

PES UNIVERSITY
EC CAMPUS BANGALORE

NAME: R SHARMILA

SRN: PES2UG19CS309

DATE: 26-02-2021

SUBJECT: Computer Network Laboratory

WEEK: 5

TOPIC: Simple Client-Server Application using Network Socket Programming

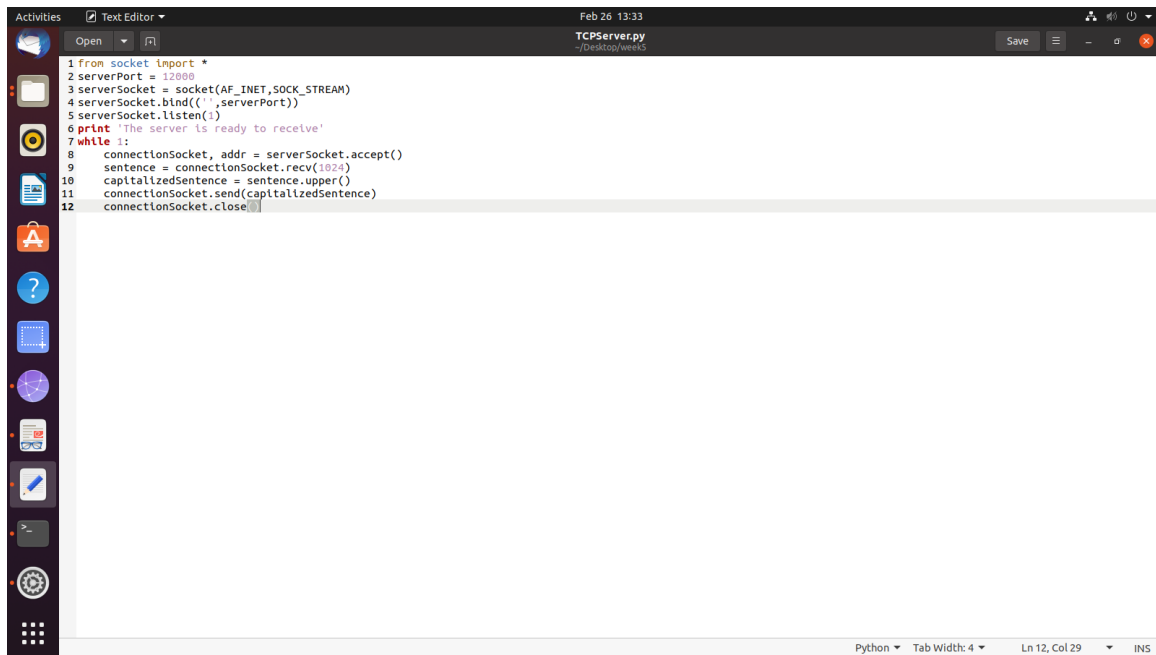
Task 1:

1. Create an application that will.
 - a. Convert lowercase letters to uppercase.
 - e.g. [a...z] to [A...Z]
 - code will not change any special characters, e.g., &*!
 - b. If the character is in uppercase, the program must not alter.
2. Create Socket API both for client and server.
3. Must take the server address and port from the Command Line Interface (CLI).

1.1 TCP Connection

- A TCP connection can be made between two machines with the help of a socket interface using the socket library on Python3.
- To create a TCP socket interface, the type of socket needs to be set as `SOCK_STREAM`.
- The type of addresses needs to be set as `AF_INET` which corresponds to IPv4.
- Once the server socket application is created, it needs to be hosted and hence needs to bind to a host IP and port number using the `bind ()` function.
- Similarly, the client socket application needs to connect to a host using the IP address and port number.
- The socket can now listen for incoming connections as well as send messages to connected host machines.

