COMPUTER NETWORKS LAB
NAME: TRISHA JAIN
SRN: PES1UG19CS542
SECTION: I

TASK 1 :-

Creating an application that will convert lowercase letters to uppercase letters using socket programming.

Socket Programming with UDP :-

A UDP connection is made by using the socket library on Python.
The type of socket needs to be set as SOCK_DGRAM and the type of address needs to be set as AF_INET(IPv4).
The client and server side of the application is created, and the server side is connected to the client using the bind() function.

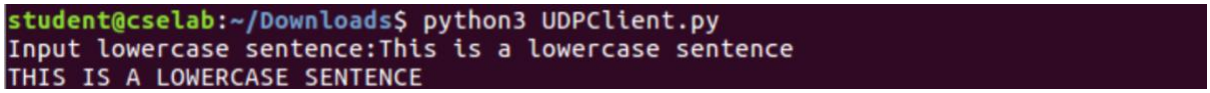
UDP SERVER :-

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_DGRAM)
serverSocket.bind(('',serverPort))
print("The server is ready to receive")
while 1:
message, clientAddress = serverSocket.recvfrom(2048)
modifiedMessage = message.upper()
serverSocket.sendto(modifiedMessage, clientAddress)
```

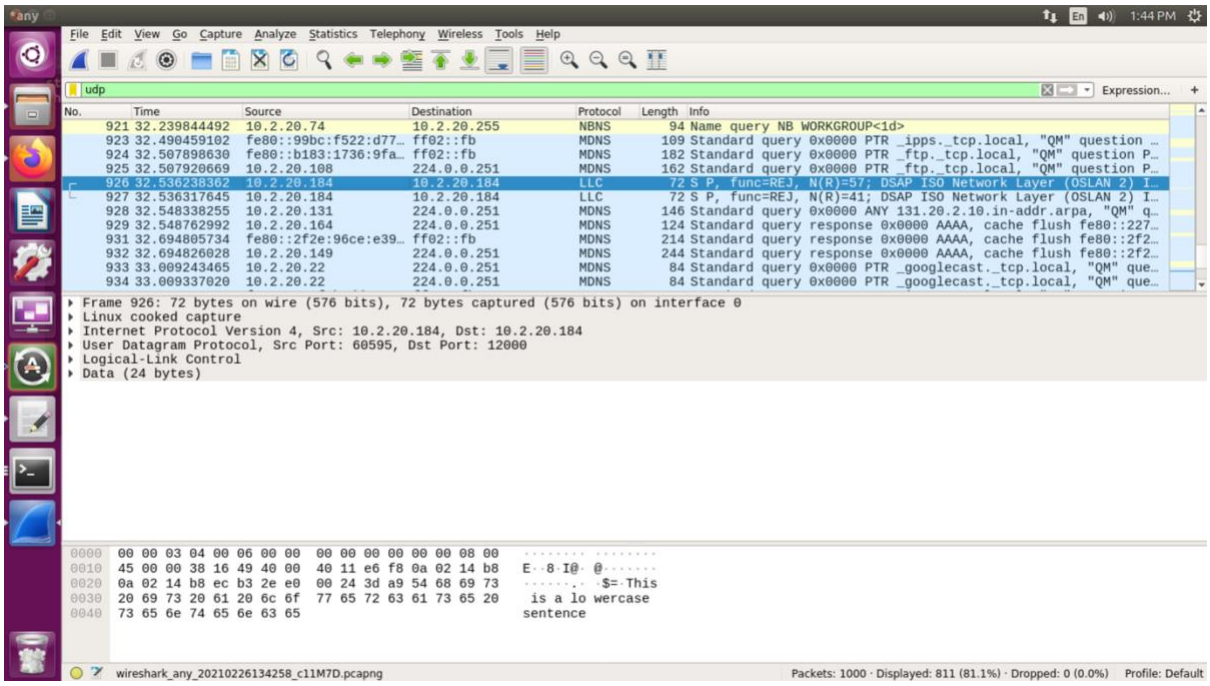
UDP CLIENT :-

