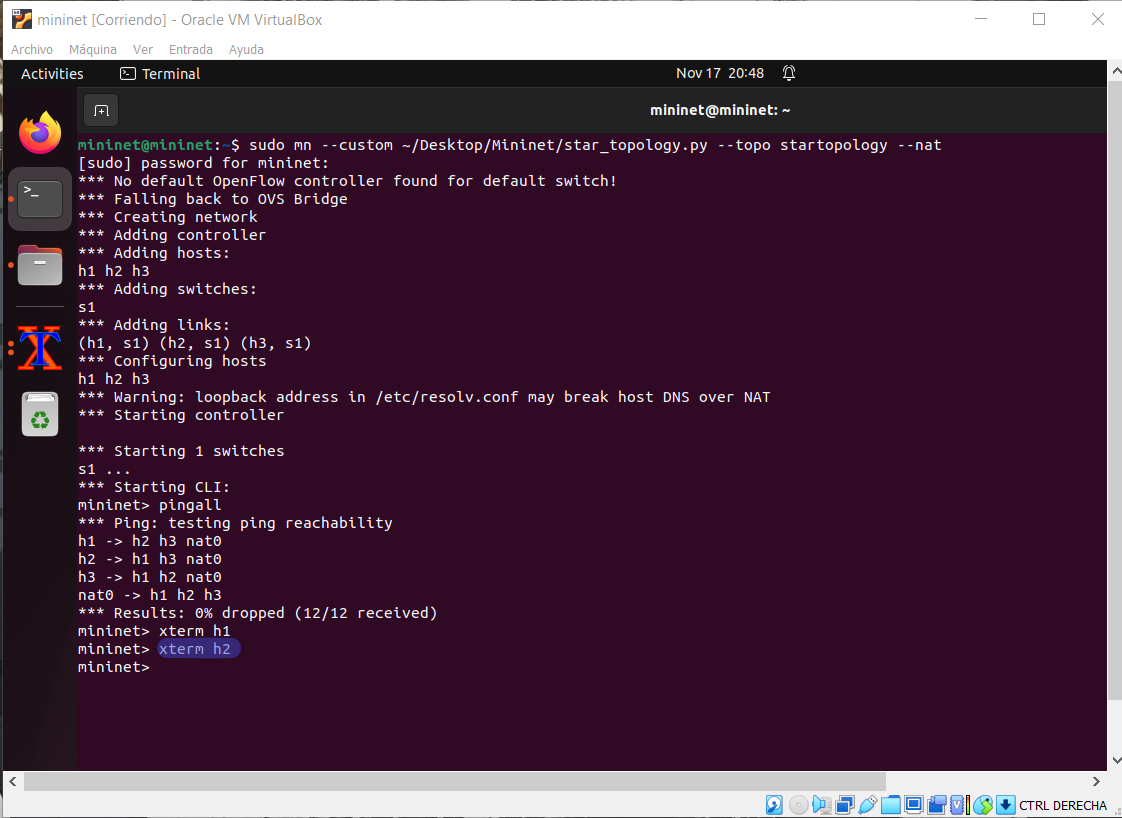
Buscar la ruta donde se encuentra el archivo Server.py con el comando **cd Assignment1**