1.1.1 TCP Server

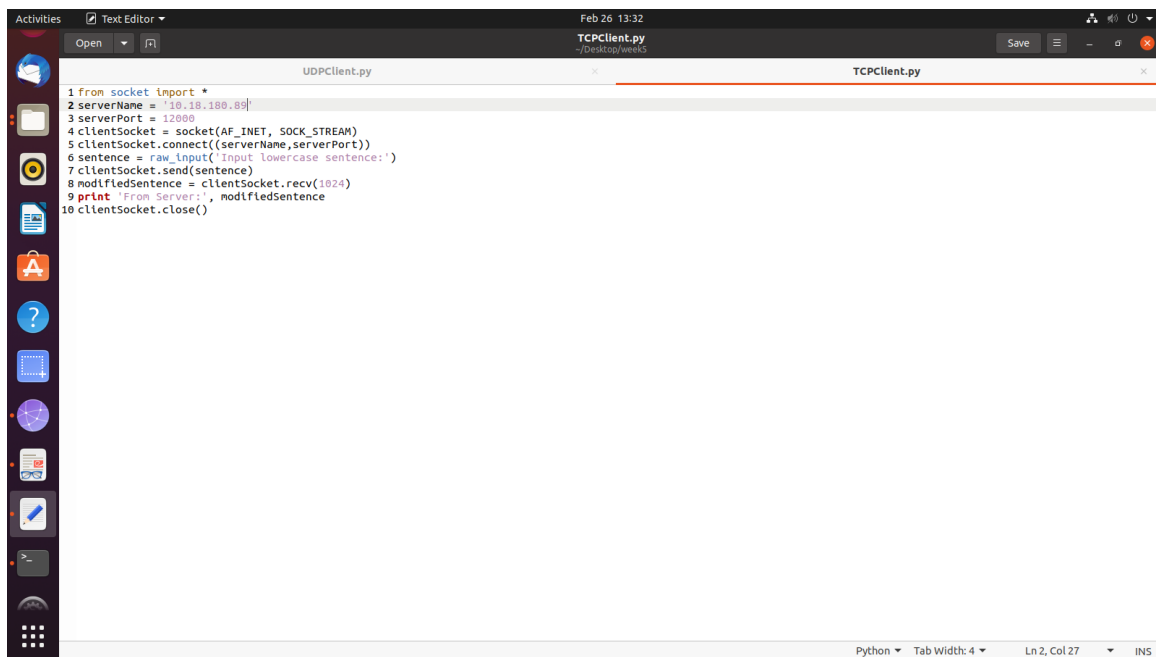


The screenshot shows a text editor window titled 'TCPServer.py' with the following Python code:

```
1 from socket import *
2 serverPort = 12000
3 serverSocket = socket(AF_INET, SOCK_STREAM)
4 serverSocket.bind(('', serverPort))
5 serverSocket.listen(1)
6 print 'The server is ready to receive'
7 while True:
8     connectionSocket, addr = serverSocket.accept()
9     sentence = connectionSocket.recv(1024)
10    capitalizedSentence = sentence.upper()
11    connectionSocket.send(capitalizedSentence)
12    connectionSocket.close()
```

The status bar at the bottom indicates 'Python', 'Tab Width: 4', 'Ln 12, Col 29', and 'INS'.

1.1.2 TCP Client

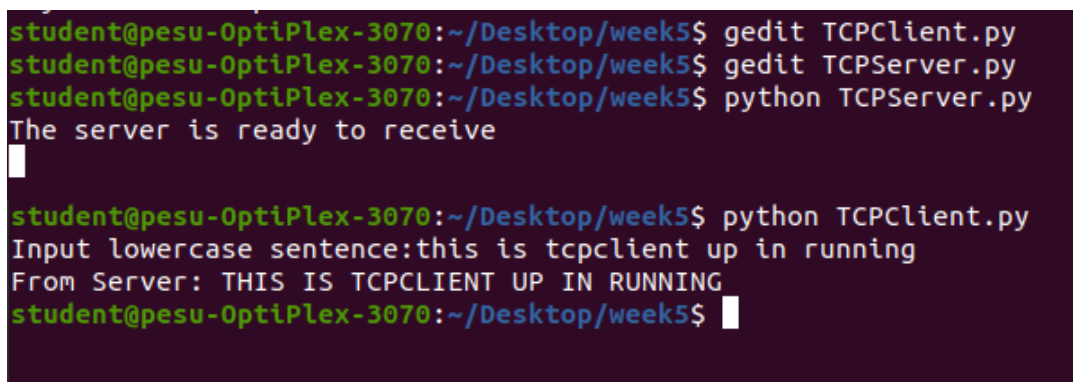


The screenshot shows a text editor window titled 'TCPClient.py' with the following Python code:

```
1 from socket import *
2 serverName = '10.18.180.89'
3 serverPort = 12000
4 clientSocket = socket(AF_INET, SOCK_STREAM)
5 clientSocket.connect((serverName, serverPort))
6 sentence = raw_input('Input lowercase sentence:')
7 clientSocket.send(sentence)
8 modifiedSentence = clientSocket.recv(1024)
9 print 'From Server:', modifiedSentence
10 clientSocket.close()
```

The status bar at the bottom indicates 'Python', 'Tab Width: 4', 'Ln 2, Col 27', and 'INS'.