```
from socket import *
serverName = '10.0.2.15'
serverPort = 12000
clientSocket = socket(AF_INET,SOCK_DGRAM)
message = input('Input lowercase sentence:')
clientSocket.sendto(message.encode(),(serverName,serverPort))
modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
print(modifiedMessage.decode())
clientSocket.close()
```

UDP Server file running on terminal



UDP Client file running on terminal



Wireshark capture of UDP packets

Socket Programming with UDP :-

A TCP connection is made by using the socket library on Python.
The type of socket needs to be set as SOCK_STREAM and the type of address needs to be set as AF_INET(IPv4).
The client and server side of the application is created, and the server side is connected to the client using the bind() function.

TCP SERVER :-

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(('',serverPort))
serverSocket.listen(1)
print('The server is ready to receive')
while True:
connectionSocket, addr = serverSocket.accept()
sentence = connectionSocket.recv(1024).decode()
capitalizedSentence = sentence.upper()
connectionSocket.send(capitalizedSentence.encode())
connectionSocket.close()
```

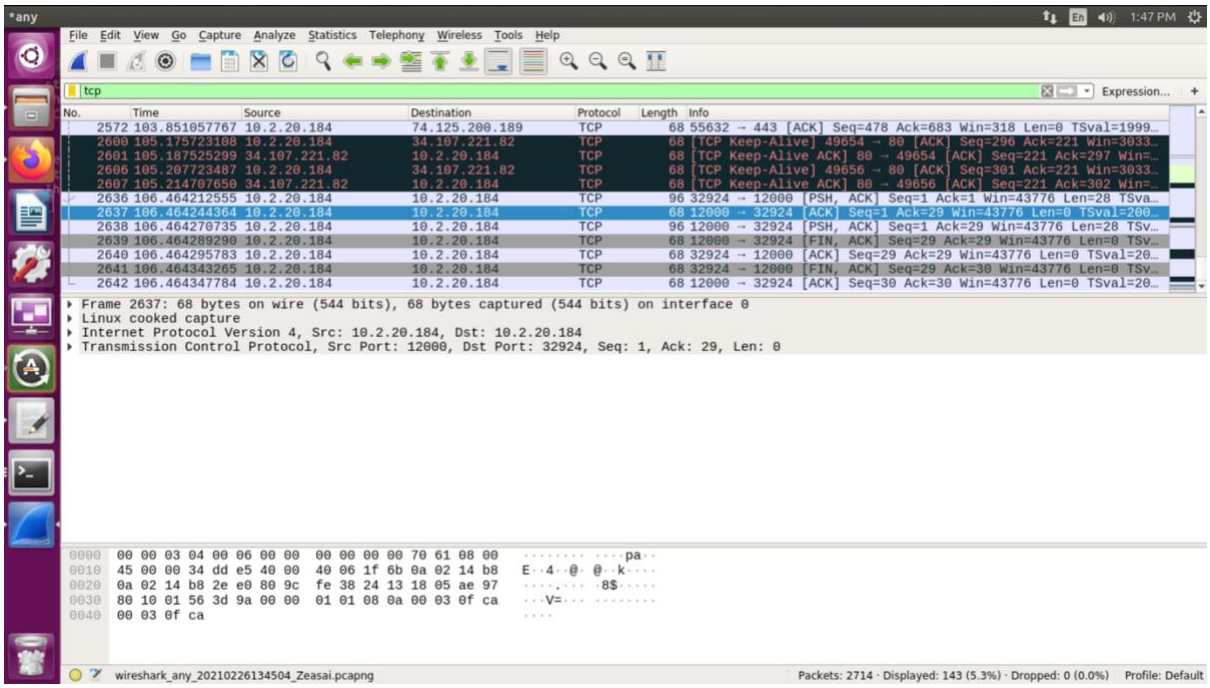
TCP CLIENT :-

```
from socket import *
serverName = '10.0.2.15'
serverPort = 12000
clientSocket = socket(AF_INET,SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
sentence = input('Input lowercase sentence:')
clientSocket.send(sentence.encode())
modifiedSentence = clientSocket.recv(1024)
print('From Server:',modifiedSentence.decode())
clientSocket.close()
```