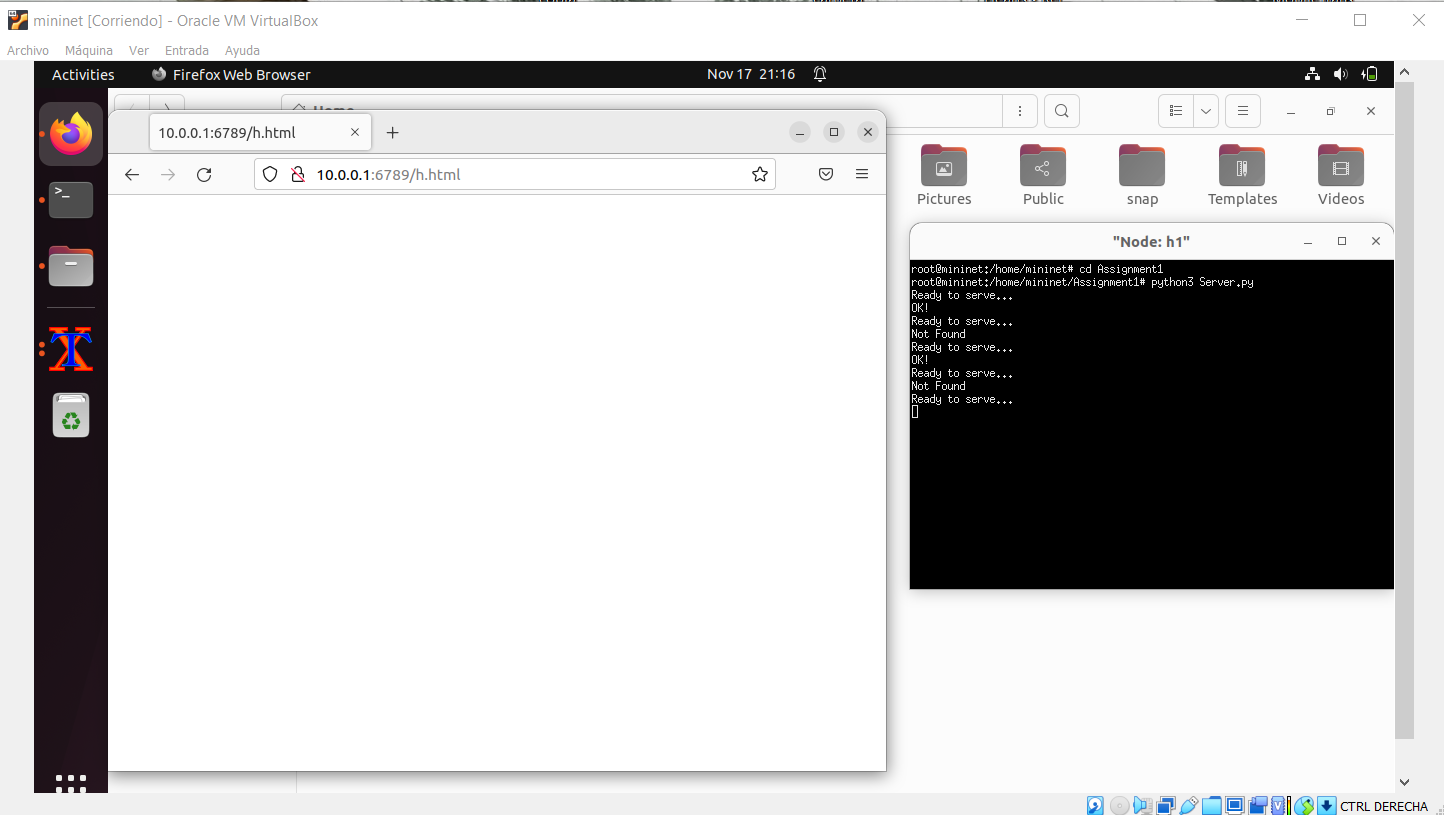
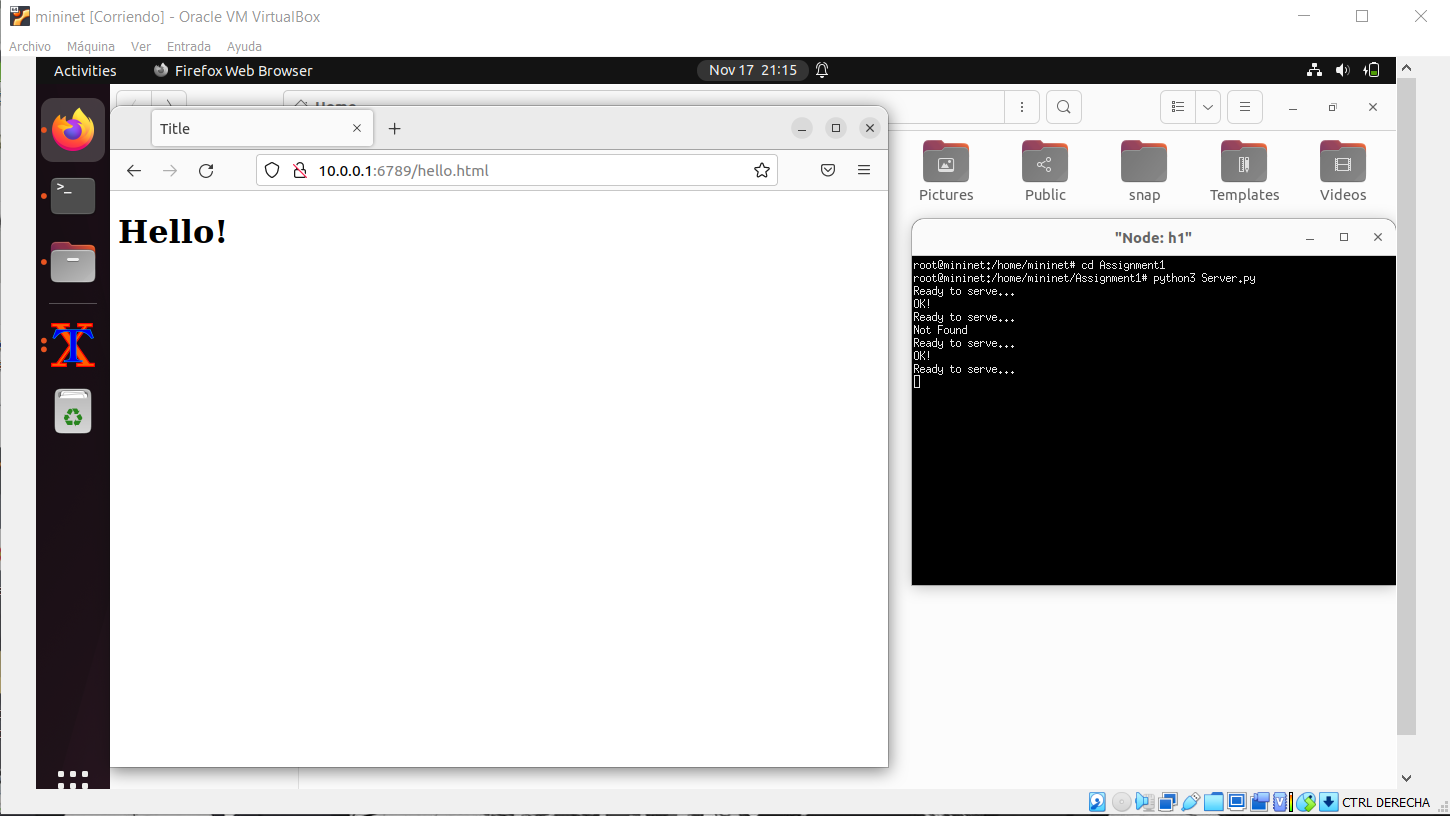