1.1.3 TCP Connection between Server and Client

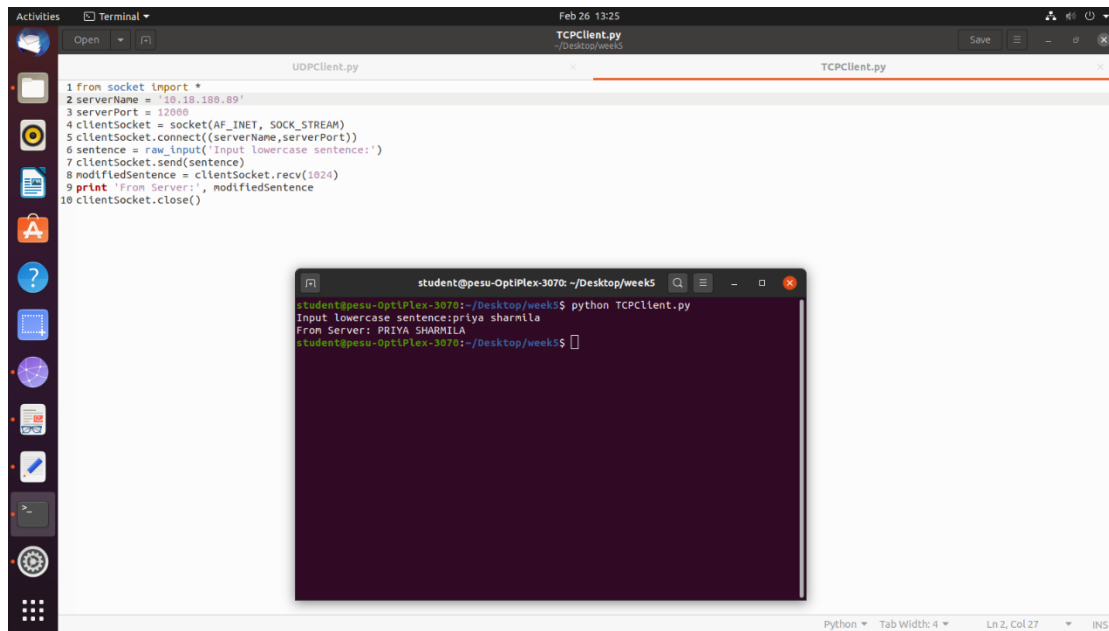


The screenshot shows a terminal window with the following commands and output:

```
student@pesu-OptiPlex-3070:~/Desktop/week5$ gedit TCPClient.py
student@pesu-OptiPlex-3070:~/Desktop/week5$ gedit TCPServer.py
student@pesu-OptiPlex-3070:~/Desktop/week5$ python TCPServer.py
The server is ready to receive

student@pesu-OptiPlex-3070:~/Desktop/week5$ python TCPClient.py
Input lowercase sentence:this is tcpclient up in running
From Server: THIS IS TCPCLIENT UP IN RUNNING
student@pesu-OptiPlex-3070:~/Desktop/week5$
```

When done using different system as client and server:



The screenshot shows a Linux desktop with a text editor window titled 'UDPClient.py' containing the following Python code:

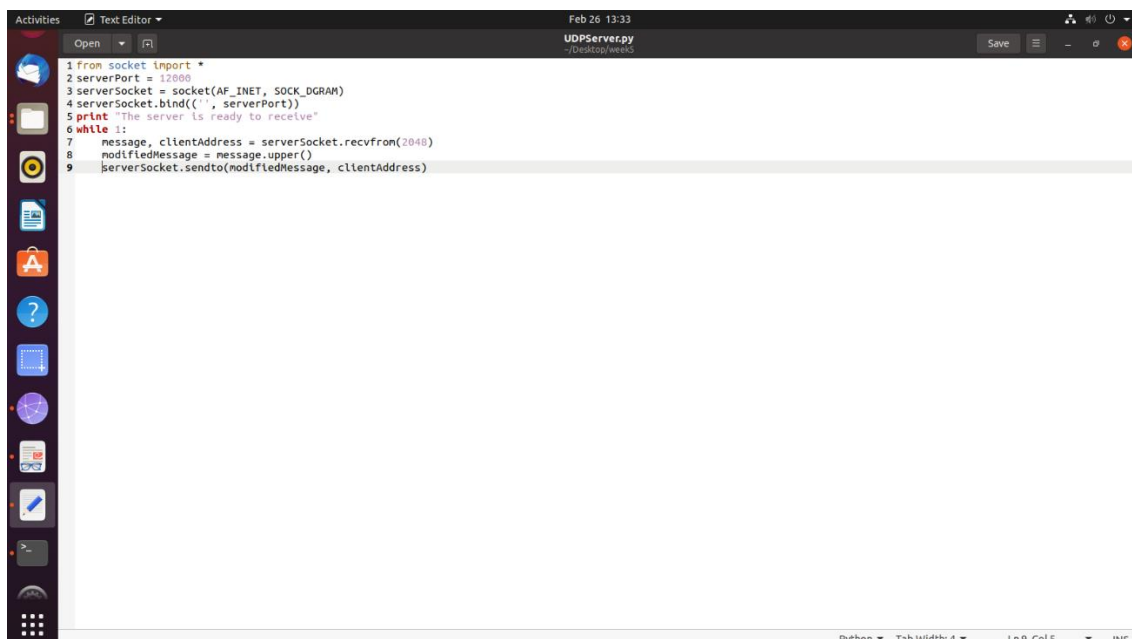
```
1 from socket import *
2 serverName = '10.10.100.89'
3 serverPort = 12000
4 clientSocket = socket(AF_INET, SOCK_STREAM)
5 clientSocket.connect((serverName, serverPort))
6 sentence = raw_input('Input lowercase sentence:')
7 clientSocket.send(sentence)
8 modifiedSentence = clientSocket.recv(1024)
9 print 'From Server:', modifiedSentence
10 clientSocket.close()
```