TCP Server file running on terminal

```
student@cselab:~/Downloads$ python3 TCPClient.py
Input lowercase sentence:This is a lowercase sentence
From Server: THIS IS A LOWERCASE SENTENCE
student@cselab:~/Downloads$
```

TCP Client file running on terminal



Wireshark Capture of TCP packets

PROBLEMS :-

Q1. Suppose you run TCPClient before you run TCPServer. What happens? Why?

A1. If we run the TCPClient before we run TCPServer we will get a ConnectionRefusedError. This happens because the server application is not listening to any connections on the given port since it’s not initiated.

```
student@cselab: ~/Downloads
student@cselab:~/Downloads$ python3 TCPClient.py
Traceback (most recent call last):
  File "TCPClient.py", line 7, in <module>
    clientSocket.connect((serverName,serverPort))
ConnectionRefusedError: [Errno 111] Connection refused
student@cselab:~/Downloads$
```

Q2. Suppose you run UDPClient before you run UDPServer. What happens? Why?

A2. Unlike TCP, in the case of UDP there will be no error since UDP is a connectionless protocol. It transfers packets to destination host without checking if a connection exists. So, the packets that are sent before the server is started are lost forever.

```
student@cselab:~/Downloads$ python3 UDPClient.py
Input lowercase sentence:lower case sentence
```

Q3. What happens if you use different port numbers for the client and server sides?

A3. In the case of TCP we will get a connection refused error like in the case when the client was run before the server file. And in the case of UDP we will not get any error since it is a connectionless protocol. But the application doesn't work as expected since the packets are lost due to unsuccessful connection.

```
student@cselab: ~/Downloads
student@cselab:~/Downloads$ python3 UDPServer.py
The server is ready to receive

student@cselab: ~/Downloads
student@cselab:~/Downloads$ python3 UDPClient.py
Input lowercase sentence:lower case sentence
```

UDP

```
student@cselab:~/Downloads$ python3 TCPServer.py
The server is ready to receive
```

```
student@cselab:~/Downloads$ python3 TCPClient.py
Traceback (most recent call last):
  File "TCPClient.py", line 7, in <module>
    clientSocket.connect((serverName,serverPort))
ConnectionRefusedError: [Errno 111] Connection refused
student@cselab:~/Downloads$
```

TCP

TASK 2 :-

A simple web server is developed in Python that is capable of processing only on request.

Steps performed by the Web Server Application :-

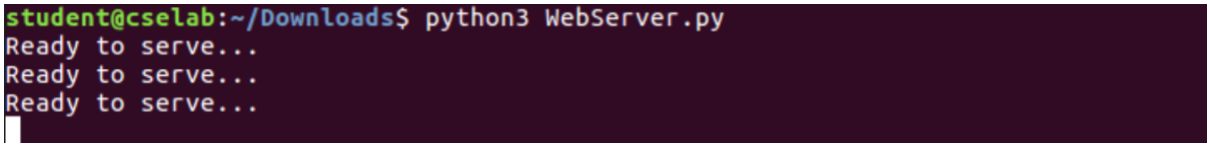
1) Create a connection socket when contacted by a client.
2) Receive the HTTP request from the connection.
3) Parse the request to determine the specific file being requested.
4) Get the requested file from the server's file system.
5) Create an HTTP response message consisting of the requested file preceded by header lines.
6) Send the response over the TCP connection to the requesting browser.
7) If the browser requests a file that is not present in the server, the server returns a 404 not found error message.