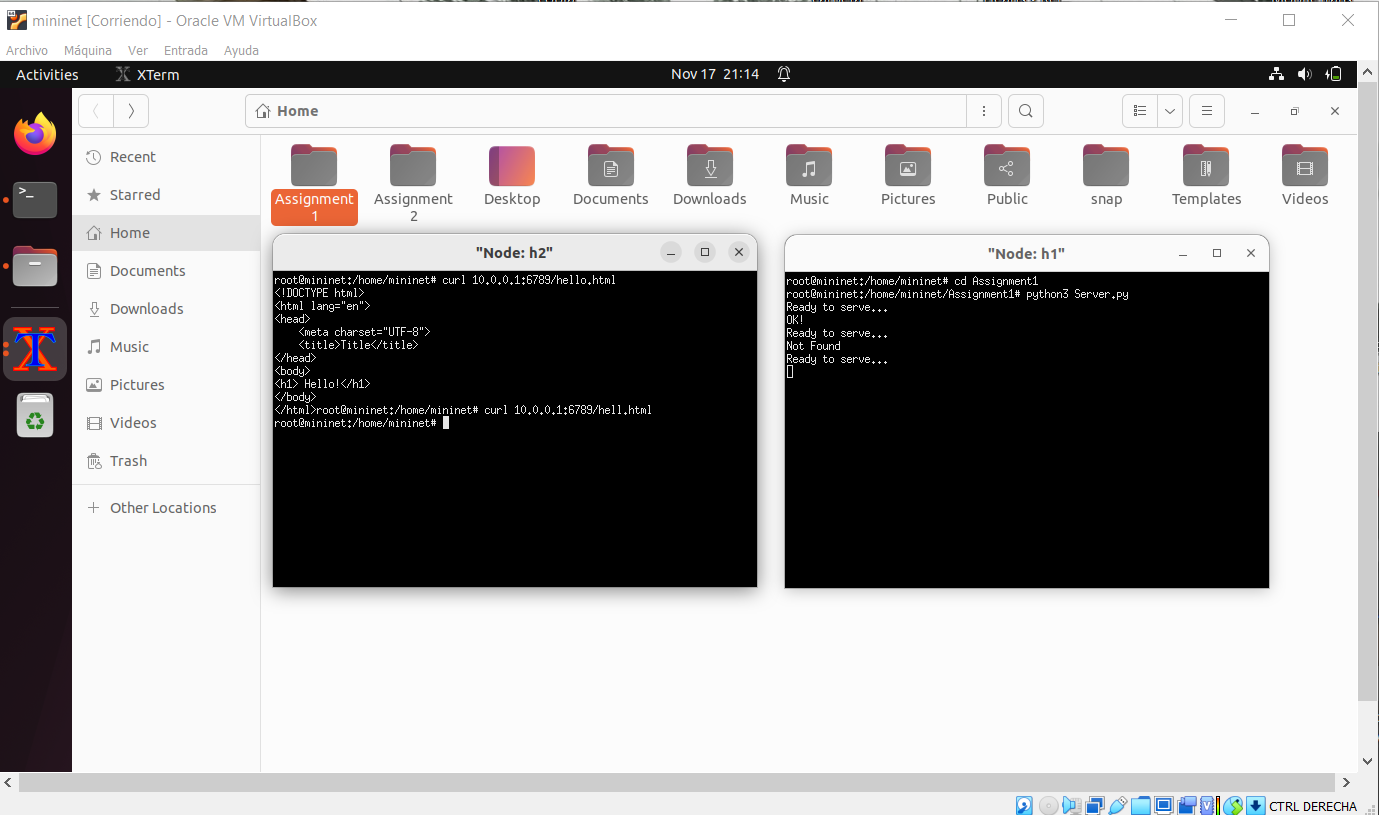
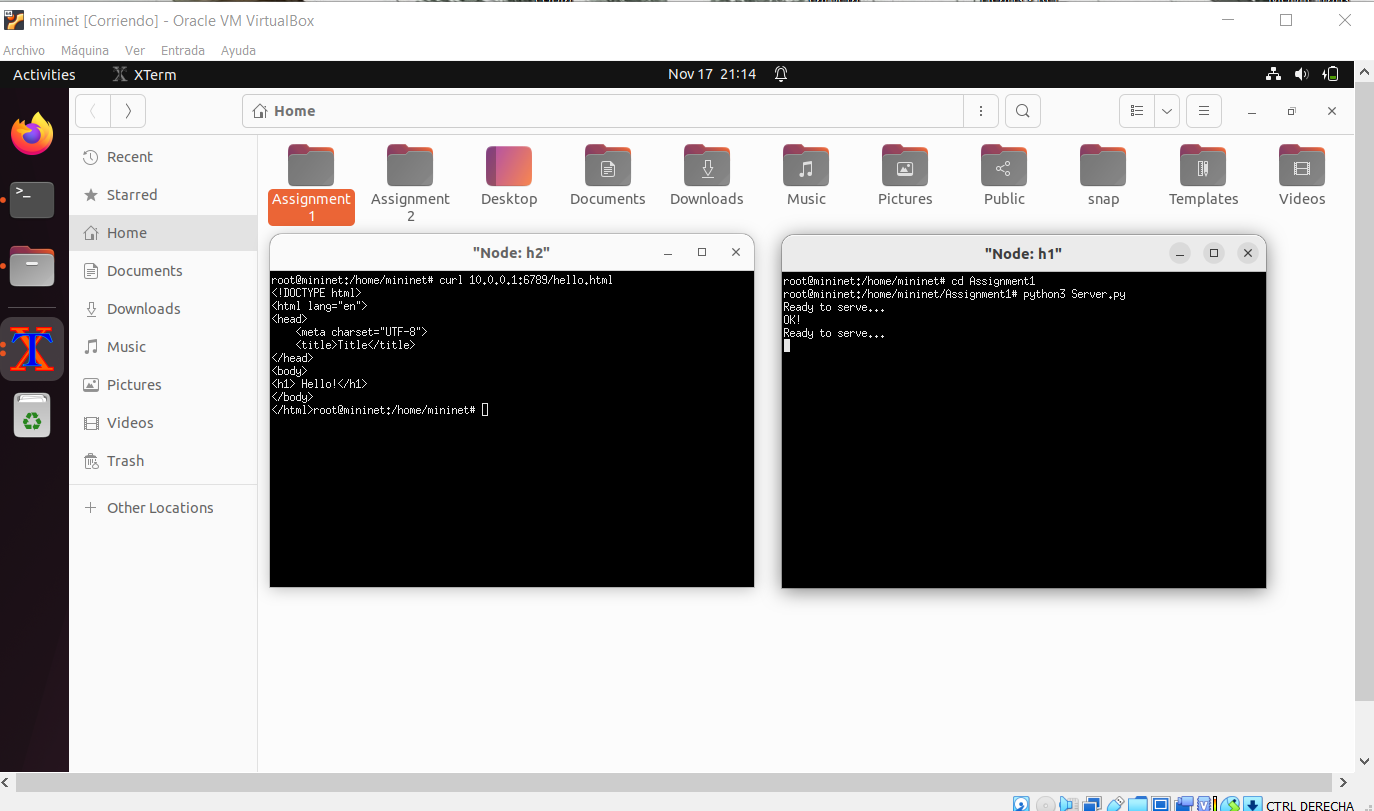
Ya establecida la ruta, ejecutar Server.py con el comando **phyton3 Server.py**