Below the text editor is a terminal window titled 'student@pesu-OptiPlex-3070: ~/Desktop/week5'. It shows the command 'python TCPClient.py' being executed, followed by the input 'Input lowercase sentence:priya sharmila' and the output 'From Server: PRIYA SHARMILA'.

1.2 UDP Connection

- A UDP connection can be made between two machines with the help of a socket interface using the socket library on Python3.
- To create a UDP socket interface, the type of socket needs to be set as `SOCK_DGRAM`.
- The type of addresses needs to be set as `AF_INET` which corresponds to IPv4.
- Once the server socket application is created, it needs to be hosted and hence needs to bind to a host IP and port number using the `bind ()` function.
- Similarly, the client socket application needs to connect to a host using the IP address and port number.
- The socket can now listen for incoming connections as well as send messages to connected host machines.

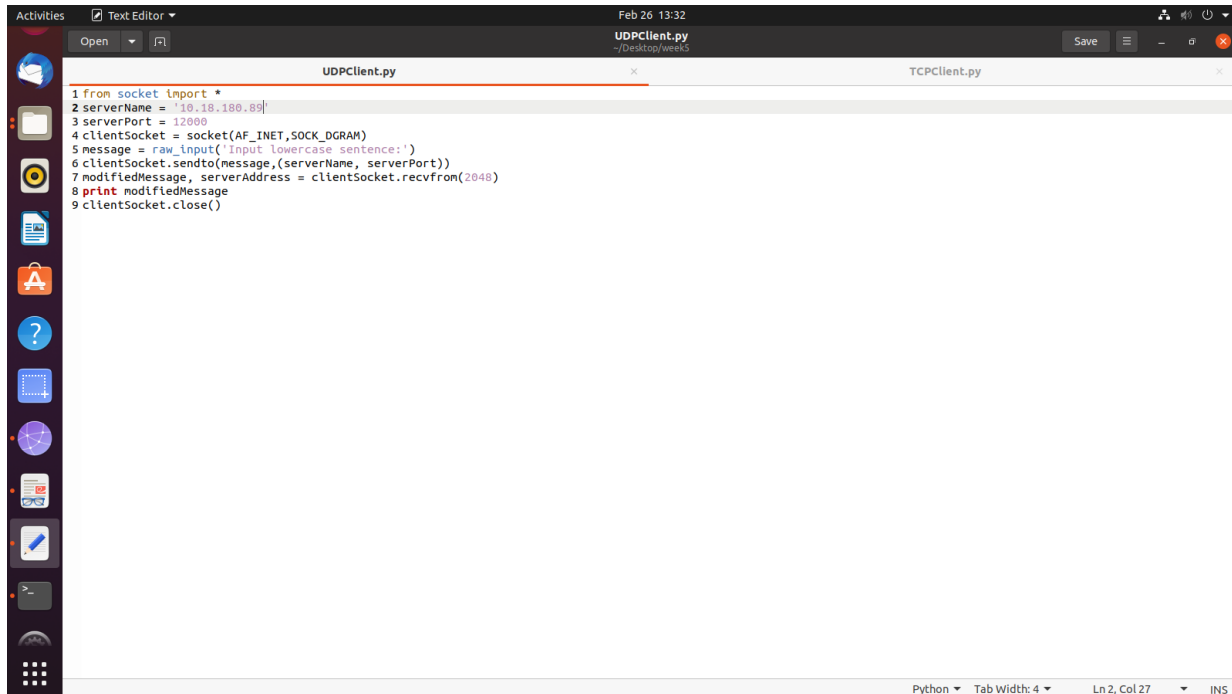