Web Server Program :-

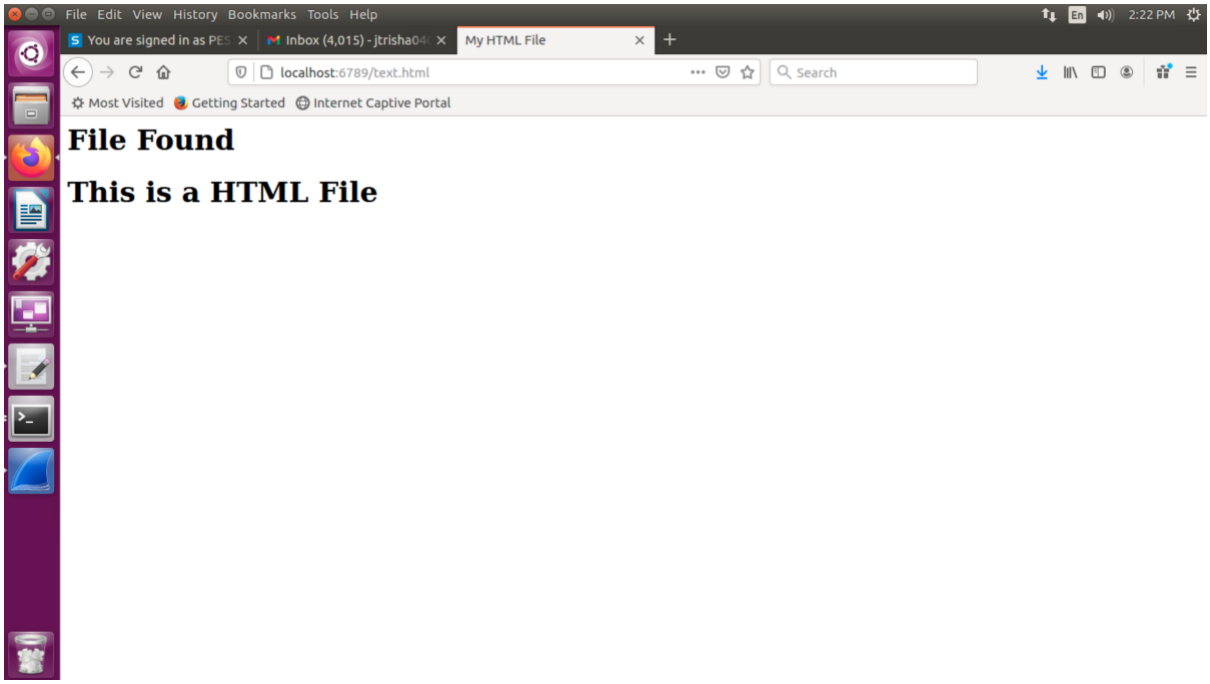
```
 # Import socket module
from socket import *

# Create a TCP server socket
#(AF_INET is used for IPv4 protocols)
#(SOCK_STREAM is used for TCP)

serverSocket = socket(AF_INET, SOCK_STREAM)

# Assign a port number
serverPort = 6789

# Bind the socket to server address and server port
serverSocket.bind(("", serverPort))
```

```
# Listen to at most 1 connection at a time
serverSocket.listen(1)

# Server should be up and running and listening to the incoming connections
while True:
print('Ready to serve...')

# Set up a new connection from the client
connectionSocket, addr = serverSocket.accept()

# If an exception occurs during the execution of try clause
# the rest of the clause is skipped
# If the exception type matches the word after except
# the except clause is executed
try:
# Receives the request message from the client
message =  connectionSocket.recv(1024).decode()
# Extract the path of the requested object from the message
# The path is the second part of HTTP header, identified by [1]
filename = message.split()[1]
# Because the extracted path of the HTTP request includes
# a character '\', we read the path from the second character
f = open(filename[1:])
# Store the entire contenet of the requested file in a temporary buffer
outputdata = f.read()
# Send the HTTP response header line to the connection socket
messages = "HTTP/1.1 200 OK\r\n\r\n"
connectionSocket.send(messages.encode())
messages = "<html><head></head><body><h1>File
Found</h1></body></html>\r\n"
connectionSocket.send(messages.encode())


# Send the content of the requested file to the connection socket
for i in range(0, len(outputdata)):
connectionSocket.send(outputdata[i].encode())
messages = "\r\n"
connectionSocket.send(messages.encode())

# Close the client connection socket
```

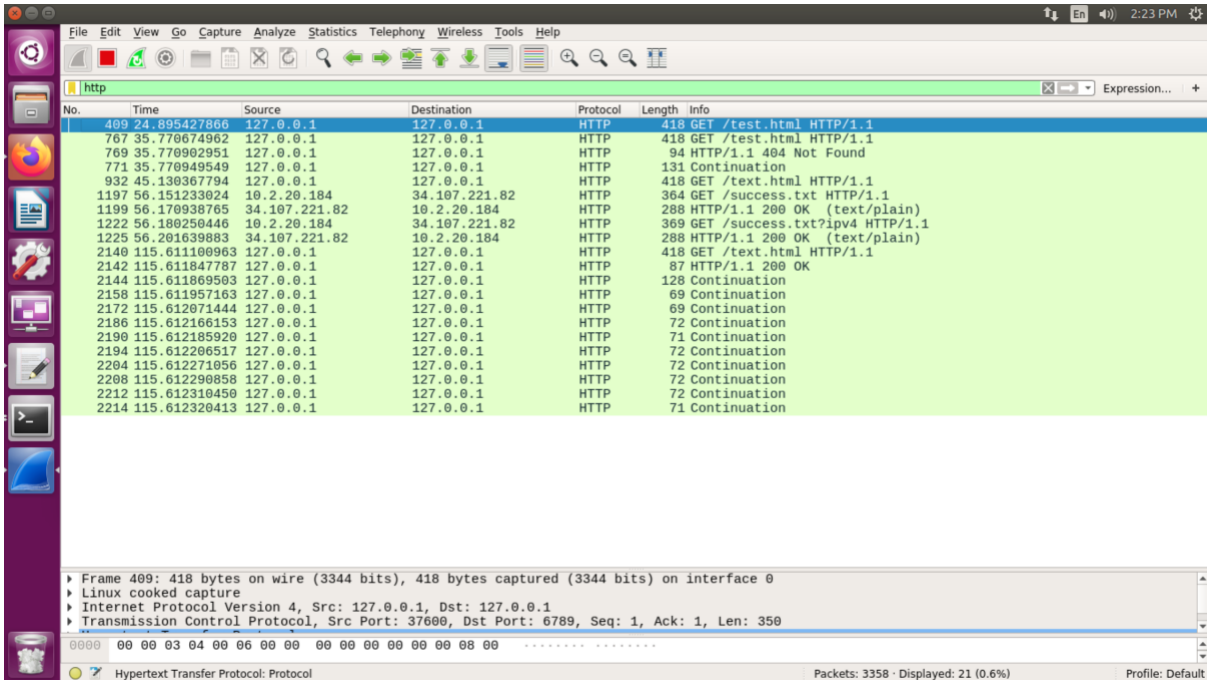
```
connectionSocket.close()

except IOError:
# Send HTTP response message for file not found
messages = "HTTP/1.1 404 Not Found\r\n\r\n"
connectionSocket.send(messages.encode())
messages = "<html><head></head><body><h1>404 Not
Found</h1></body></html>\r\n"
connectionSocket.send(messages.encode())
# Close the client connection socket
connectionSocket.close()

serverSocket.close()
```



WebServer file running on terminal



Web Browser displaying the file requested



Wireshark Packet Capture