4)Establecer un host cliente(h2) que realizará peticiones al servidor(h1) con el comando **xterm h2**



Realizar la petición desde h2 con el comando **curl 10.0.0.1:6789/nombre del recurso solicitado** (en este caso **hello.html**) o desde el navegador.



Otra vía es crear un cliente HTTP para probar tu servidor. El cliente se conecta al servidor mediante una conexión TCP, envía una solicitud HTTP al servidor y muestra respuesta como salida al servidor.

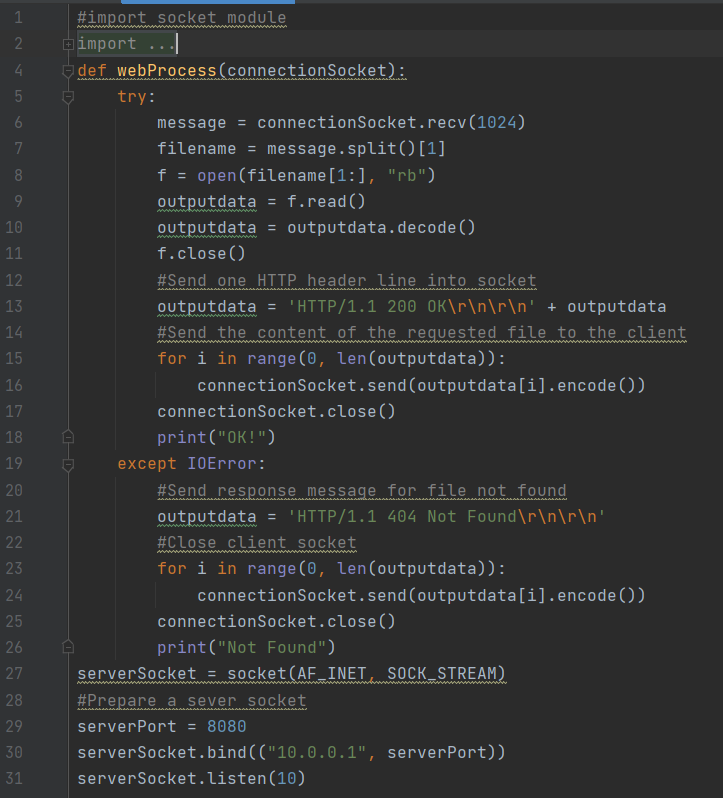
## Texto Descripción generada automáticamente

En h2 vas para la carpeta raíz que contiene al documento .py y lo ejecutas en este casocon el comando **python3 Client.py**

## 

## **Servidor multiproceso**

### **Código**



### **Explicación**

El código es exactamente el mismo que el anterior, pero con la única diferencia que este puede manejar múltiples conexiones simultáneamente utilizando multithreading.

Todo el contenido de la función anterior que constituye la definición de una función que maneja la conexión con un cliente tomando un socket como argumento, yace en una función llamada **def webProcess(connectionSocket)**

Ahora bien, dentro del bnucle infinito, se encuentra el procesamiento multihilo y es donde se llama a la función

**thread = threading.Thread(target = webProcess, args = (connectionSocket, )):** Aquí se crea un nuevo hilo para manejar la conexión con el cliente. La función **webProcess** se pasa como el objetivo del hilo, y el socket de conexión se pasa como argumento.