1.2.1 UDP Server



The screenshot shows a Linux desktop with a text editor window titled 'UDPServer.py' containing the following Python code:

```
1 from socket import *
2 serverPort = 12000
3 serverSocket = socket(AF_INET, SOCK_DGRAM)
4 serverSocket.bind(('', serverPort))
5 print 'The server is ready to receive'
6 while 1:
7     message, clientAddress = serverSocket.recvfrom(2048)
8     modifiedMessage = message.upper()
9     serverSocket.sendto(modifiedMessage, clientAddress)
```

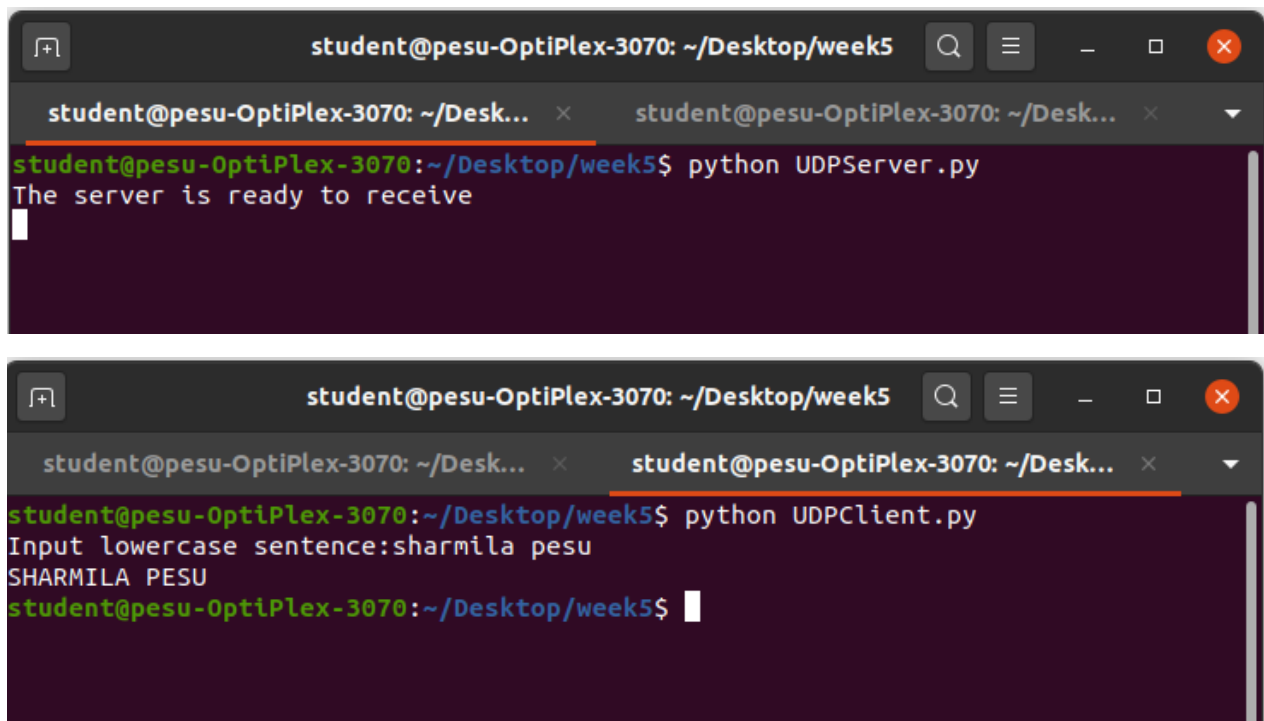
1.2.2 UDP Client



The screenshot shows a text editor window titled 'UDPClient.py' with the following code:

```
1 from socket import *
2 serverName = '10.18.180.89'
3 serverPort = 12000
4 clientSocket = socket(AF_INET,SOCK_DGRAM)
5 message = raw_input('Input lowercase sentence:')
6 clientSocket.sendto(message,(serverName, serverPort))
7 modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
8 print modifiedMessage
9 clientSocket.close()
```

