CSE 476
MOBILE COMMUNICATION NETWORKS

MIDTERM PROJECT

PROGRESS REPORT

ÖMER ÇEVİK
161044004

1. Assignment 1: Web Server

In this assignment, main idea is to create a web server which serves an HTML file. I used the skeleton of web server which is shared as a lab of Web Server for this assignment and filled the empty places.

First of all, creating a socket in Python programming language is in **socket[11]** library and using **socket(…)[13]** returns a socket. To use TCP socket we give parameter **SOCK_STREAM[13]** to socket function. We use Web Server so it needs internet connection we use **AF_INET[13]** for socket parameter as address from the internet.

After server socket is created then we create the **HOST** and **PORT** which are represented ***"0.0.0.0"*[18]** as host address and ***"6789"*[19]** as port number. Then socket is **bind**ing**[20]** to host with its port number and starting to **listen[21]** socket. Server is ready for serve**[28]** right now infinitely**[25]**. Then server **accept[29]** the requests, creates connection socket**[29]** and reads message from clients using **recv[32]** function on that connection socket. Clients requests the file name of example HTML file. In try catch block**[31, 50]** that happens. Because if the file is not exist**[35]** then we **send[54]** client ***"404 Not Found"*** information message on socket and closing connection socket**[61]**. If it is exist then it reads whole HTML file**[37]**, **send**ing **200 HTTP Success OK Code[42]** to client and outputs to client the HTML file using socket's **send** function**[47, 48]**. Then it closes connection socket**[49]** and closes server socket**[64]**.

Notes:

- Code lines in WebServer.py are represented inside bold square brackets.
- Python 2.7.18 version is used to run code.
- Ubuntu 20.04.1 LTS operating system and its Terminal is workspace.
- To run code ***python WebServer.py*** command is realized in Terminal.
- No optional exercises are implemented in Web Server part.

WebServer.py :

```python
1.  """
2.  CSE 476 Mobile Communication Networks
3.  Midterm Project
4.  Web Server
5.
6.  WebServer.py
7.  @author Omer CEVIK
8.  161044004
9.  """
10. #import socket module
11. from socket import *
12.
13. serverSocket = socket(AF_INET, SOCK_STREAM)
14.
15. #Prepare a server socket
16. #Fill in start
17.
18. HOST = '0.0.0.0'
19. PORT = 6789
20. serverSocket.bind((HOST, PORT))
21. serverSocket.listen(1)
22.
23. #Fill in end
24.
25. while True:
26.
27.     #Establish the connection
28.     print('Ready to serve...')
29.     connectionSocket, addr = serverSocket.accept()
30.
31.     try:
32.         message = connectionSocket.recv(1024)
33.
34.         filename = message.split()[1]
35.         f = open(filename[1:])
36.
37.         outputdata = f.read()
38.
39.         #Send one HTTP header line into socket
40.         #Fill in start
41.
42.         connectionSocket.send('HTTP/1.1 200 OK\r\n\r\n'.encode())
43.
44.         #Fill in end
45.
46.         #Send the content of the requested file to the client
47.         for i in range(0, len(outputdata)):
48.             connectionSocket.send(outputdata[i])
49.         connectionSocket.close()
50.     except IOError:
51.         #Send response message for file not found
52.         #Fill in start
53.
54.         connectionSocket.send('404 Not Found')
55.
56.         #Fill in end
57.
58.         #Close client socket
59.         #Fill in start
60.
61.         connectionSocket.close()
62.
63.         #Fill in end
64. serverSocket.close()
```

Web Server Program Test Results :
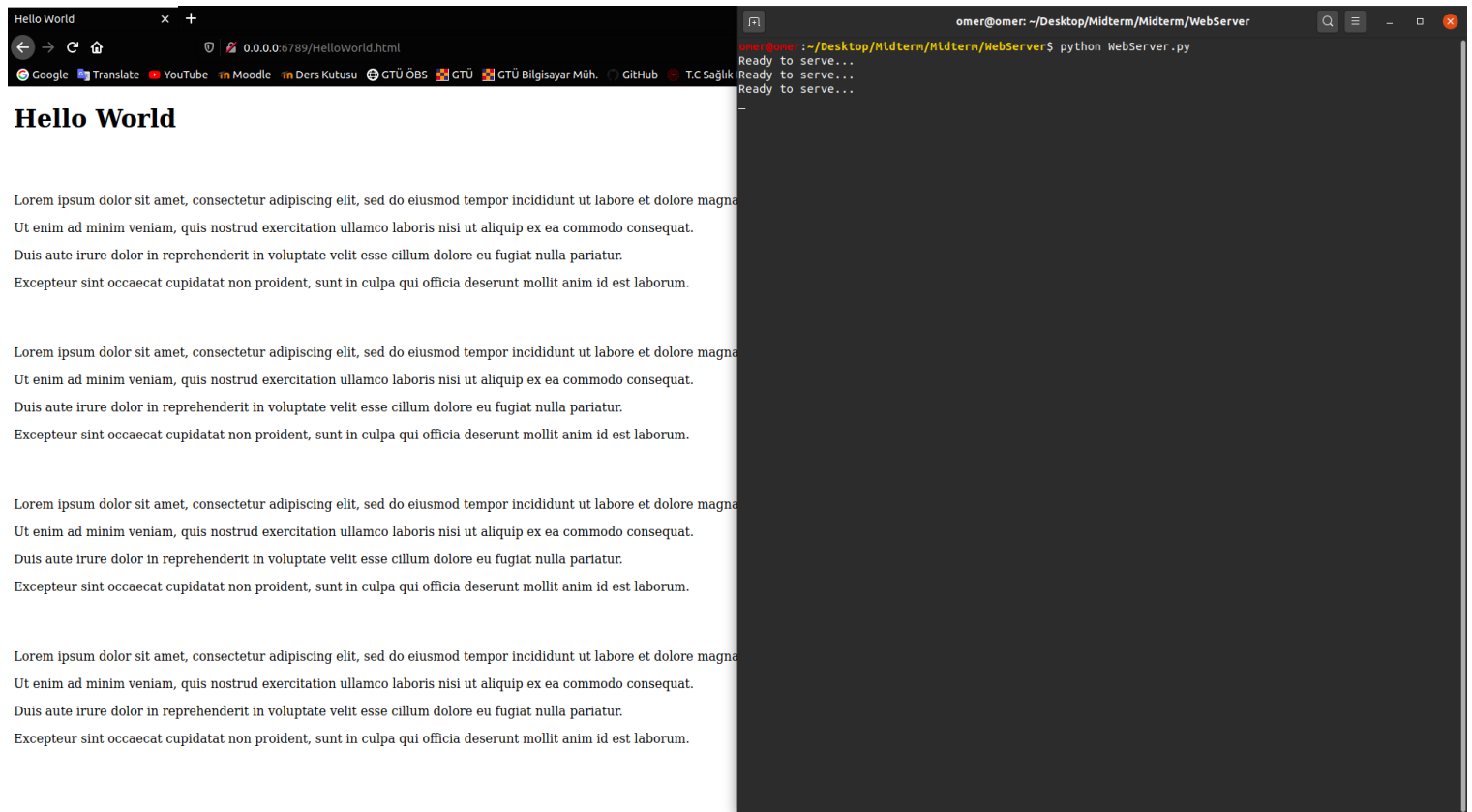

1. HelloWorld.html File

2. Running Web Server and Browser Client Result
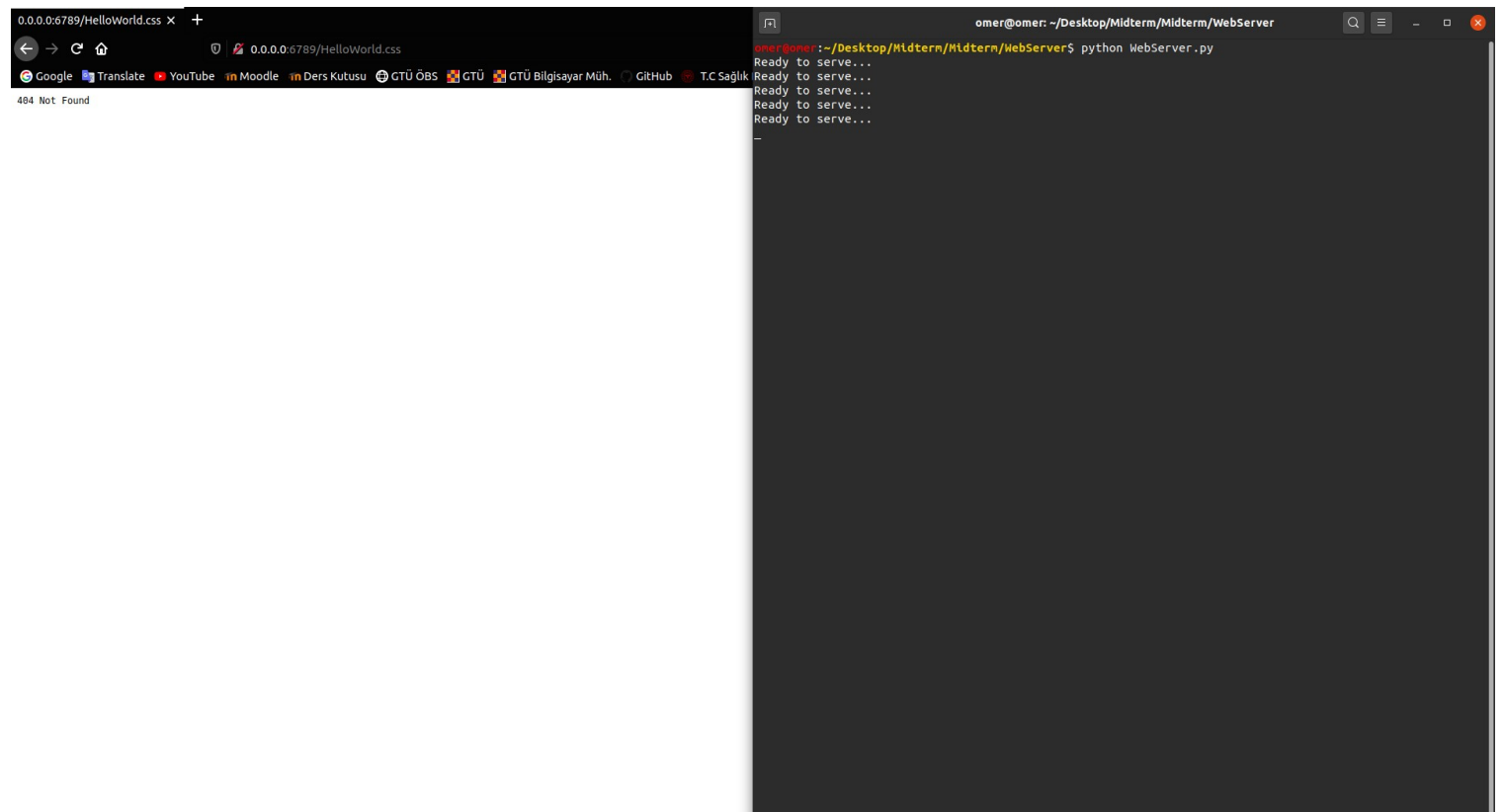
3. Running Web Server and Browser Client Requests HelloWorld.css File

2. Assignment 2: UDP Pinger

In this assignment, main idea is to create a server-client relation using UDP between that processes. UDPPingerServer.py is attached to assignment file as lab work which creates random integer values and if the value is less then 4 it keeps creating random values**[31]**. Otherwise, server reads the message**[26]** from socket which is hosted in *"127.0.0.1"* in *"12000"* port number and makes the message upper case each character**[20]**. If random value is **not** less than 4 then it sends the upper cased message to client which is connected by socket**[34]**. To make the protocol as UDP is provided using **SOCK_DGRAM** while creating server's socket in UDPPingerServer.py**[18]**.

In client side which is inside UDPPingerClient.py, host address and port number are set to *"127.0.0.1"***[17]**, *"12000"***[18]**. To read from socket buffer, initialized the buffer size as 1024**[19]**. The ping message is declared as **"Ping Message "[23]** and it is encoded**[24]**. For request to server 10 times is declared using a message counter**[27]**. To create a socket using UDP is provided using **SOCK_DGRAM** in client side too**[30]**. If server doesn't response in 1 second, the socket is set the time out for 1 second using **settimeout(1)** function of socket**[33]**. Then for each message request to server is applied using **sendto()** function of socket passing parameter as encoded ping message**[42]**. While doing that, using **time** library it starts to count **RTT** time**[39]** before sending ping message then tries to get **recvfrom** server**[46]**. If the time out is realized then it will be catched in **except[55]** and printing the message **"Request timed out"[57]**. If everything is fine, then it gets the end of **RTT** time**[49]** and distinct of end and starting time of RTT represents the message's RTT time**[50]**. After evaluating RTT time, printing the message which is upper cased and received by server, message counter and distinct RTT time**[53, 54]**. Then counting the messages for 10 times**[60]**. Finally, closing UDP socket**[63]**.

Notes:

- Code lines in UDPPingerServer.py and UDPPingerClient.py are represented inside bold square brackets.
- Python 2.7.18 version is used to run code.
- Ubuntu 20.04.1 LTS operating system and its Terminal is workspace.
- To run UDPPingerServer code *python UDPPingerServer.py* command is realized in Terminal.
- To run UDPPingerClient code *python UDPPingerClient.py* command is realized in another Terminal.
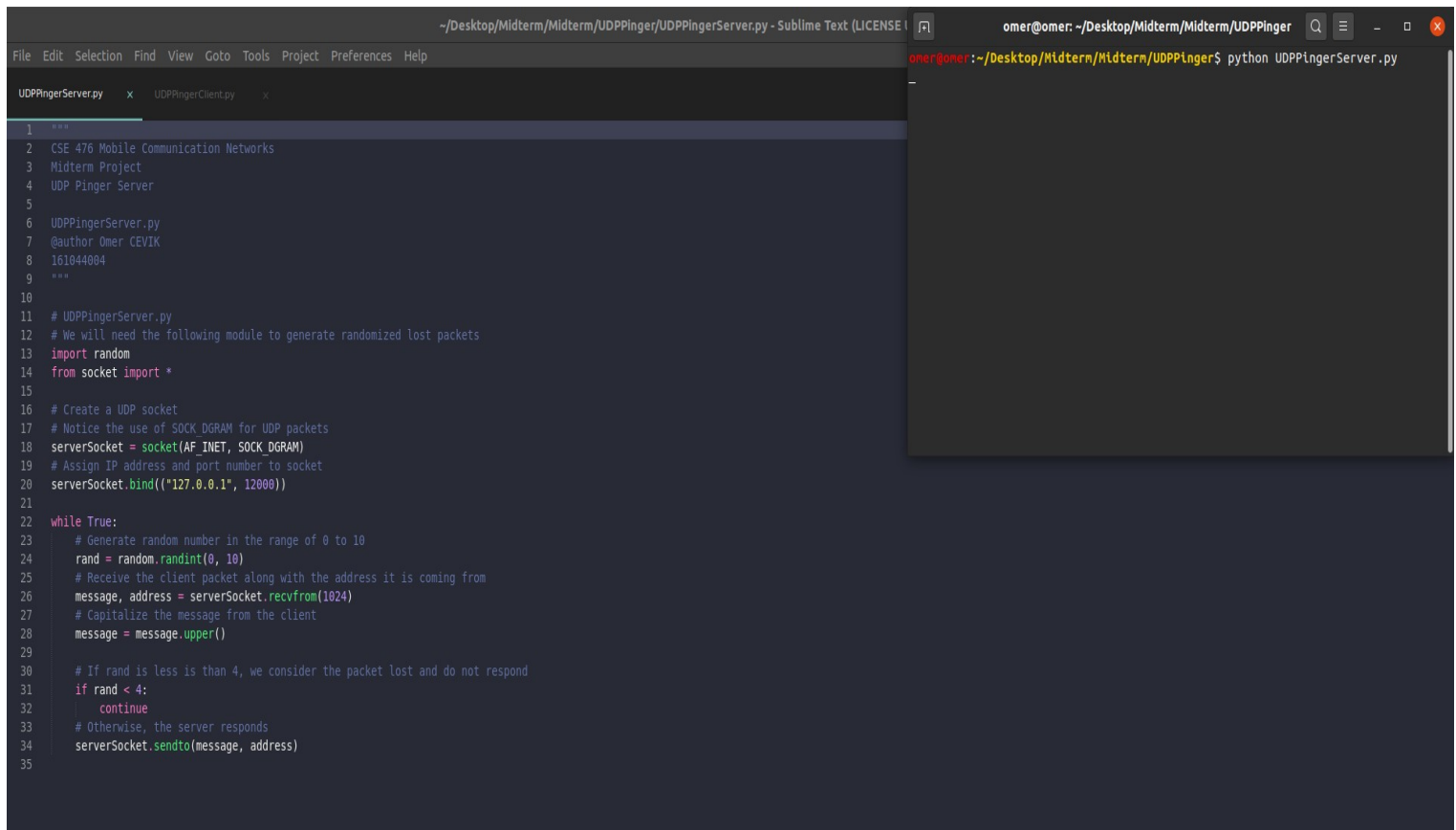- No optional exercises are implemented in UDP Pinger Server/Client part.
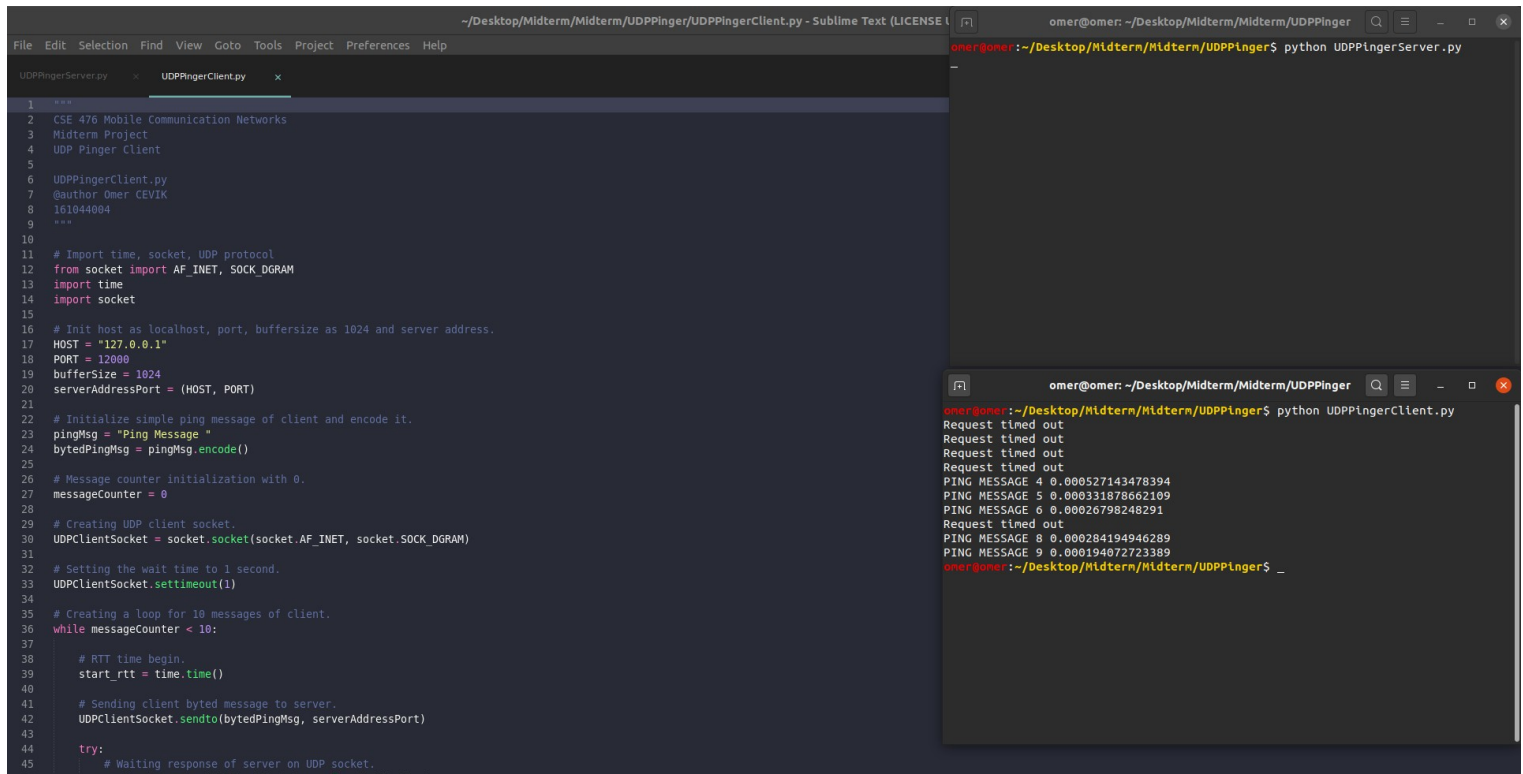
UDPPingerServer.py :

```python
1. """
2. CSE 476 Mobile Communication Networks
3. Midterm Project
4. UDP Pinger Server
5.
6. UDPPingerServer.py
7. @author Omer CEVIK
8. 161044004
9. """
10.
11. # UDPPingerServer.py
12. # We will need the following module to generate randomized lost packets
13. import random
14. from socket import *
15.
16. # Create a UDP socket
17. # Notice the use of SOCK_DGRAM for UDP packets
18. serverSocket = socket(AF_INET, SOCK_DGRAM)
19. # Assign IP address and port number to socket
20. serverSocket.bind(("127.0.0.1", 12000))
21.
22. while True:
23.     # Generate random number in the range of 0 to 10
24.     rand = random.randint(0, 10)
25.     # Receive the client packet along with the address it is coming from
26.     message, address = serverSocket.recvfrom(1024)
27.     # Capitalize the message from the client
28.     message = message.upper()
29.
30.    # If rand is less is than 4, we consider the packet lost and do not respond
31.     if rand < 4:
32.         continue
33.     # Otherwise, the server responds
34.     serverSocket.sendto(message, address)
35.
36. # Closing socket.
37. serverSocket.close()
```

UDPPingerClient.py :

```python
1. """
2. CSE 476 Mobile Communication Networks
3. Midterm Project
4. UDP Pinger Client
5.
6. UDPPingerClient.py
7. @author Omer CEVIK
8. 161044004
9. """
10.
11. # Import time, socket, UDP protocol
12. from socket import AF_INET, SOCK_DGRAM
13. import time
14. import socket
15.
16. # Init host as localhost, port, buffersize as 1024 and server address.
17. HOST = "127.0.0.1"
18. PORT = 12000
19. bufferSize = 1024
20. serverAddressPort = (HOST, PORT)
21.
22. # Initialize simple ping message of client and encode it.
23. pingMsg = "Ping Message "
24. bytedPingMsg = pingMsg.encode()
25.
26. # Message counter initialization with 1.
27. messageCounter = 1
28.
29. # Creating UDP client socket.
30. UDPClientSocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
31.
32. # Setting the wait time to 1 second.
33. UDPClientSocket.settimeout(1)
34.
35. # Creating a loop for 10 messages of client.
36. while messageCounter <= 10:
37.
38.     # RTT time begin.
39.     start_rtt = time.time()
40.
41.     # Sending client byted message to server.
42.     UDPClientSocket.sendto(bytedPingMsg, serverAddressPort)
43.
44.     try:
45.         # Waiting response of server on UDP socket.
46.         msgFromServer = UDPClientSocket.recvfrom(bufferSize)
47.
48.         # RTT time end and distinct of start-end time of message trip.
49.         end_rtt = time.time()
50.         distinct_rtt = end_rtt - start_rtt
51.
52.         # Printing the message which is upper cased by server, message counter and
                RTT time.
53.         msg = str(msgFromServer[0]) + str(messageCounter) + " " + str(distinct_rtt)
54.         print(msg)
55.     except socket.timeout:
56.         # If time out is generated about 1 second then printing the information.
57.         print("Request timed out")
58.
59.     # Increasing message counter.
60.     messageCounter += 1
61.
62. # Closing socket.
63. UDPClientSocket.close()
```

UDPPingerServer and UDPPingerClient Program Test Results :

1. Running UDPPingerServer.py in Terminal

2. Running UDPPingerClient.py in Another Terminal

3. Running UDPPingerClient.py in Another Terminal Again

4. Running UDPPingerClient.py While Server is Closed

3. Assignment 3: Mail Client

In this assignment, main idea is to create a client mail program which use **SMTP** to connect a mail server using socket.

Firstly, to send a message content the message is declared**[16]** and to represent the end of message it is declared**[17]**. Then the mail server's address and port number are declared (**gmail** server is used in this example using port number as **587**)**[22, 23]**. With **TCP** client socket is created**[29]** and **connect**ed**[32]** to mail server. Waiting the connection result using **recv[34]** on socket and **decode[35]** it about receive from server is applied. Then printing the receive result from server**[36]**. If result starts with '**220**'**[37]** it means client is not received by server**[38]**.

After connection succeed, **HELO** command is **sent** to server**[44, 45]** and its response is got using **recv[47, 48, 49]** function on client socket. If receive from server starts with '**250**' it means client is not reply received by server**[51, 52]**.

If **HELO** command is succeed then **STARTTLS** command is **sent** to server using **encoding[59]** and getting receive from server using **recv[60]** on socket. Printing the receive from socket is applied**[61]**.

After starting TLS, using **ssl** library socket is wrapped to **SSL** socket using ssl's **SSLv23** protocol version**[64]**. If mail server is live then it has to send to server **HELO** command**[67, 68]**. Otherwise it continues to **AUTH LOGIN** command sending and receiving by server with printing message of received command**[71, 72, 73, 74]**.

After authenticate login command, it needs user name and password (mail address and password) which is declared and encoded using **base64** library**[77, 78, 81, 82]**. Then **send**ing encoded user name and password with printing receive from socket**[85, 86, 87, 90, 91, 92]**.

Then it sends **MAIL FROM** command to socket with user name while **encoding** and printing receive from socket**[95, 96, 97, 98]**.

RCPT TO command is used to **send**ing**[111, 112]** mail to whom using mail address. In that example sending mail to itself**[107]**. You can try other mail address which is commented line**[108]**. RCPT TO command is **decode**d and printed the receive from socket**[113, 114]**.

DATA command is **sent** to socket and printed the receive message from socket**[123, 124, 125, 126]**. Then message and end of message is **sent** to socket using **encode** and printed the receive from socket**[135, 143, 145, 146]**.

After all **QUIT** command is **sent** to socket using **encode** and printing the receive from socket**[154, 156, 157, 158]**.

Finally, closing the wrapped socket and client socket successfully**[161, 162]**.

Notes:

- Code lines in MailClient.py is represented inside bold square brackets.
- Python 2.7.18 version is used to run code.
- Ubuntu 20.04.1 LTS operating system and its Terminal is workspace.
- To run MailClient code *python MailClient.py* command is realized in Terminal.

- First optional exercise is implemented in Mail Client part (SSL).

MailClient.py :

```
1.  """
2.  CSE 476 Mobile Communication Networks
3.  Midterm Project
4.  Mail Client
5.
6.  MailClient.py
7.  @author Omer CEVIK
8.  161044004
9.  """
10.
11. from socket import *
12. import base64
13. import sys
14. import ssl
15.
16. msg = "\r\n I love computer networks!"
17. endmsg = "\r\n.\r\n"
18.
19. # Choose a mail server (e.g. Google mail server) and call it mailserver
20.
21. # Choose mailserver as live or gmail.
22. # mailserver = ("smtp.live.com", 587)
23. mailserver = ("smtp.gmail.com", 587)
24.
25. # Create socket called clientSocket and establish a TCP connection with mailserver
26. #Fill in start
27.
28. # Creating socket TCP protocoled.
29. clientSocket = socket(AF_INET, SOCK_STREAM)
30.
31. # Connecting to mail server and printing the response.
32. clientSocket.connect(mailserver)
33.
34. recvConnect = clientSocket.recv(1024)
35. recvConnect = recvConnect.decode()
36. print("RECV CONNECT : " + recvConnect)
37. if recvConnect[:3] != '220':
38.     print('220 reply not received from server.')
39.
40. #Fill in end
41.
42.
43. # Send HELO command and print server response.
44. heloCommand = 'HELO Alice\r\n'
45. clientSocket.send(heloCommand.encode())
46.
47. recvHelloCommand = clientSocket.recv(1024)
48. recvHelloCommand = recvHelloCommand.decode()
49. print("RECV HELLO COMMAND : " + recvHelloCommand)
50.
51. if recvHelloCommand[:3] != '250':
52.     print('250 reply not received from server.')
53.
54. # Send MAIL FROM command and print server response.
55. # Fill in start
56.
57. # Start TLS sending STARTTLS command and printing the response.
58. startTlsMsg = 'STARTTLS\r\n'
59. clientSocket.send(startTlsMsg.encode())
60. recvTls = clientSocket.recv(1024)
61. print("RECV STARTTLS : " + recvTls.decode())
62.
63. # Wraps the client socket using SSL.
```

```python
64. wrappedClientSocket = ssl.wrap_socket(clientSocket, ssl_version=ssl.PROTOCOL_SSLv23)
65.
66. # If hotmail/outlook/live server is used then it sends hello command again.
67. if "live" in mailserver[0]:
68.     wrappedClientSocket.send(heloCommand.encode())
69.
70. # Sending wrapped socket AUTH LOGIN command.
71. authLoginMsg = 'AUTH LOGIN\r\n'
72. wrappedClientSocket.send(authLoginMsg.encode())
73. recvAuthLogin = wrappedClientSocket.recv(1024)
74. print("RECV AUTH LOGIN : " + recvAuthLogin.decode())
75.
76. # Test mail username and password.
77. username = "omerceviktest@gmail.com"
78. password = "KjhdveB7CMqwH7r"
79.
80. # Encode the username and password.
81. encodedUserName = (base64.b64encode(username.encode()))+('\r\n').encode()
82. encodedPassword = (base64.b64encode(password.encode()))+('\r\n').encode()
83.
84. # Sending encoded username and printing the response.
85. wrappedClientSocket.send(encodedUserName)
86. recvUserName = wrappedClientSocket.recv(1024)
87. print("RECV USER NAME : " + recvUserName.decode())
88.
89. # Sending encoded password and printing the response.
90. wrappedClientSocket.send(encodedPassword)
91. recvPassword = wrappedClientSocket.recv(1024)
92. print("RECV PASSWORD : " + recvPassword.decode())
93.
94. # Sending MAIL FROM command with username and printing the response.
95. mailFromMsg = "MAIL FROM: <"+ username +">\r\n"
96. wrappedClientSocket.send(mailFromMsg.encode())
97. recvMailFrom = wrappedClientSocket.recv(1024)
98. print("RECV MAIL FROM : " + recvMailFrom.decode())
99.
100.# Fill in end
101.
102.
103.# Send RCPT TO command and print server response.
104.# Fill in start
105.
106.# Using rcpt to mail with itself or other one.
107.rcptToMail = username
108.# rcptToMail = "omer.cevik2016@gtu.edu.tr"
109.
110.# Sending RCPT TO command with rcpt to mail and printing the response.
111.rcptToMsg = "RCPT TO: <"+ rcptToMail +">\r\n"
112.wrappedClientSocket.send(rcptToMsg.encode())
113.recvRcptTo = wrappedClientSocket.recv(1024)
114.print("RECV RCPT TO : " + recvRcptTo.decode())
115.
116.# Fill in end
117.
118.
119.# Send DATA command and print server response.
120.# Fill in start
121.
122.# Sending DATA command and printing the response.
123.dataMsg = "DATA\r\n"
124.wrappedClientSocket.send(dataMsg.encode())
125.recvData = wrappedClientSocket.recv(1024)
126.print("RECV DATA : " + recvData.decode())
127.
128.# Fill in end
129.
130.
131.# Send message data.
132.# Fill in start
```

```
133.
134.# Sending message.
135.wrappedClientSocket.send(msg.encode())
136.
137.# Fill in end
138.
139.# Message ends with a single period.
140.# Fill in start
141.
142.# Sending end of message and printing the response.
143.wrappedClientSocket.send(endmsg.encode())
144.
145.recvMsg = wrappedClientSocket.recv(1024)
146.print("RECV MSG : " + recvMsg.decode())
147.
148.# Fill in end
149.
150.# Send QUIT command and get server response.
151.# Fill in start
152.
153.# Sending QUIT command and printing the response.
154.quitMsg = "QUIT\r\n"
155.
156.wrappedClientSocket.send(quitMsg.encode())
157.recvQuit = wrappedClientSocket.recv(1024)
158.print("RECV QUIT : " + recvQuit.decode())
159.
160.# Closing wrapper socket and client socket.
161.wrappedClientSocket.close()
162.clientSocket.close()
163.
164.# Fill in end
```

MailClient Program Test Results :

1. Running MailClient.py on Terminal
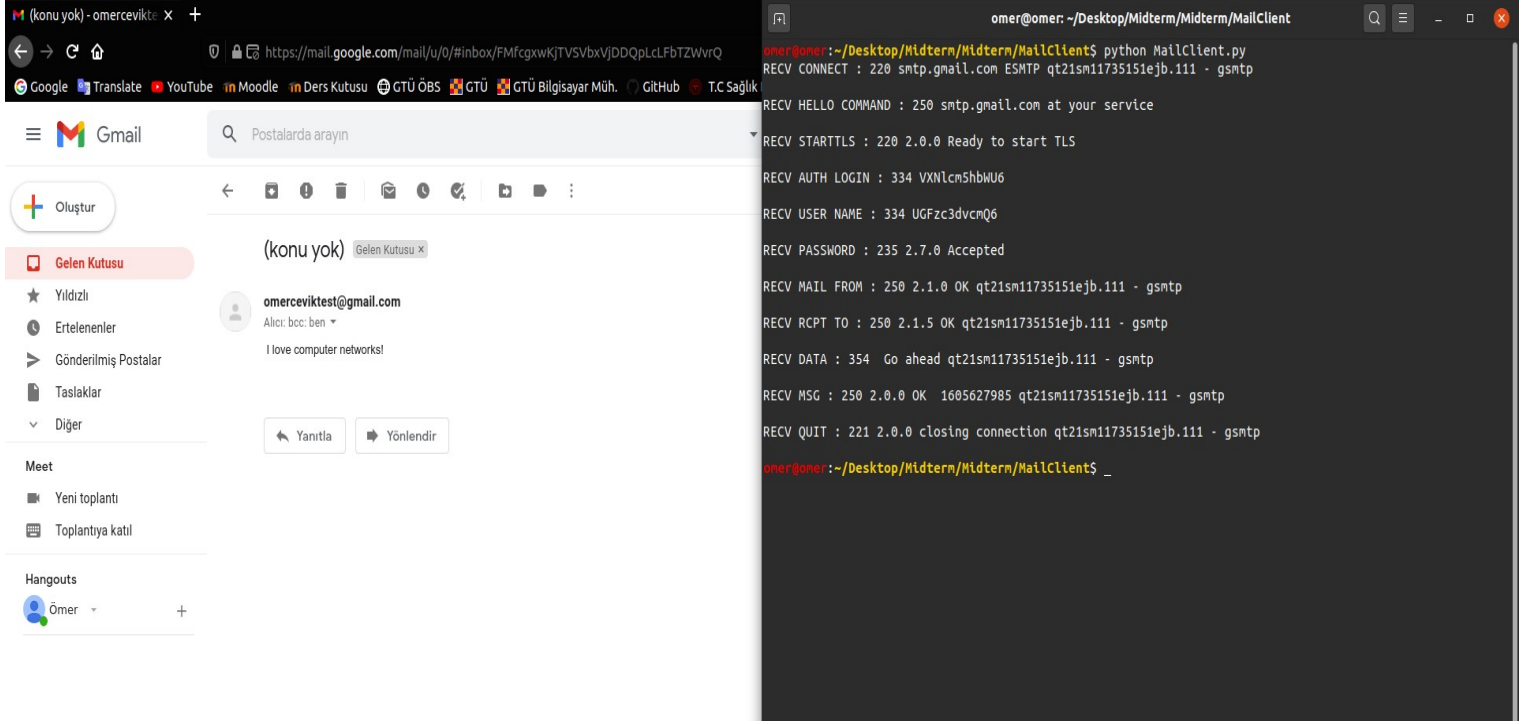
2. After Running Program on Gmail
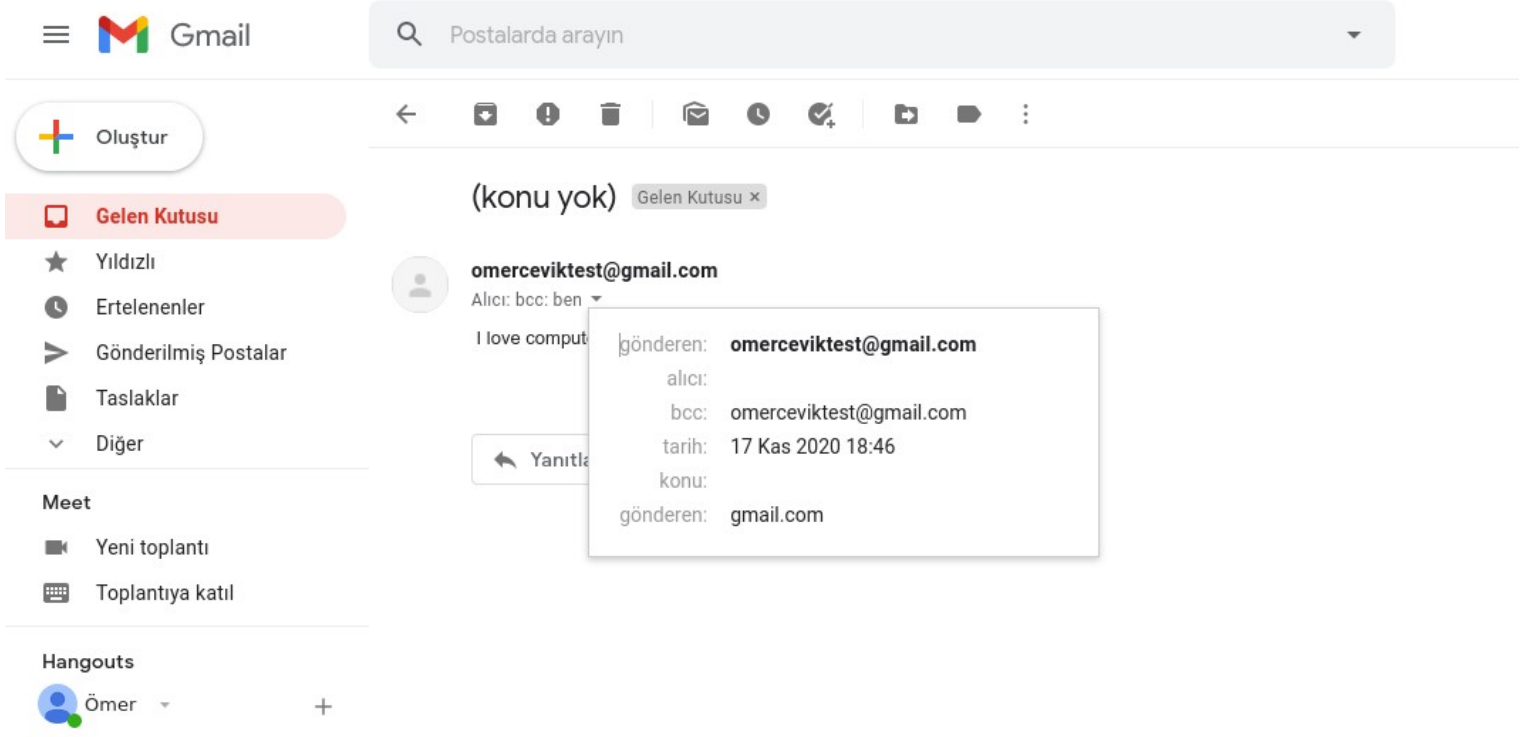
3. After Running Program Sender and Reciever on Gmail