**thread.start():** Aquí se inicia el nuevo hilo.

**serverSocket.close():** Finalmente, se cierra el socket del servidor.

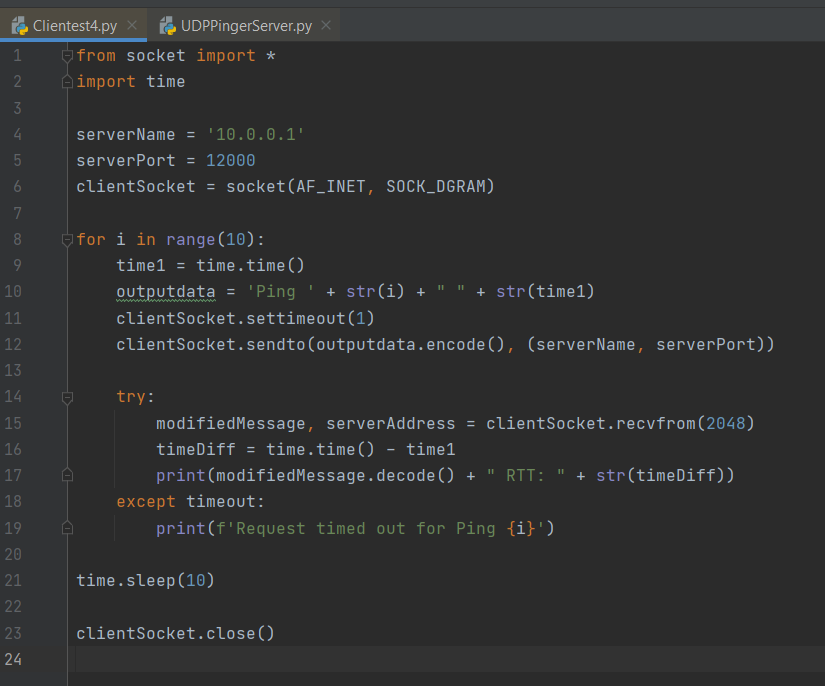
### **Despliegue en Mininet**

El despliegue es exactamente el mismo que el anterior, pero en vez de levantar en h1 **Server.py,** se debe levantar **Server\_thread.py** con el comando **python3 Server\_thread.py**

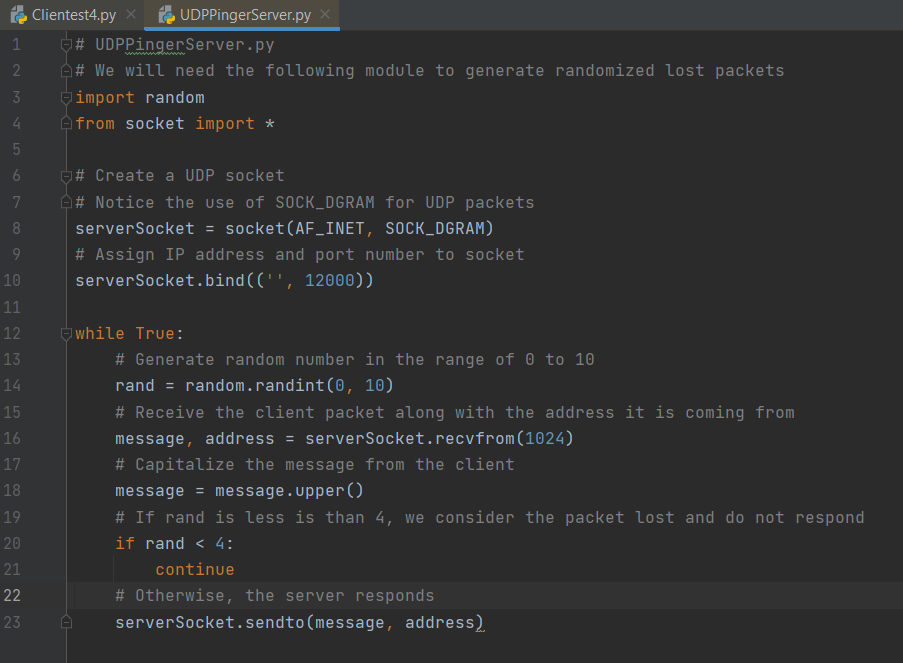
# **Assignment2**

### **Código**

**Cliente**



**Servidor**



### **Explicación**

**Cliente**

El primer código corresponde un cliente de ping simple que utiliza el protocolo UDP (User Datagram Protocol) y envía 10 pings a un servidor y registra el tiempo de ida y vuelta para cada ping. Si no se recibe una respuesta a un ping en 1 segundo, se imprime un mensaje indicando que la solicitud de ping ha superado el tiempo de espera:

**serverName = '10.0.0.1' y serverPort = 12000**: Estas líneas definen la dirección IP y el puerto del servidor al que el cliente se conectará.