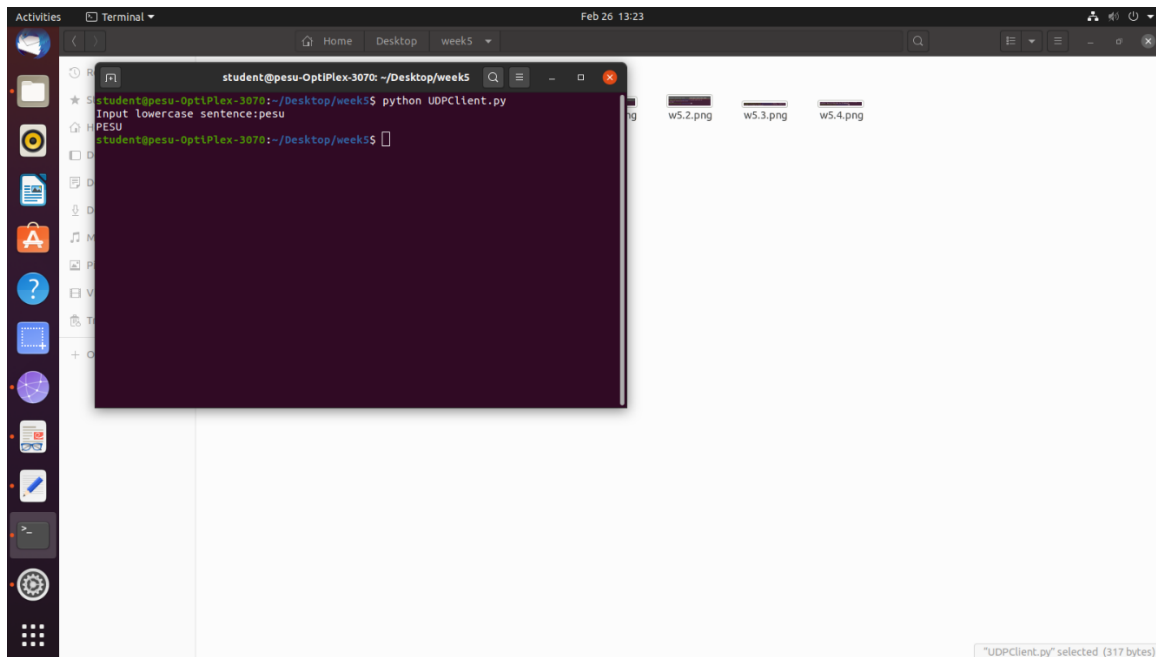
1.2.3 UDP Connection between Server and Client



The first screenshot shows a terminal window with the command `python UDPServer.py` being executed. The output is `The server is ready to receive`.

The second screenshot shows a terminal window with the command `python UDPClient.py` being executed. The user inputs `sharmila pesu`, and the output is `SHARMILA PESU`.

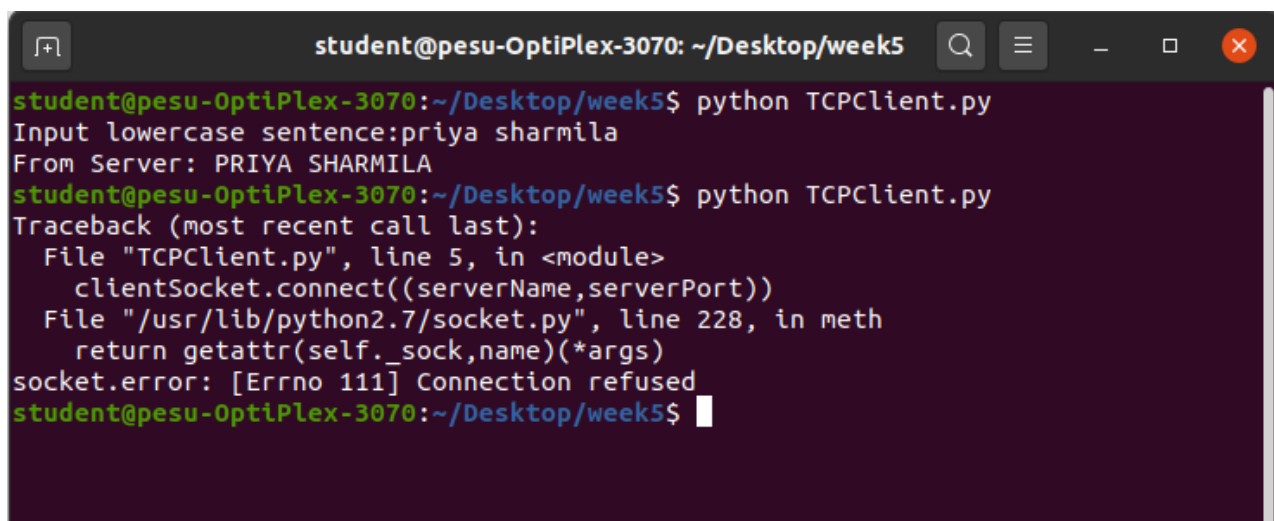
When done using different system as client and server:



1.2 Problems

1. Suppose you run TCP Client before you run TCP Server. What happens? Why?

This will lead to a Connection Refused Error, since the server socket application we are trying to connect to has not been initiated and is not listening for connections on the given port number. Hence, any connection requests sent by a client machine at that IP and port number immediately fail since the connection gets refused. A TCP connection can be established between two socket interfaces only when a host machine listens to requests on a given IP address and port number and accepts connections made by another machine at the same address and port.



2. Suppose you run UDP Client before you run UDP Server. What happens? Why?

No error will be obtained since UDP does not require a prior connection to be set up between the host machines for data transfer to begin. It is a connectionless protocol which transfers packets of data to a destination IP and port number without verifying the existence of the connection. Hence, it is prone to data integrity issues such as loss of packets. If any packets of data are sent before the server is executed, the packets are lost forever and will not reach the server socket application. However, if any packets of data are sent after the server is executed, the client will be able to send packets to a destination server and receive response packets in return.

3. What happens if you use different port numbers for the client and server sides?

This will lead to a Connection Refused Error for a TCP connection, since the server socket application we are trying to connect to is not listening for requests at the same port number as the one the client socket application is trying to connect with. Hence, the connection between the two socket interfaces is never setup and the connection are downright refused. However, on a UDP connection, since no prior connection is required to be established between the host machines for data transfer to take place, no error as such is obtained. Any messages sent by the client are lost since the destination server does not exist.

2. Task 2 –Web Server

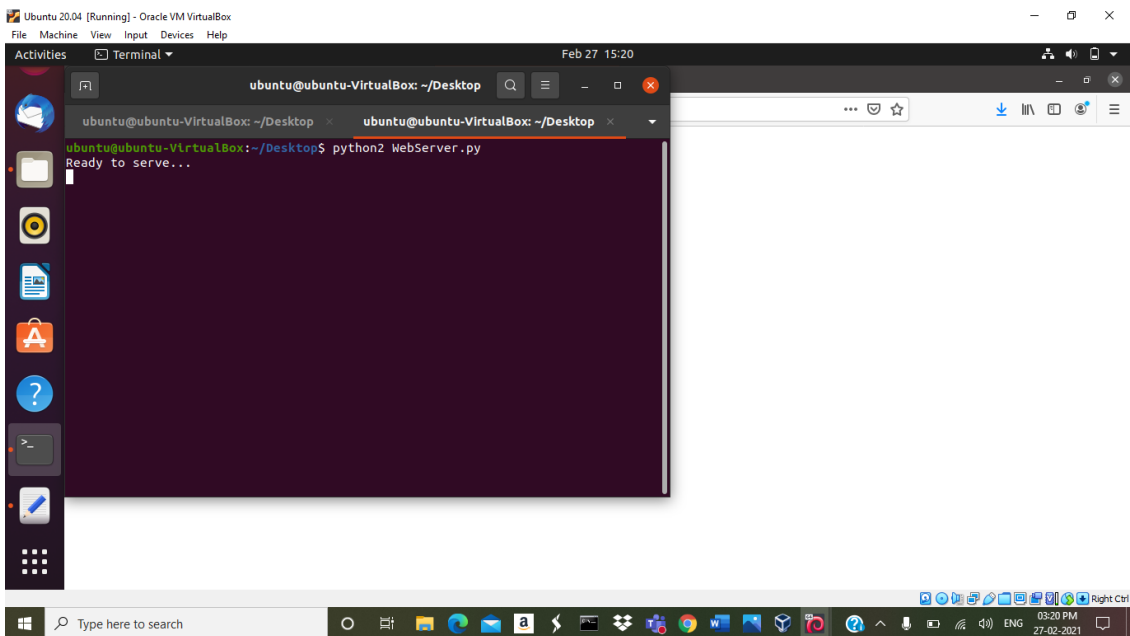
In this assignment, you will develop a simple Web server in Python that is capable of processing only one request. Specifically, your Web server will

- a) Create a connection socket when contacted by a client(browser)
- b) Receive the HTTP request from this connection.
- c) Parse the request to determine the specific file being requested.
- d) Get the requested file from the server's file system.
- e) Create an HTTP response message consisting of the requested file preceded by header lines.
- f) Send the response over the TCP connection to the requesting browser.

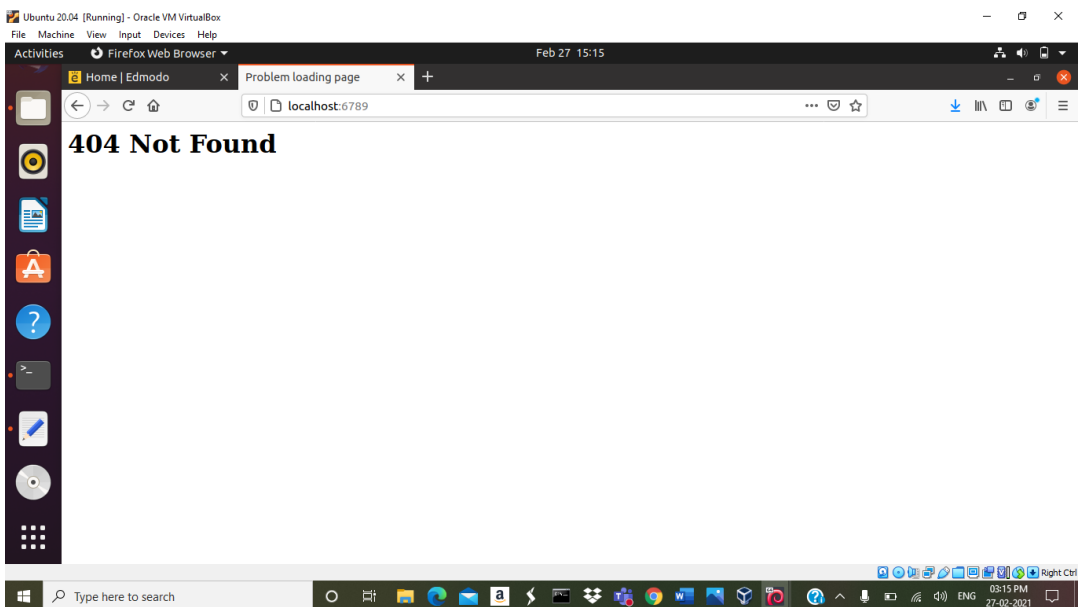
If a browser requests a file that is not present in your server, your server should return a “404 Not Found” error message.

Webserver.py

```
1 # Import socket module
2 from socket import *
3 # Create a TCP server socket
4 # (AF_INET is used for IPv4 protocols)
5 # (SOCK_STREAM is used for TCP)
6 serverSocket = socket(AF_INET, SOCK_STREAM)
7 # Assign a port number
8 serverPort = 6789
9 # Bind the socket to server address and server port
10 serverSocket.bind(('', serverPort))
11 # Listen to at most 1 connection at a time
12 serverSocket.listen(1)
13 # Server should be up and running and listening to the incoming connections
14 while True:
15     print 'Ready to serve...'
16     # Set up a new connection from the client
17     connectionSocket, addr = serverSocket.accept()
18     # If an exception occurs during the execution of try clause
19     # the rest of the clause is skipped
20     # If the exception type matches the word after except
21     # the except clause is executed
22     try:
23         # Receives the request message from the client
24         message = connectionSocket.recv(1024)
25         # Extract the path of the requested object from the message
26         # The path is the second part of HTTP header, identified by [1]
27         filename = message.split()[1]
28         # Because the extracted path of the HTTP request includes
29         # a character '\', we read the path from the second character
30         f = open(filename[1:])
31         # Store the entire content of the requested file in a temporary buffer
32         outputdata = f.read()
33         # Send the HTTP response header line to the connection socket
34         connectionSocket.send("HTTP/1.1 200 OK\r\n\r\n")
35         # Send the content of the requested file to the connection socket
36         for i in range(0, len(outputdata)):
37             connectionSocket.send(outputdata[i])
38         connectionSocket.send("\r\n")
39         # Close the client connection socket
40         connectionSocket.close()
41     except IOError:
42         # Send HTTP response message for file not found
43         connectionSocket.send("HTTP/1.1 404 Not Found\r\n\r\n")
44         connectionSocket.send("<html><head></head><body><h1>404 Not Found</h1></body></html>\r\n")
45         # Close the client connection socket
46         connectionSocket.close()
47 serverSocket.close()
```



We observe 404 NOT FOUND ERROR message here



Index.html

```
WebServer.py  index.html
1 <html>
2 <h1>
3 HELLO
4 </h1>
5 <body>
6 <h2>
7 I am Sharmila
8 </h2>
9 </body>
10 </html>
```

```
Processing triggers for desktop file utils (0.24-1ubuntu5) ...
ubuntu@ubuntu-VirtualBox:~/Desktop$ python2 WebServer.py
Ready to serve...
Ready to serve...
Ready to serve...
Ready to serve...
```

HTML code seen on the web browser