**clientSocket = socket(AF\_INET, SOCK\_DGRAM)**: Aquí se crea un objeto de socket. AF\_INET se refiere a la familia de direcciones del protocolo de internet, mientras que SOCK\_DGRAM significa que es un socket UDP.

**for i in range(10):**: Este es un bucle que se ejecuta 10 veces. En cada iteración, el cliente envía un mensaje de ping al servidor y espera una respuesta.

**time1 = time.time()**: Aquí se registra el tiempo actual en segundos desde la época (normalmente 1 de enero de 1970).

**outputdata = 'Ping ' + str(i) + " " + str(time1)**: Aquí se prepara el mensaje de ping para enviar al servidor. El mensaje incluye el número de ping y el tiempo en que se envió el ping.

**clientSocket.settimeout(1)**: Aquí se establece un tiempo de espera de 1 segundo para las operaciones de socket. Si el socket no recibe una respuesta en 1 segundo, se lanza una excepción de tiempo de espera.

**clientSocket.sendto(outputdata.encode(), (serverName, serverPort))**: Aquí se envía el mensaje de ping al servidor.

**modifiedMessage, serverAddress = clientSocket.recvfrom(2048)**: Aquí el cliente intenta recibir un mensaje del servidor. Si no se recibe ningún mensaje en 1 segundo, se lanza una excepción de tiempo de espera.

**timeDiff = time.time() - time1**: Aquí se calcula el tiempo que ha pasado desde que se envió el ping hasta que se recibió la respuesta. Este es el tiempo de ida y vuelta (RTT).

**print(modifiedMessage.decode() + " RTT: " + str(timeDiff))**: Aquí se imprime el mensaje de respuesta del servidor y el RTT.

**print(f'Request timed out for Ping {i}')**: Si se produce una excepción de tiempo de espera, se imprime un mensaje indicando que la solicitud de ping ha superado el tiempo de espera.

**time.sleep(10)**: Aquí el programa se detiene durante 10 segundos.

**clientSocket.close()**: Finalmente, se cierra el socket del cliente.

**Servidor**

El segundo código corresponde a un servidor UDP simple que simula la pérdida de paquetes en una red, para lograr dicha simulación el servidor no responde a aproximadamente el 30% de los paquetes que recibe.

**serverSocket = socket(AF\_INET, SOCK\_DGRAM):** Aquí se crea un objeto de socket. AF\_INET se refiere a la familia de direcciones del protocolo de internet, mientras que SOCK\_DGRAM significa que es un socket UDP.

**serverSocket.bind(('', 12000))**: Aquí se vincula el socket del servidor a una dirección IP y un puerto. La dirección IP vacía ('') significa que el servidor escuchará en todas las interfaces de red disponibles.

**while True:**: Este es un bucle infinito donde el servidor sigue aceptando nuevas conexiones y procesándolas.

**rand = random.randint(0, 10)**: Aquí se genera un número aleatorio entre 0 y 10.

**message, address = serverSocket.recvfrom(1024)**: Aquí el servidor recibe un paquete del cliente. El paquete puede tener hasta 1024 bytes.

**message = message.upper()**: Aquí se convierte el mensaje del cliente a mayúsculas.

**if rand < 4:**: Aquí se verifica si el número aleatorio es menor que 4. Si es así, el servidor simula la pérdida de un paquete y no responde al cliente.

**serverSocket.sendto(message, address)**: Si el número aleatorio es 4 o mayor, el servidor responde al cliente. La respuesta es el mensaje del cliente convertido a mayúsculas.

### **Despliegue en Mininet**

Usando la topología expuesta en el assingment1 y siguiendo los mismos pasos (del 1 al 4), pero este caso se busca la ruta **Assingment2** y levanta el archivo Clientes4.py como cliente y el UDPPingerServer.py del lado del servidor. Al ejecutarlos queda de la siguiente manera:

