Gebze Technical University
Computer Engineering
CSE 476 / CSE 575
Fall 2020 Term Project Assignments

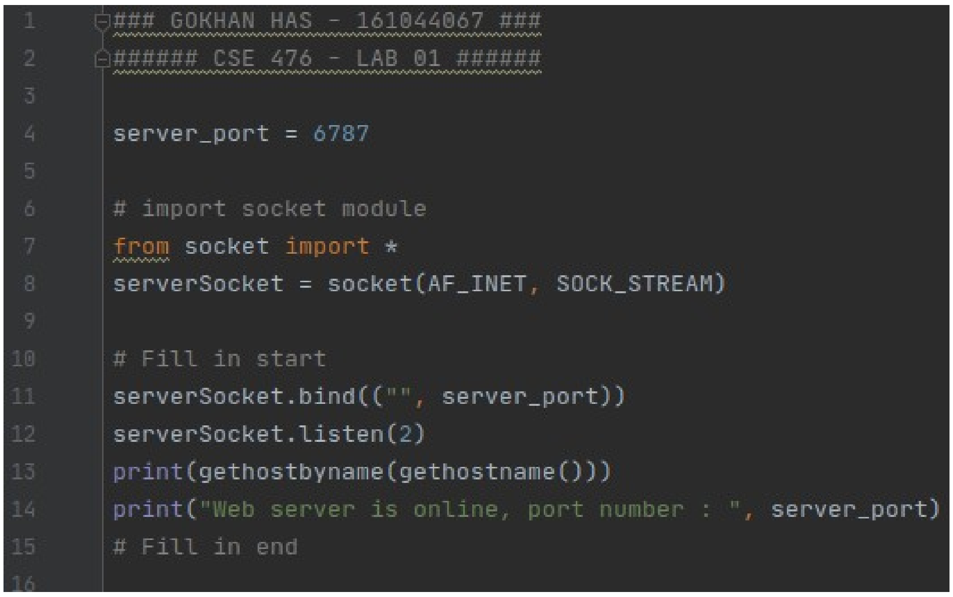
Final Report

Gökhan Has
161044067

Lecturer : Prof. Dr. Hasari Çelebi
Assistant : Abdullah Salih Bayraktar

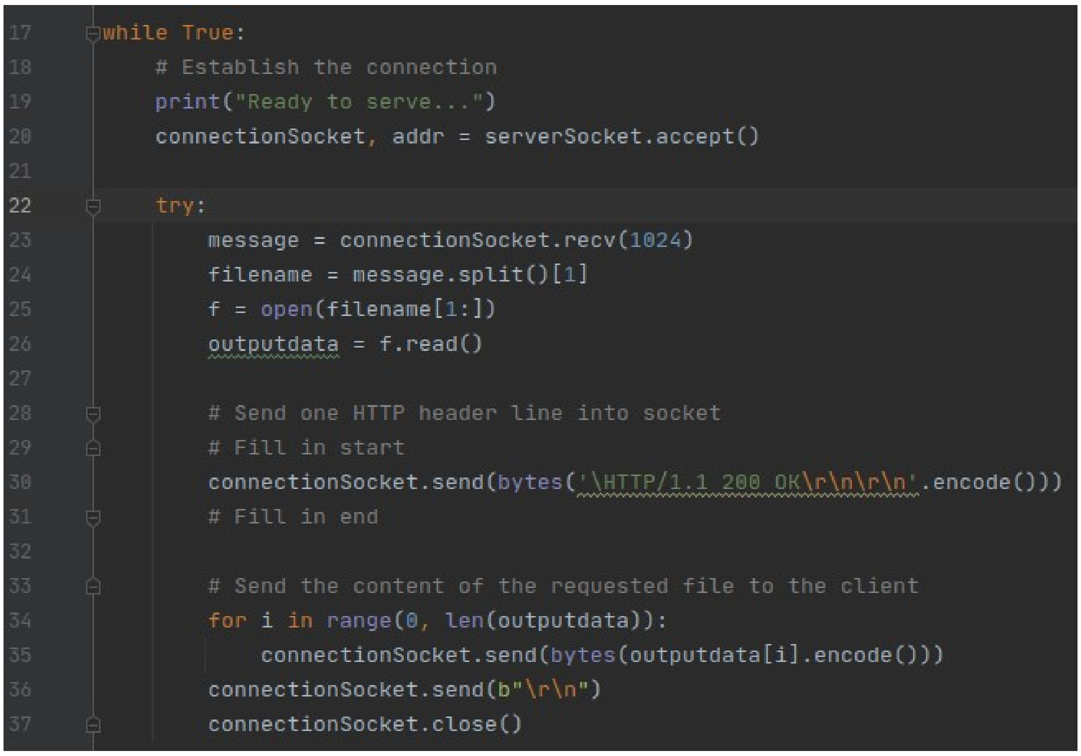
Lab 1: Web Server Lab

The sockets can be configured to act as a server and listen to incoming messages or connect to other applications as a client. Once both ends of a TCP / IP socket are connected, communication is two-way. That's why sockets were used in this lab.



```
1   ### GOKHAN HAS - 161044067 ###
2   ###### CSE 476 - LAB 01 ######
3
4   server_port = 6787
5
6   # import socket module
7   from socket import *
8   serverSocket = socket(AF_INET, SOCK_STREAM)
9
10  # Fill in start
11  serverSocket.bind(("", server_port))
12  serverSocket.listen(2)
13  print(gethostbyname(gethostname()))
14  print("Web server is online, port number : ", server_port)
15  # Fill in end
16
```

The socket used is associated with the server address using the bind () function, as seen in line 11. The address is the address of the machine where the server will run. Then, with the help of the called listen () function, the socket switches to server mode.



```
17  while True:
18      # Establish the connection
19      print("Ready to serve...")
20      connectionSocket, addr = serverSocket.accept()
21
22      try:
23          message = connectionSocket.recv(1024)
24          filename = message.split()[1]
25          f = open(filename[1:])
26          outputdata = f.read()
27
28          # Send one HTTP header line into socket
29          # Fill in start
30          connectionSocket.send(bytes('\HTTP/1.1 200 OK\r\n\r\n'.encode()))
31          # Fill in end
32
33          # Send the content of the requested file to the client
34          for i in range(0, len(outputdata)):
35              connectionSocket.send(bytes(outputdata[i].encode()))
36          connectionSocket.send(b"\r\n")
37          connectionSocket.close()
```

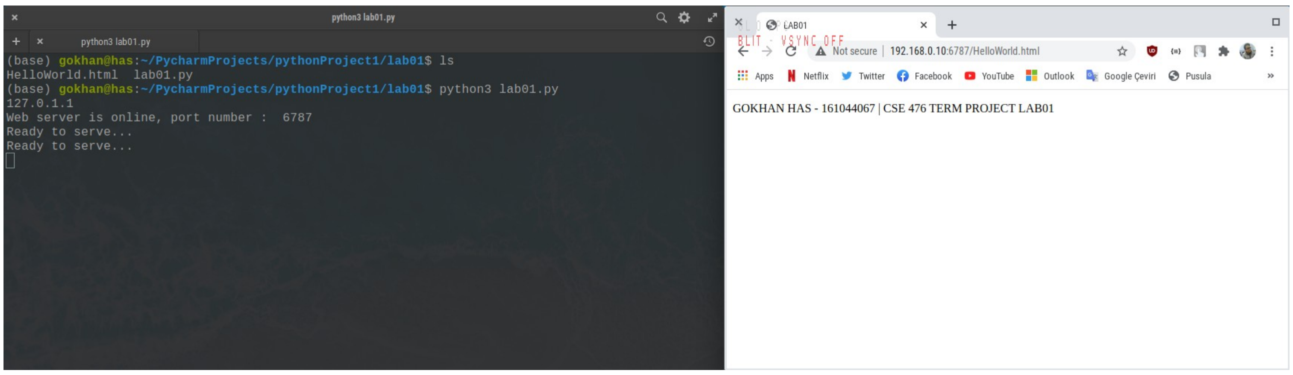
As seen in the 20th line, the server will now start to wait for the connection to come by using the accept () function. The code will stay on this line until the connection arrives. After the connection is made, the code will continue to run.

Function accept() returns an open connection between the server and client, along with the address of the client. The connection is actually a different socket on another port. Data is read from the connection with recv().

The output data on line 26 holds the content of the HelloWorld.html file in the same directory. The contents of this outputdata, namely HellloWorld.html, will be sent to the connected client.

A 200-coded HTTP message is then sent indicating that the client has successfully connected. This event took place in line 31.

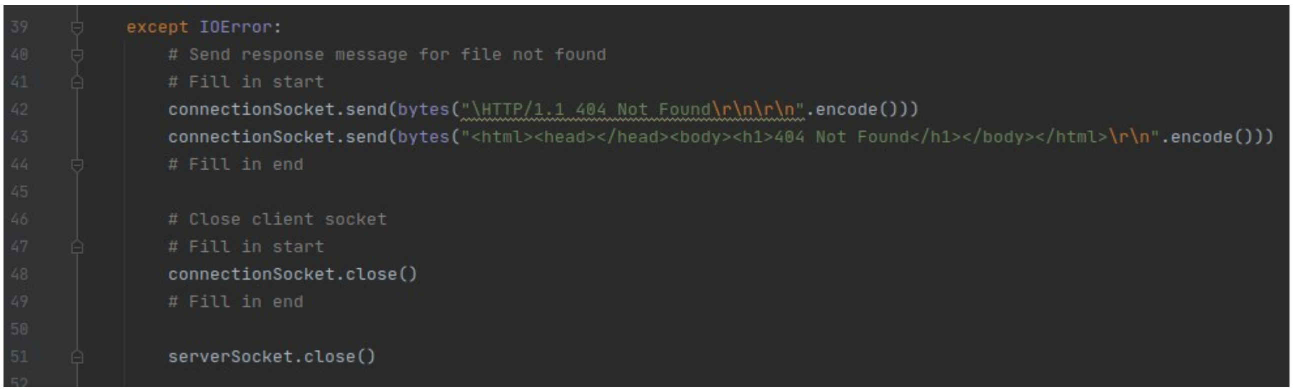
With the loop starting from the 35th line in the skeleton code, the content of this html file is sent to the client byte byte. When this is done, the file content is returned and the web page is displayed on the client side.



```
(base) gokhan@has:~/PycharmProjects/pythonProject1/lab01$ ls
HelloWorld.html  lab01.py
(base) gokhan@has:~/PycharmProjects/pythonProject1/lab01$ python3 lab01.py
127.0.1.1
Web server is online, port number :  6787
Ready to serve...
Ready to serve...
```

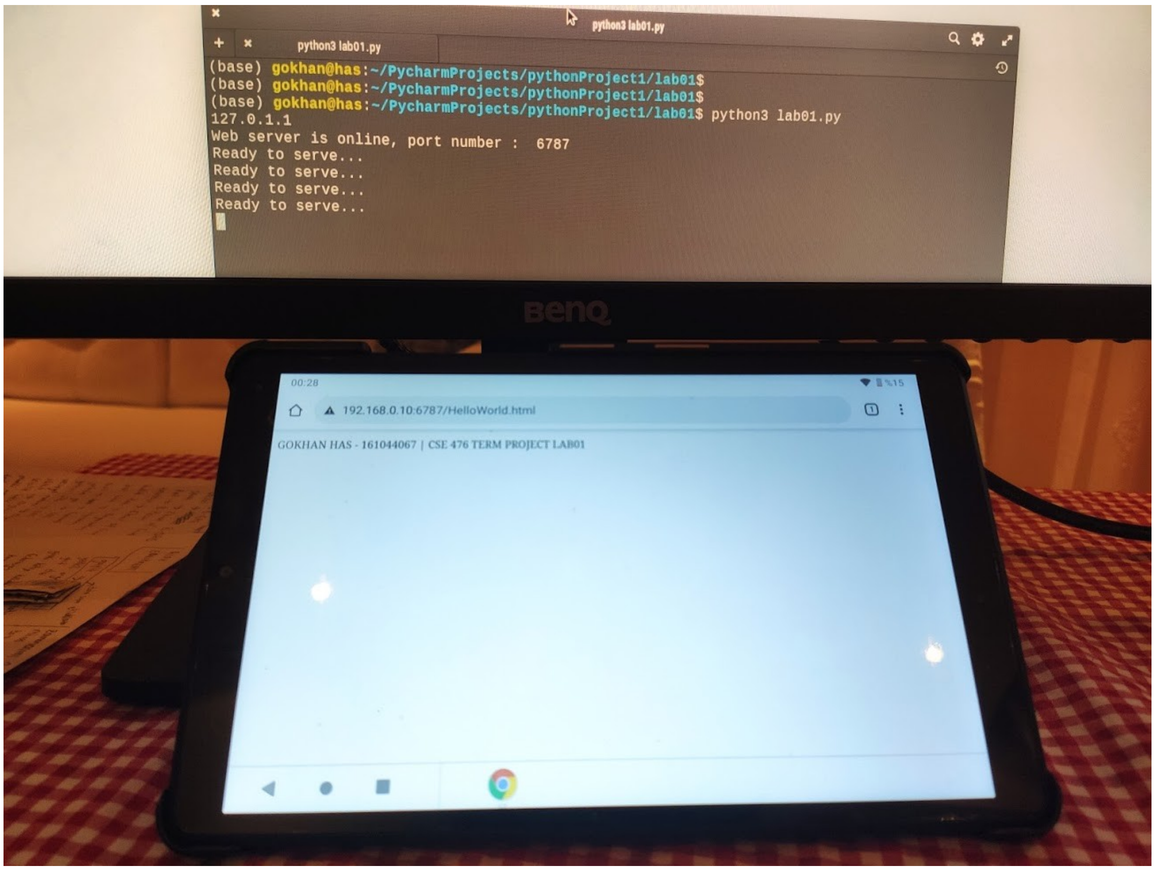
GOKHAN HAS - 161044067 | CSE 476 TERM PROJECT LAB01

Figure 1: Program running in same device



```python
except IOError:
    # Send response message for file not found
    # Fill in start
    connectionSocket.send(bytes("\HTTP/1.1 404 Not Found\r\n\r\n".encode()))
    connectionSocket.send(bytes("<html><head></head><body><h1>404 Not Found</h1></body></html>\r\n".encode()))
    # Fill in end

    # Close client socket
    # Fill in start
    connectionSocket.close()
    # Fill in end

    serverSocket.close()
```

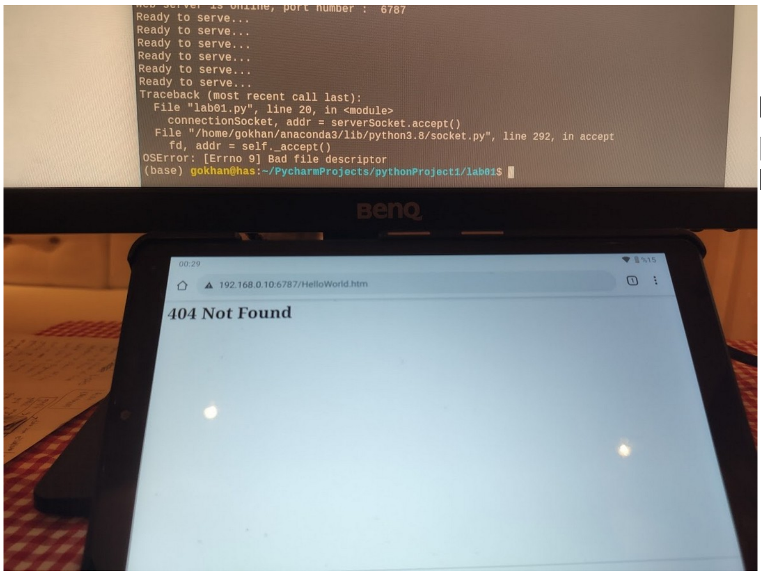
If an incorrect page request is received or the link is broken, a text 404 not found will be displayed. It is normal for the program to give an error in the output as seen in the pictures below. Because the socket is closed and it returns to the beginning of the loop and tries to get new connections with the accept () function. After the 51st line, break was put in order not to give an error message.



```
(base) gokhan@has:~/PycharmProjects/pythonProject1/lab01$
(base) gokhan@has:~/PycharmProjects/pythonProject1/lab01$
(base) gokhan@has:~/PycharmProjects/pythonProject1/lab01$ python3 lab01.py
127.0.1.1
Web server is online, port number :  6787
Ready to serve...
Ready to serve...
Ready to serve...
Ready to serve...
```

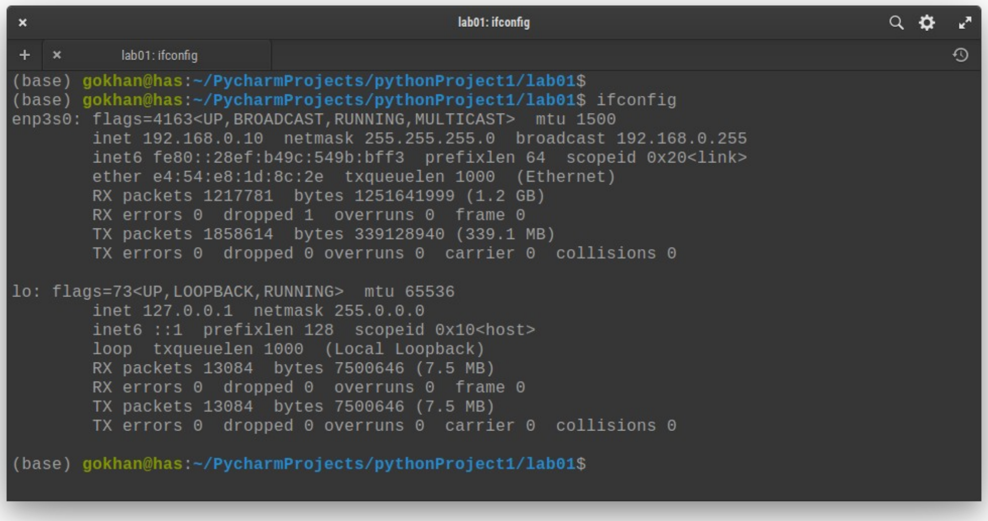
python3 lab01.py
00:28
192.168.0.10:6787/HelloWorld.html
GOKHAN HAS - 161044067 | CSE 476 TERM PROJECT LAB01

Figure 2: Program running other device



```
Web server is online, port number :  6787
Ready to serve...
Ready to serve...
Ready to serve...
Ready to serve...
Ready to serve...
Ready to serve...
Traceback (most recent call last):
  File "lab01.py", line 20, in <module>
    connectionSocket, addr = serverSocket.accept()
  File "/home/gokhan/anaconda3/lib/python3.8/socket.py", line 292, in accept
    fd, addr = self._accept()
OSError: [Errno 9] Bad file descriptor
(base) gokhan@has:~/PycharmProjects/pythonProject1/lab01$
```

Here an error has been provided by entering HelloWorld.html.



```
(base) gokhan@has:~/PycharmProjects/pythonProject1/lab01$
(base) gokhan@has:~/PycharmProjects/pythonProject1/lab01$ ifconfig
enp3s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.0.10  netmask 255.255.255.0  broadcast 192.168.0.255
        inet6 fe80::28ef:b49c:549b:bff3  prefixlen 64  scopeid 0x20<link>
        ether e4:54:e8:1d:8c:2e  txqueuelen 1000  (Ethernet)
        RX packets 1217781  bytes 1251641999 (1.2 GB)
        RX errors 0  dropped 1  overruns 0  frame 0
        TX packets 1858614  bytes 339128940 (339.1 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 13084  bytes 7500646 (7.5 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 13084  bytes 7500646 (7.5 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

(base) gokhan@has:~/PycharmProjects/pythonProject1/lab01$
```

In the Linux operating system, the IP address can be learned with the ifconfig command on the computer where the program will run. Here you can see the ip address in the form of inet 192.168.0.10.

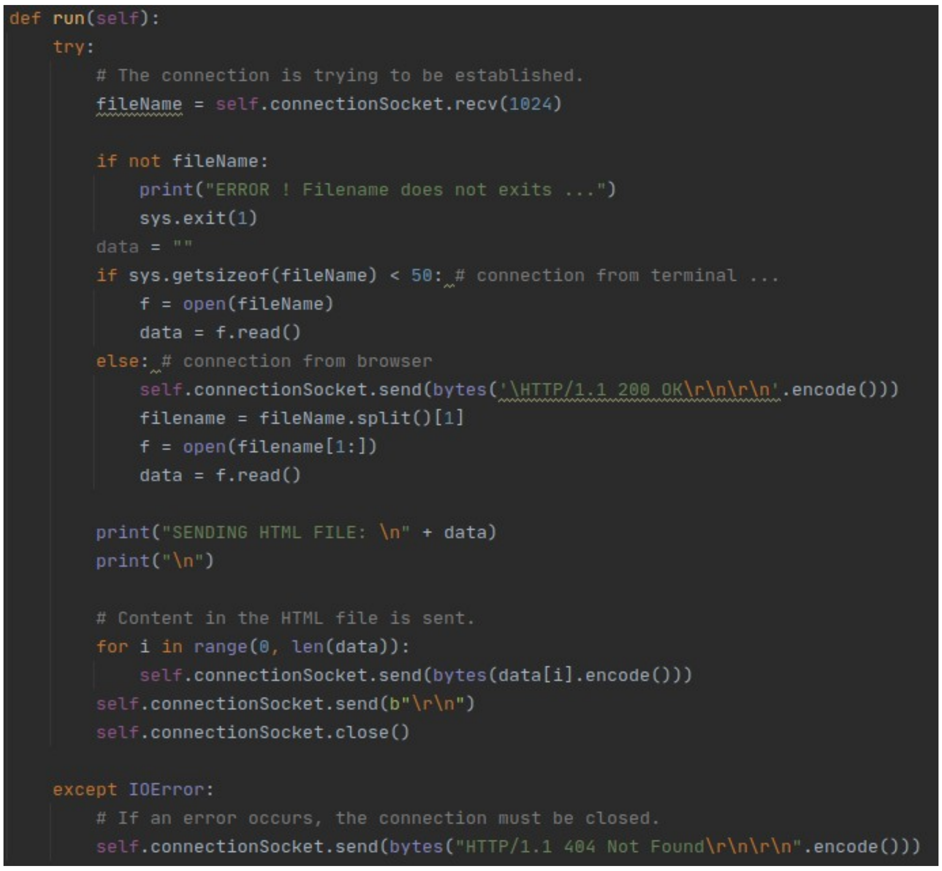
Lab 1: Web Server Lab – Optional Exercises

Here I coded the first two optional exercises together. I wrote one Thread class using the threading library in Python. This class gets the address with the connection socket in conmsturctor. Later, the run () function, which is the function that will be executed when the thread coming from the threading library runs, has been overridden.

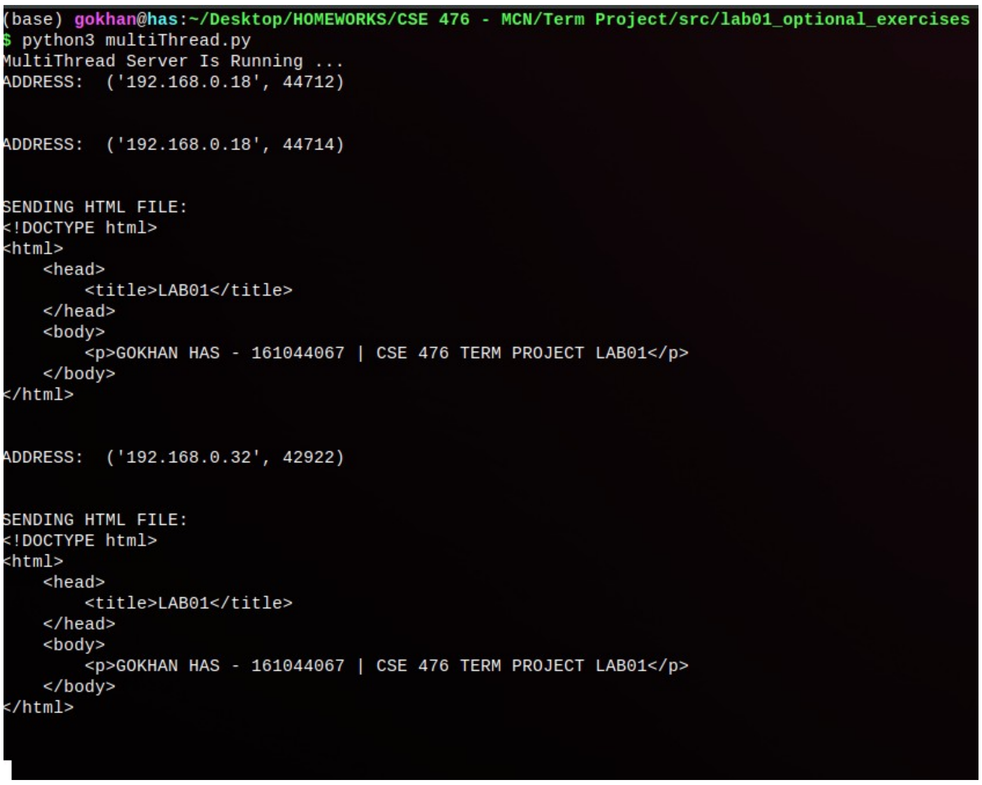
First of all, it is determined whether your network is from a browser (first question) or a terminal (second question). The content of the HelloWorld.html file is sent, which will then be sent to the browser or terminal.

If there is an error, 404 not found is sent in the same way as in the previous section.

An array of threads is created in Main and each incoming request is assigned to a thread, and the request continues to be listened. While only one request was received in the previous party, now more than one request can be received at the same time.

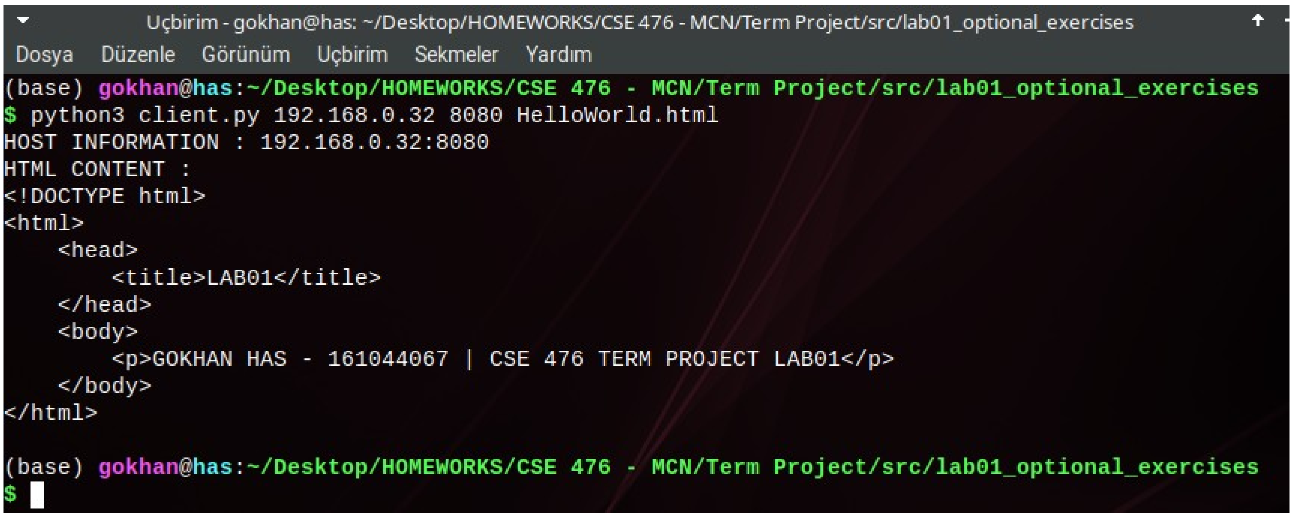


```
def run(self):
    try:
        # The connection is trying to be established.
        fileName = self.connectionSocket.recv(1024)

        if not fileName:
            print("ERROR ! Filename does not exits ...")
            sys.exit(1)
        data = ""
        if sys.getsizeof(fileName) < 50: # connection from terminal ...
            f = open(fileName)
            data = f.read()
        else: # connection from browser
            self.connectionSocket.send(bytes('\HTTP/1.1 200 OK\r\n\r\n'.encode()))
            filename = fileName.split()[1]
            f = open(filename[1:])
            data = f.read()

        print("SENDING HTML FILE: \n" + data)
        print("\n")

        # Content in the HTML file is sent.
        for i in range(0, len(data)):
            self.connectionSocket.send(bytes(data[i].encode()))
        self.connectionSocket.send(b"\r\n")
        self.connectionSocket.close()

    except IOError:
        # If an error occurs, the connection must be closed.
        self.connectionSocket.send(bytes("HTTP/1.1 404 Not Found\r\n\r\n".encode()))
```



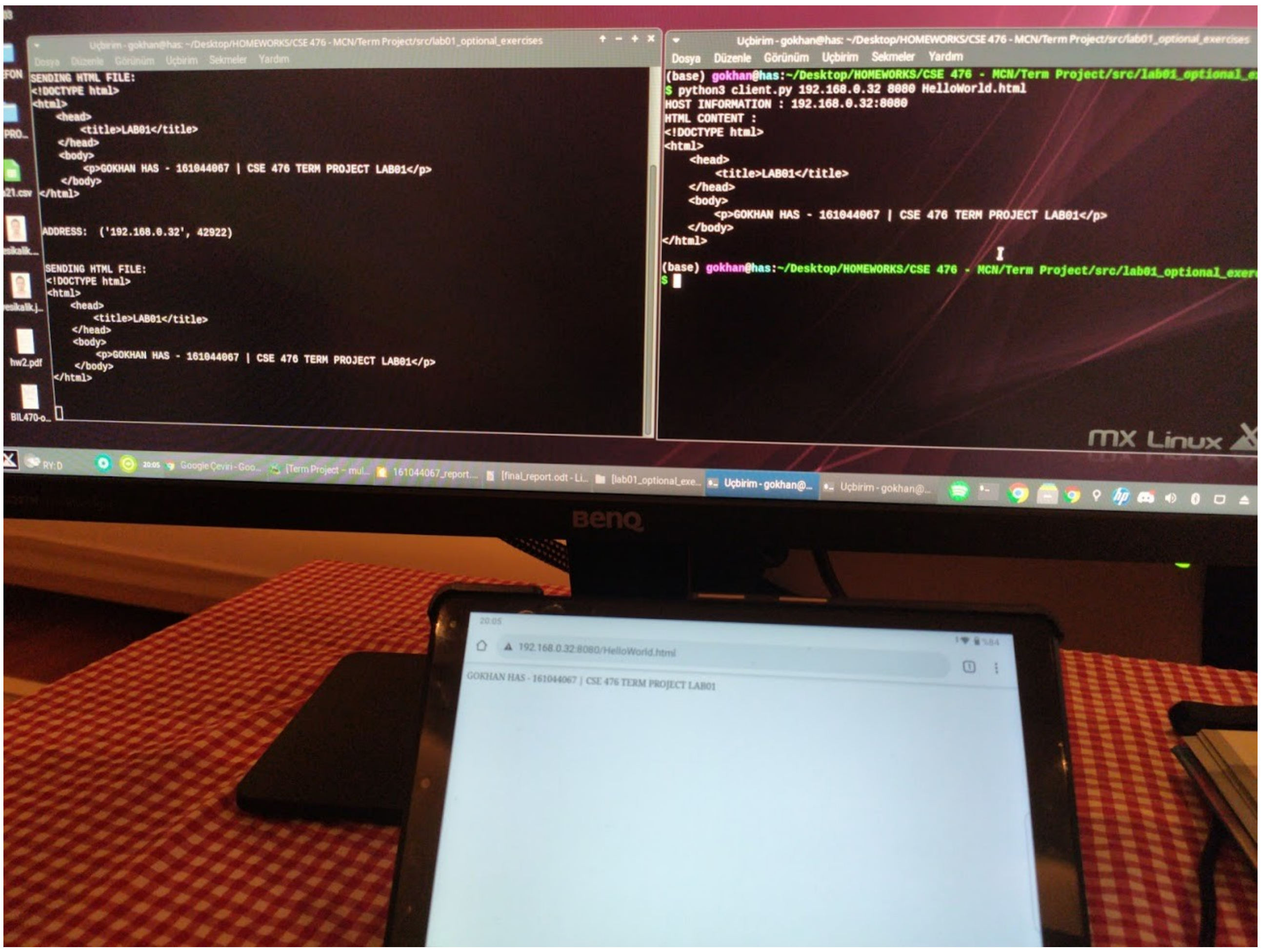
```
(base) gokhan@has:~/Desktop/HOMEWORKS/CSE 476 - MCN/Term Project/src/lab01_optional_exercises
$ python3 multiThread.py
MultiThread Server Is Running ...
ADDRESS:  ('192.168.0.18', 44712)


ADDRESS:  ('192.168.0.18', 44714)


SENDING HTML FILE:
<!DOCTYPE html>
<html>
    <head>
        <title>LAB01</title>
    </head>
    <body>
        <p>GOKHAN HAS - 161044067 | CSE 476 TERM PROJECT LAB01</p>
    </body>
</html>


ADDRESS:  ('192.168.0.32', 42922)


SENDING HTML FILE:
<!DOCTYPE html>
<html>
    <head>
        <title>LAB01</title>
    </head>
    <body>
        <p>GOKHAN HAS - 161044067 | CSE 476 TERM PROJECT LAB01</p>
    </body>
</html>
```

Figure 3: Multithread server code


Uçbirim - gokhan@has: ~/Desktop/HOMEWORKS/CSE 476 - MCN/Term Project/src/lab01_optional_exercises
Dosya Düzenle Görünüm Uçbirim Sekmeler Yardım

```
(base) gokhan@has:~/Desktop/HOMEWORKS/CSE 476 - MCN/Term Project/src/lab01_optional_exercises
$ python3 client.py 192.168.0.32 8080 HelloWorld.html
HOST INFORMATION : 192.168.0.32:8080
HTML CONTENT :
<!DOCTYPE html>
<html>
    <head>
        <title>LAB01</title>
    </head>
    <body>
        <p>GOKHAN HAS - 161044067 | CSE 476 TERM PROJECT LAB01</p>
    </body>
</html>

(base) gokhan@has:~/Desktop/HOMEWORKS/CSE 476 - MCN/Term Project/src/lab01_optional_exercises
$ 
```

Figure 4: Client terminal code



```
SENDING HTML FILE:
<!DOCTYPE html>
<html>
    <head>
        <title>LAB01</title>
    </head>
    <body>
        <p>GOKHAN HAS - 161044067 | CSE 476 TERM PROJECT LAB01</p>
    </body>
</html>


ADDRESS:  ('192.168.0.32', 42922)


SENDING HTML FILE:
<!DOCTYPE html>
<html>
    <head>
        <title>LAB01</title>
    </head>
    <body>
        <p>GOKHAN HAS - 161044067 | CSE 476 TERM PROJECT LAB01</p>
    </body>
</html>
```

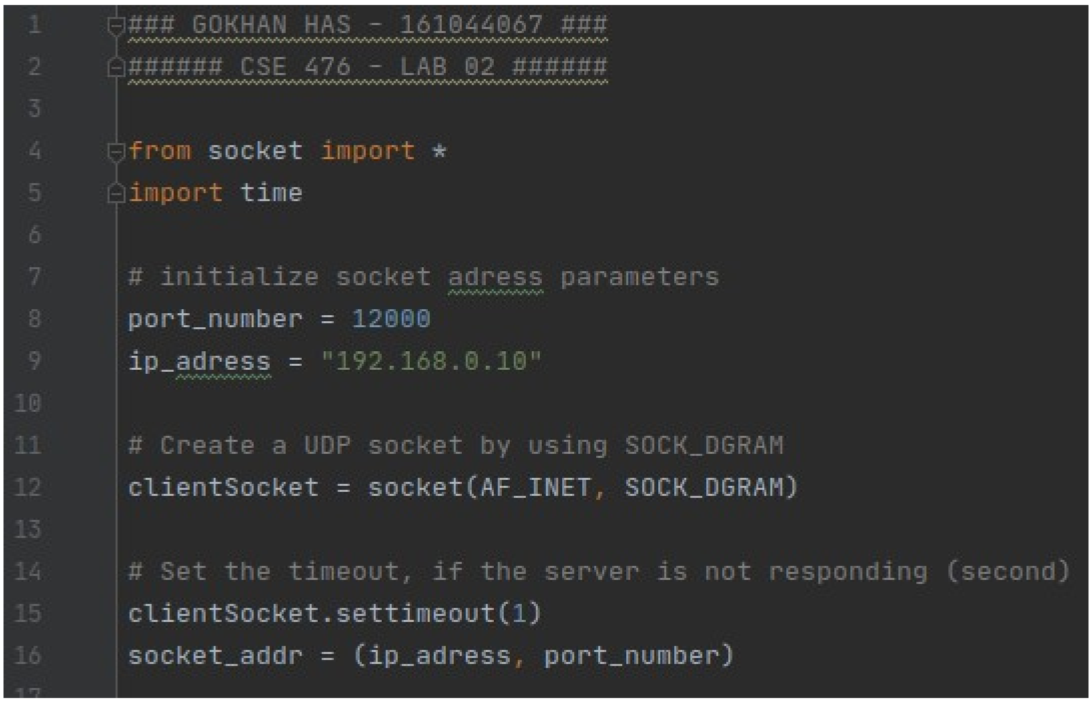
```
(base) gokhan@has:~/Desktop/HOMEWORKS/CSE 476 - MCN/Term Project/src/lab01_optional_e
$ python3 client.py 192.168.0.32 8080 HelloWorld.html
HOST INFORMATION : 192.168.0.32:8080
HTML CONTENT :
<!DOCTYPE html>
<html>
    <head>
        <title>LAB01</title>
    </head>
    <body>
        <p>GOKHAN HAS - 161044067 | CSE 476 TERM PROJECT LAB01</p>
    </body>
</html>

(base) gokhan@has:~/Desktop/HOMEWORKS/CSE 476 - MCN/Term Project/src/lab01_optional_exer
$ 
```

192.168.0.32:8080/HelloWorld.html
GOKHAN HAS - 161044067 | CSE 476 TERM PROJECT LAB01
MX Linux
BenQ

Figure 5: Browser and terminal running example (other machine)

Lab 2: UDP Pinger Lab

The UDP client is similar the UDPPingerServer, but does not use bind() to attach its socket to an address. It uses sendto() to deliver its message directly to the server, and recvfrom() to receive the response.

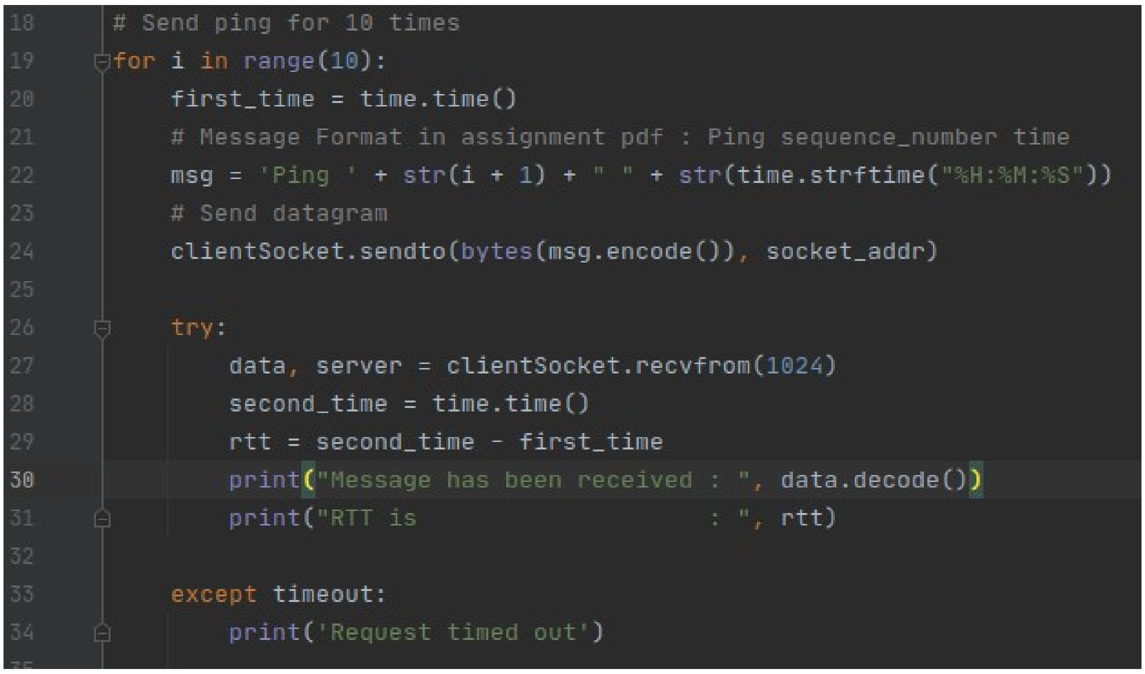


```
1   ### GOKHAN HAS - 161044067 ###
2   ###### CSE 476 - LAB 02 ######
3
4   from socket import *
5   import time
6
7   # initialize socket adress parameters
8   port_number = 12000
9   ip_adress = "192.168.0.10"
10
11  # Create a UDP socket by using SOCK_DGRAM
12  clientSocket = socket(AF_INET, SOCK_DGRAM)
13
14  # Set the timeout, if the server is not responding (second)
15  clientSocket.settimeout(1)
16  socket_addr = (ip_adress, port_number)
```

The server code is already as shown in the assignment pdf. No modifications have been made. At first, the client code was assigned socket address variables. These operations were done in 8 and 9 lines. Since 12000 port is used in server code, 12000 port is used in client code. The IP address of the machine where the server code will work is given as the IP address. This variable must be changed during the check.

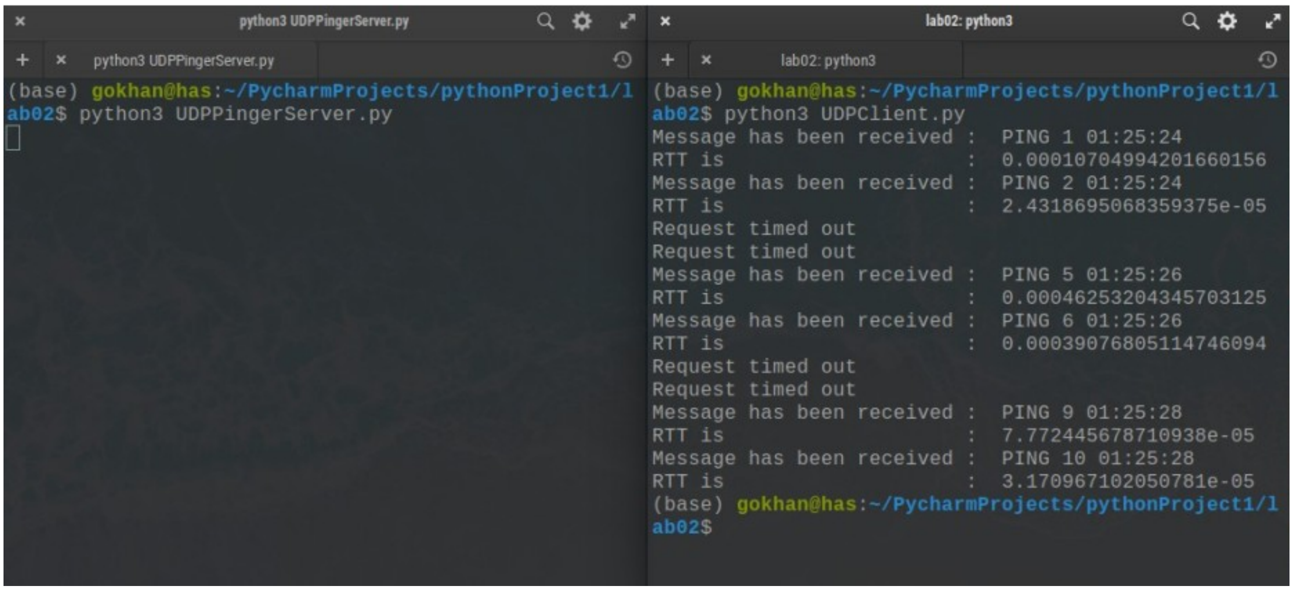
Since data will be received from the UDP server, the SOCK_DGRAM tag is used in the 12th line. Unlike TCP, there is no bind () operation in UDP client code.

It is written that if the message does not arrive in Assignmet within 1 second, timeout status must be switched to. Therefore, settimeout (1) function is used in the 15th line. A socket address is created immediately afterwards.



```
18  # Send ping for 10 times
19  for i in range(10):
20      first_time = time.time()
21      # Message Format in assignment pdf : Ping sequence_number time
22      msg = 'Ping ' + str(i + 1) + " " + str(time.strftime("%H:%M:%S"))
23      # Send datagram
24      clientSocket.sendto(bytes(msg.encode()), socket_addr)
25
26      try:
27          data, server = clientSocket.recvfrom(1024)
28          second_time = time.time()
29          rtt = second_time - first_time
30          print("Message has been received : ", data.decode())
31          print("RTT is                    : ", rtt)
32
33      except timeout:
34          print('Request timed out')
```

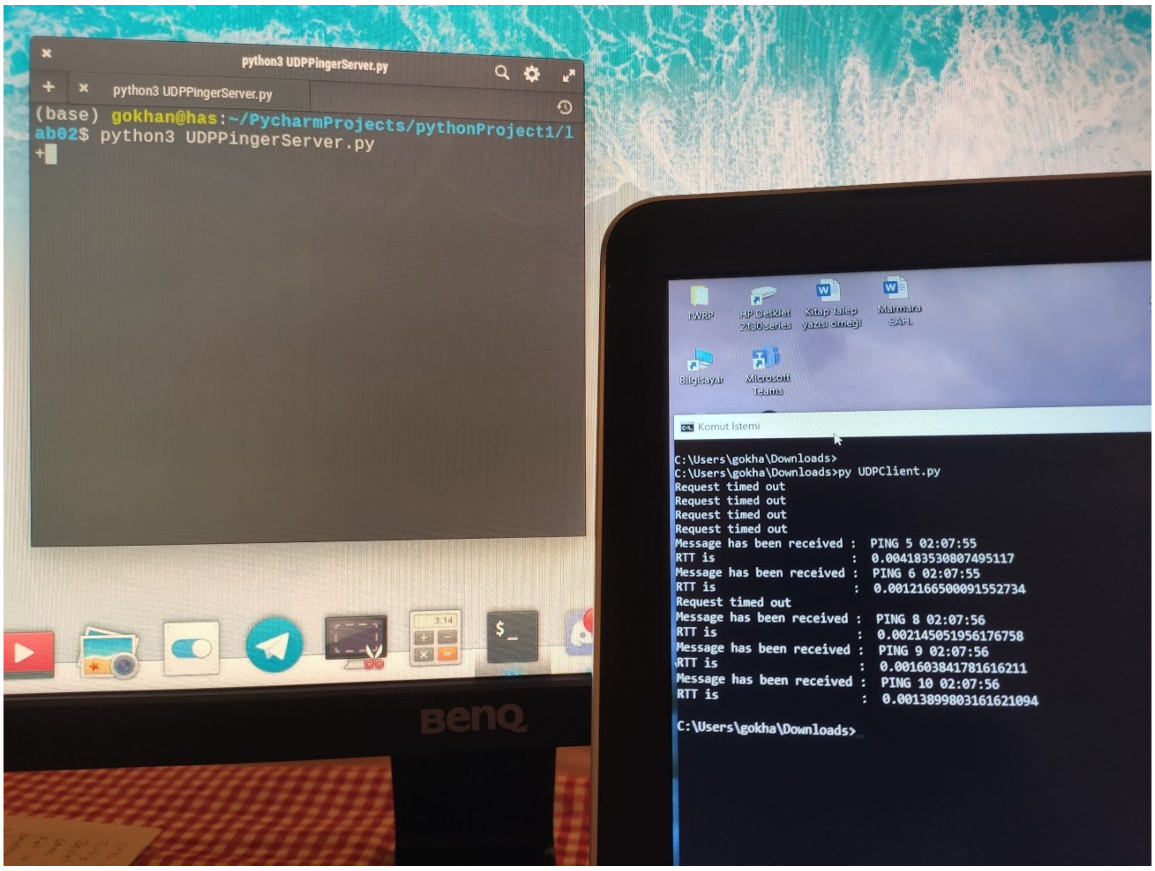
When asked to send 10 pings, the loop in line 19 was used. Then time functions are used. These functions are available in the time library. Message format is as shown in the assignment pdf. It is in the form of "Ping sequence_number time". If return is received in less than 1 second, data acquisition is performed on the 27th line. If the process takes longer than 1 second, it skips to the 33rd line and the desired message is printed on the screen. If the message is received correctly, the round trip time value is printed as requested.



```
(base) gokhan@has:~/PycharmProjects/pythonProject1/l
ab02$ python3 UDPPingerServer.py
```

```
(base) gokhan@has:~/PycharmProjects/pythonProject1/l
ab02$ python3 UDPClient.py
Message has been received :  PING 1 01:25:24
RTT is                    :  0.00010704994201660156
Message has been received :  PING 2 01:25:24
RTT is                    :  2.4318695068359375e-05
Request timed out
Request timed out
Message has been received :  PING 5 01:25:26
RTT is                    :  0.00046253204345703125
Message has been received :  PING 6 01:25:26
RTT is                    :  0.00039076805114746094
Request timed out
Request timed out
Message has been received :  PING 9 01:25:28
RTT is                    :  7.772445678710938e-05
Message has been received :  PING 10 01:25:28
RTT is                    :  3.170967102050781e-05
(base) gokhan@has:~/PycharmProjects/pythonProject1/l
ab02$
```

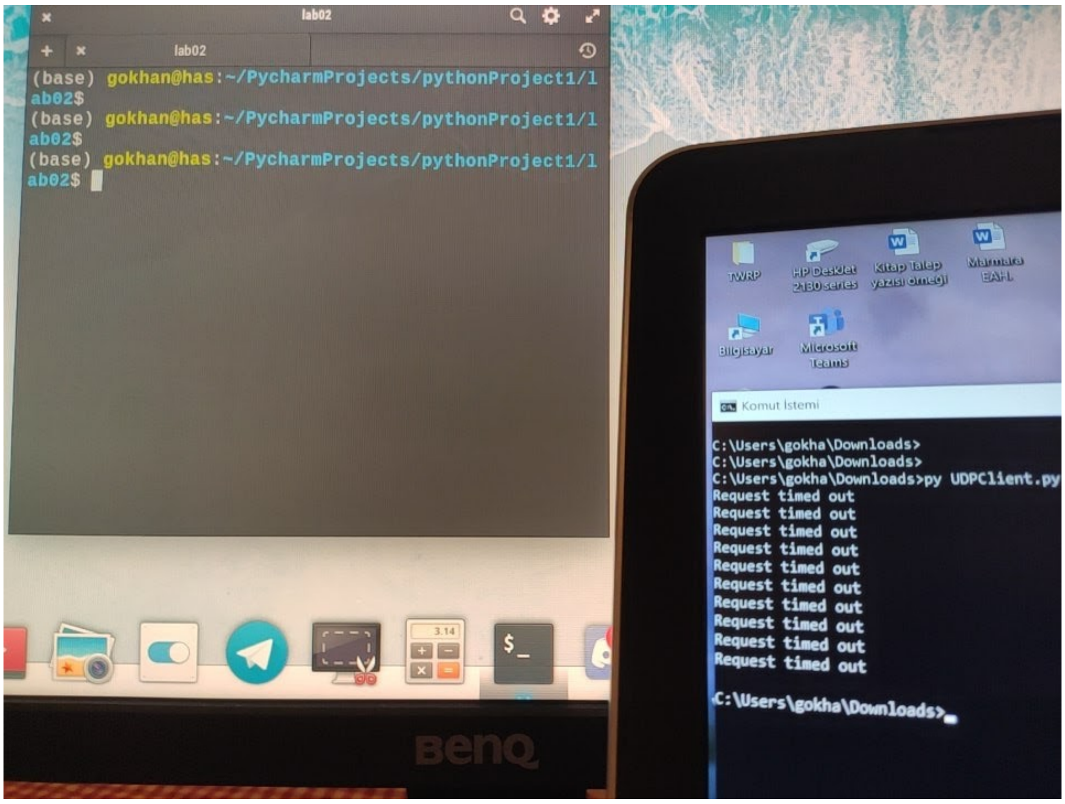
Figure 6: Running server and client in same machine



```
(base) gokhan@has:~/PycharmProjects/pythonProject1/l
ab02$ python3 UDPPingerServer.py
+
```

```
C:\Users\gokha\Downloads>
C:\Users\gokha\Downloads>py UDPClient.py
Request timed out
Request timed out
Request timed out
Request timed out
Message has been received :  PING 5 02:07:55
RTT is                    :  0.004183530807495117
Message has been received :  PING 6 02:07:55
RTT is                    :  0.0012166500091552734
Request timed out
Message has been received :  PING 8 02:07:56
RTT is                    :  0.002145051956176758
Message has been received :  PING 9 02:07:56
RTT is                    :  0.001603841781616211
Message has been received :  PING 10 02:07:56
RTT is                    :  0.0013899803161621094

C:\Users\gokha\Downloads>
```

Figure 7: Client code is different machine



```
(base) gokhan@has:~/PycharmProjects/pythonProject1/l
ab02$
(base) gokhan@has:~/PycharmProjects/pythonProject1/l
ab02$
(base) gokhan@has:~/PycharmProjects/pythonProject1/l
ab02$
```

```
C:\Users\gokha\Downloads>
C:\Users\gokha\Downloads>
C:\Users\gokha\Downloads>py UDPClient.py
Request timed out
Request timed out
Request timed out
Request timed out
Request timed out
Request timed out
Request timed out
Request timed out
Request timed out
Request timed out

C:\Users\gokha\Downloads>
```

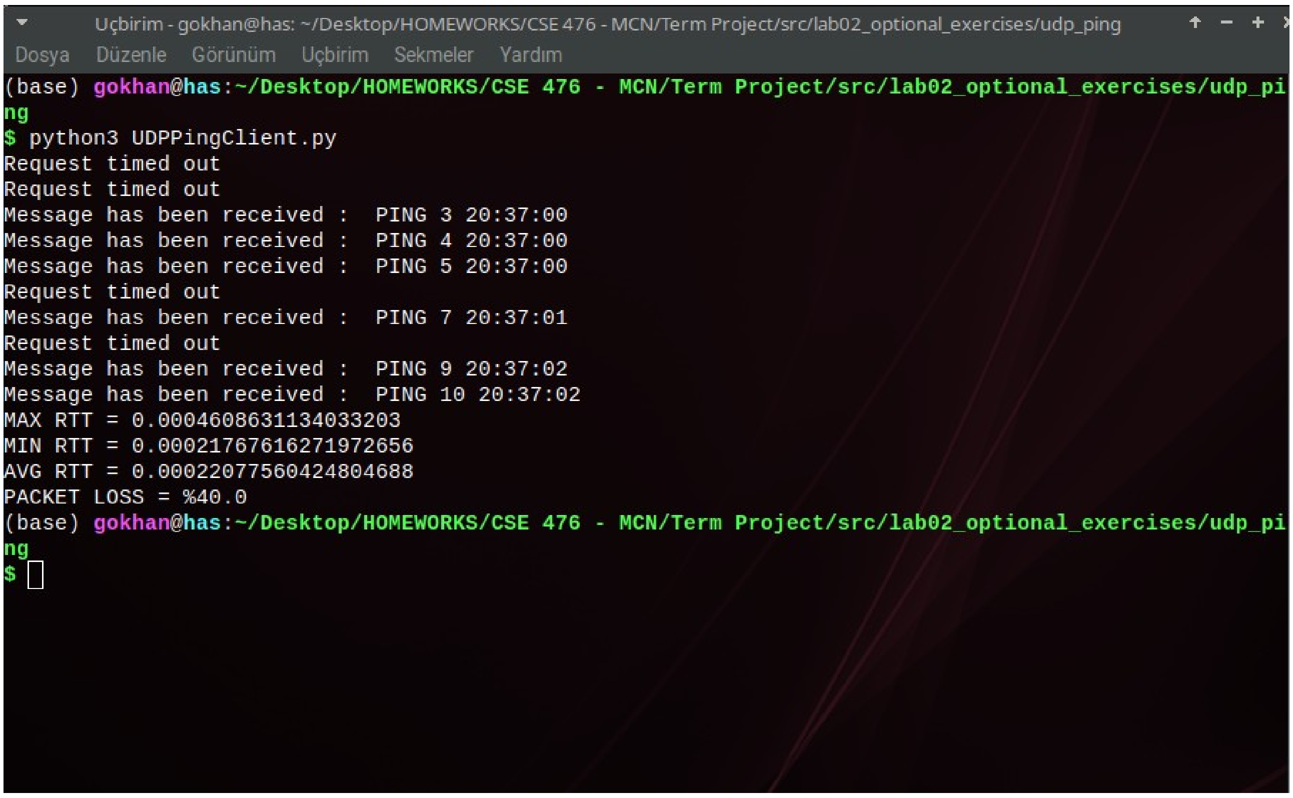
Figure 8: Client code is different machine and server code does not running

Lab 2: UDP Pinger Lab – Optional Exercises

I put them in separate folders so that the codes do not get mixed up and there is no difficulty in checking. Although most codes are the same, that folder only performs the specified function. In order for client codes to work on another computer, the topmost ip address part of the code must be changed. It is not done because it is not requested to be entered from the command line at runtime. It is possible to change the desired parts from the command line.

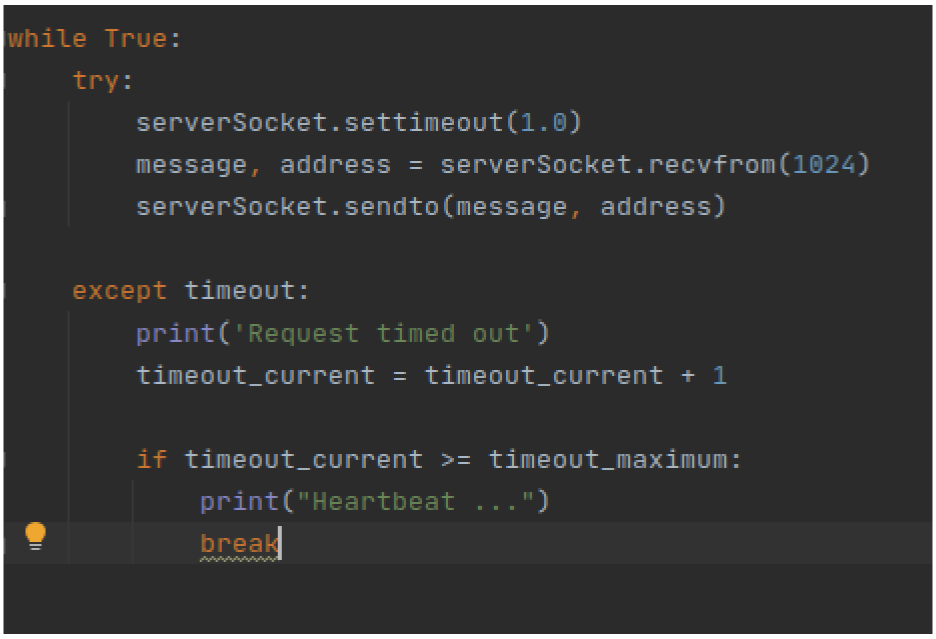
I have kept 4 variables as extra to the client code. These are the variables rtt_max, rtt_min, rtt_sum and loss_packet. After importing data from the server, the classic rtt value is calculated. The rtt_max and rtt_min values are found by comparing with this calculated rtt value. rtt_sum value is added by making + = rtt value. If it enters the except timeout part, the loss_packet value is increased by 1. Because the package has been lost.

After doing these 10 times, the value of rtt_avg is calculated. Then the loss percentage is calculated. These calculated values are printed on the screen respectively.

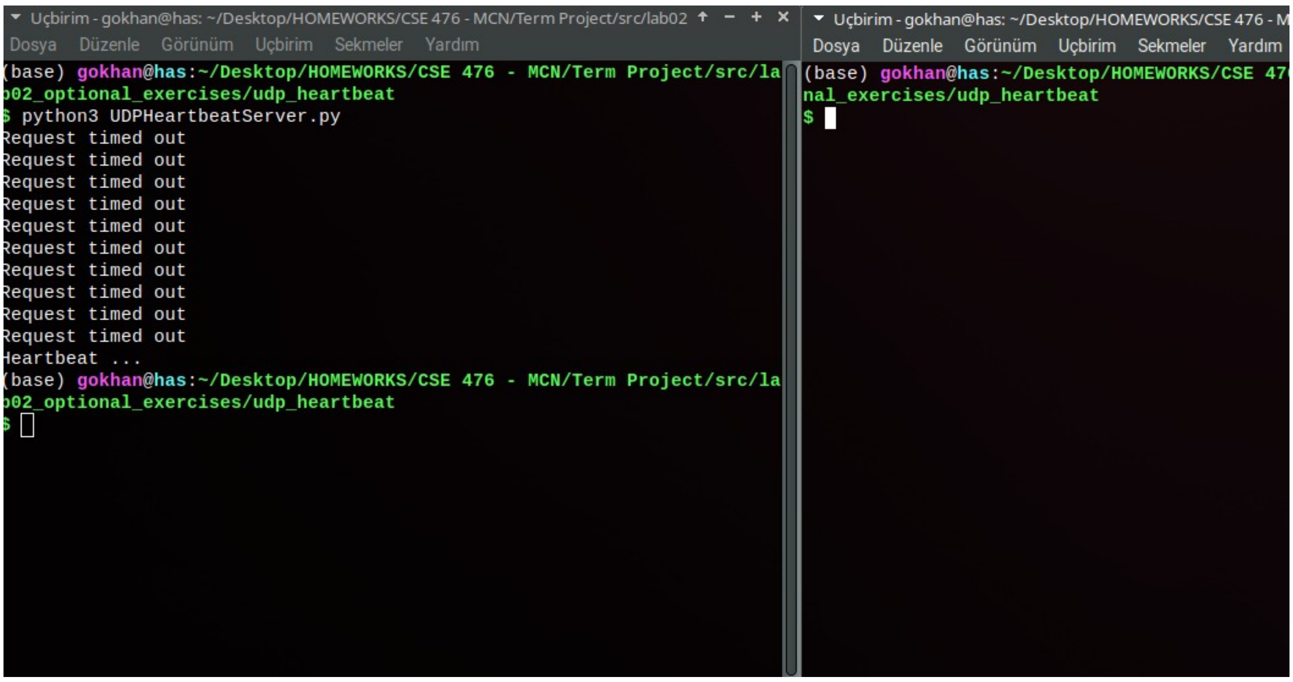

Uçbirim - gokhan@has: ~/Desktop/HOMEWORKS/CSE 476 - MCN/Term Project/src/lab02_optional_exercises/udp_ping
Dosya Düzenle Görünüm Uçbirim Sekmeler Yardım

```
(base) gokhan@has:~/Desktop/HOMEWORKS/CSE 476 - MCN/Term Project/src/lab02_optional_exercises/udp_pi
ng
$ python3 UDPPingClient.py
Request timed out
Request timed out
Message has been received :  PING 3 20:37:00
Message has been received :  PING 4 20:37:00
Message has been received :  PING 5 20:37:00
Request timed out
Message has been received :  PING 7 20:37:01
Request timed out
Message has been received :  PING 9 20:37:02
Message has been received :  PING 10 20:37:02
MAX RTT = 0.0004608631134033203
MIN RTT = 0.00021767616271972656
AVG RTT = 0.00022077560424804688
PACKET LOSS = %40.0
(base) gokhan@has:~/Desktop/HOMEWORKS/CSE 476 - MCN/Term Project/src/lab02_optional_exercises/udp_pi
ng
$ 
```

The UDPHeartbeatClient instance is identical to the UDPPingClient. Extra rtt values are calculated. But UDPHeartbeatServer is completely different. Timeout_maximum value is kept in this server and this value is given 10 by me. So the heart rate can be thought of as 10. If it timeouts 10 times, the server will shut down. The ip variable can still be changed in the server. Links count. If timeout is greater than or equal to timeout_maximum, the server will be terminated.



```
while True:
    try:
        serverSocket.settimeout(1.0)
        message, address = serverSocket.recvfrom(1024)
        serverSocket.sendto(message, address)

    except timeout:
        print('Request timed out')
        timeout_current = timeout_current + 1

        if timeout_current >= timeout_maximum:
            print("Heartbeat ...")
            break
```

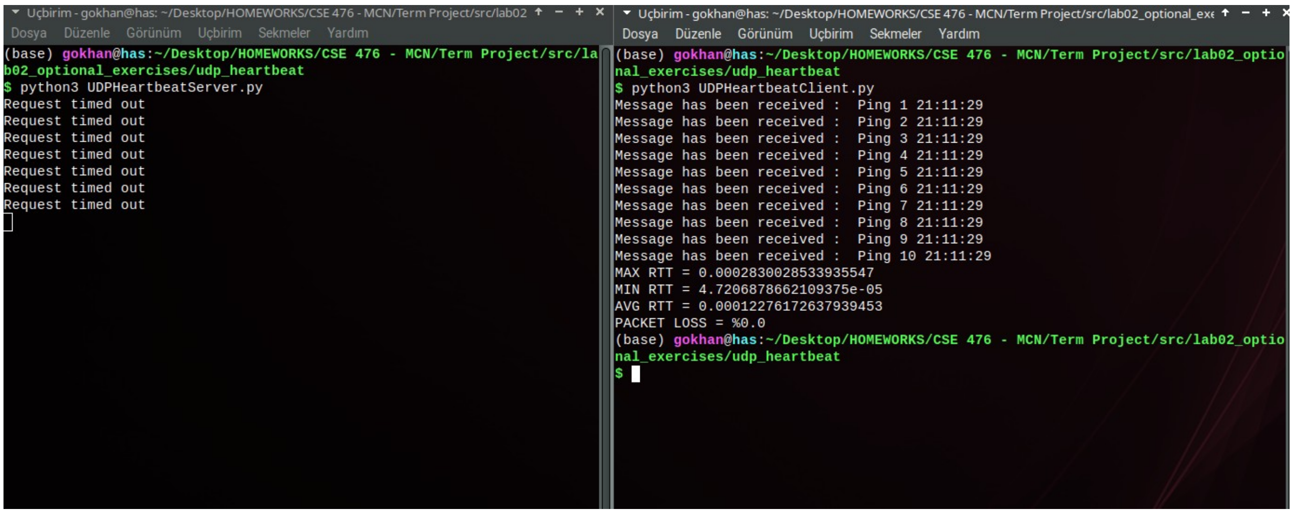


Uçbirim - gokhan@has: ~/Desktop/HOMEWORKS/CSE 476 - MCN/Term Project/src/lab02
Dosya Düzenle Görünüm Uçbirim Sekmeler Yardım
Uçbirim - gokhan@has: ~/Desktop/HOMEWORKS/CSE 476 - M
Dosya Düzenle Görünüm Uçbirim Sekmeler Yardım

```
(base) gokhan@has:~/Desktop/HOMEWORKS/CSE 476 - MCN/Term Project/src/la
b02_optional_exercises/udp_heartbeat
$ python3 UDPHeartbeatServer.py
Request timed out
Request timed out
Request timed out
Request timed out
Request timed out
Request timed out
Request timed out
Request timed out
Request timed out
Request timed out
Heartbeat ...
(base) gokhan@has:~/Desktop/HOMEWORKS/CSE 476 - MCN/Term Project/src/la
b02_optional_exercises/udp_heartbeat
$
```

```
(base) gokhan@has:~/Desktop/HOMEWORKS/CSE 47
nal_exercises/udp_heartbeat
$
```

Figure 9: Heartbeat example. Client is not running.


Uçbirim - gokhan@has: ~/Desktop/HOMEWORKS/CSE 476 - MCN/Term Project/src/lab02
Dosya Düzenle Görünüm Uçbirim Sekmeler Yardım
Uçbirim - gokhan@has: ~/Desktop/HOMEWORKS/CSE 476 - MCN/Term Project/src/lab02_optional_exe
Dosya Düzenle Görünüm Uçbirim Sekmeler Yardım

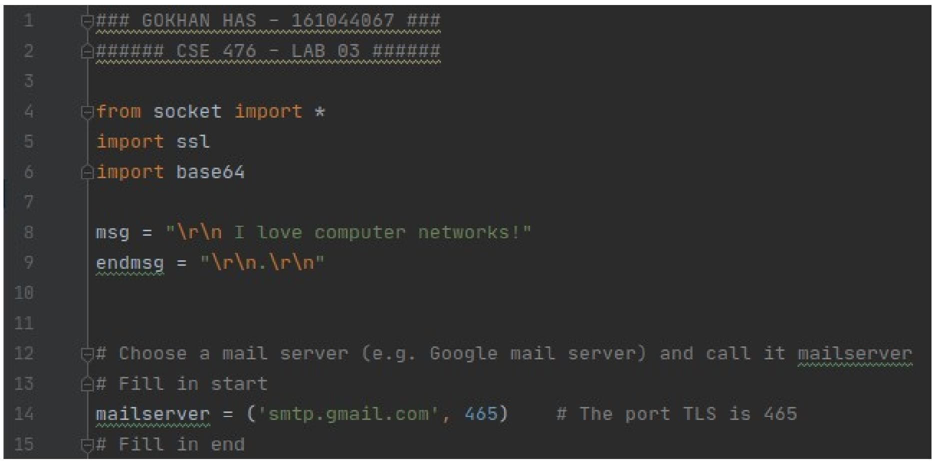
```
(base) gokhan@has:~/Desktop/HOMEWORKS/CSE 476 - MCN/Term Project/src/la
b02_optional_exercises/udp_heartbeat
$ python3 UDPHeartbeatServer.py
Request timed out
Request timed out
Request timed out
Request timed out
Request timed out
Request timed out
Request timed out
```

```
(base) gokhan@has:~/Desktop/HOMEWORKS/CSE 476 - MCN/Term Project/src/lab02_optio
nal_exercises/udp_heartbeat
$ python3 UDPHeartbeatClient.py
Message has been received :  Ping 1 21:11:29
Message has been received :  Ping 2 21:11:29
Message has been received :  Ping 3 21:11:29
Message has been received :  Ping 4 21:11:29
Message has been received :  Ping 5 21:11:29
Message has been received :  Ping 6 21:11:29
Message has been received :  Ping 7 21:11:29
Message has been received :  Ping 8 21:11:29
Message has been received :  Ping 9 21:11:29
Message has been received :  Ping 10 21:11:29
MAX RTT = 0.0002830028533935547
MIN RTT = 4.7206878662109375e-05
AVG RTT = 0.00012276172637939453
PACKET LOSS = %0.0
(base) gokhan@has:~/Desktop/HOMEWORKS/CSE 476 - MCN/Term Project/src/lab02_optio
nal_exercises/udp_heartbeat
$ 
```

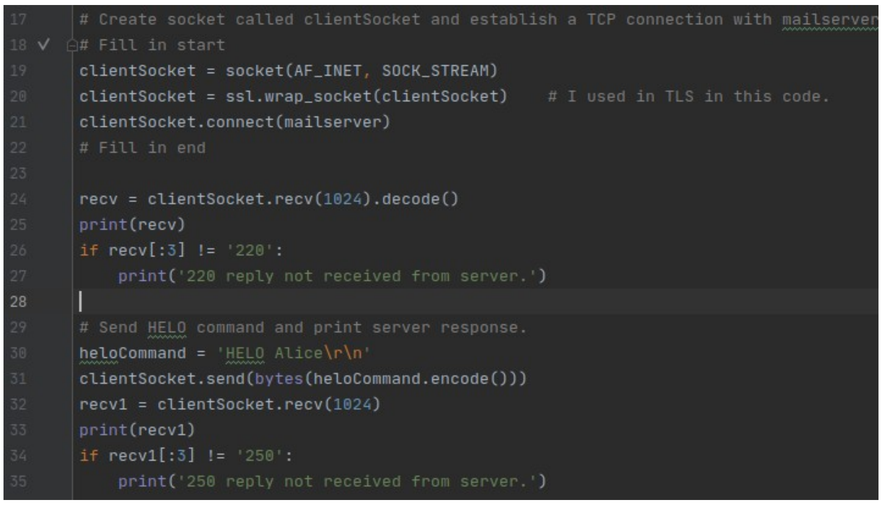
Figure 10: Heartbeat server client both are working.

Lab 3: SMTP LAB

When a message is created in your mailbox, it is delivered to the other party with the Simple Mail Transfer Protocol known as SMTP. Individuals or organizations that use mail intensively prefer to use an SMTP server to alleviate the burden on the system. When using TLS, you need to connect to port 465 (instead of port 587).

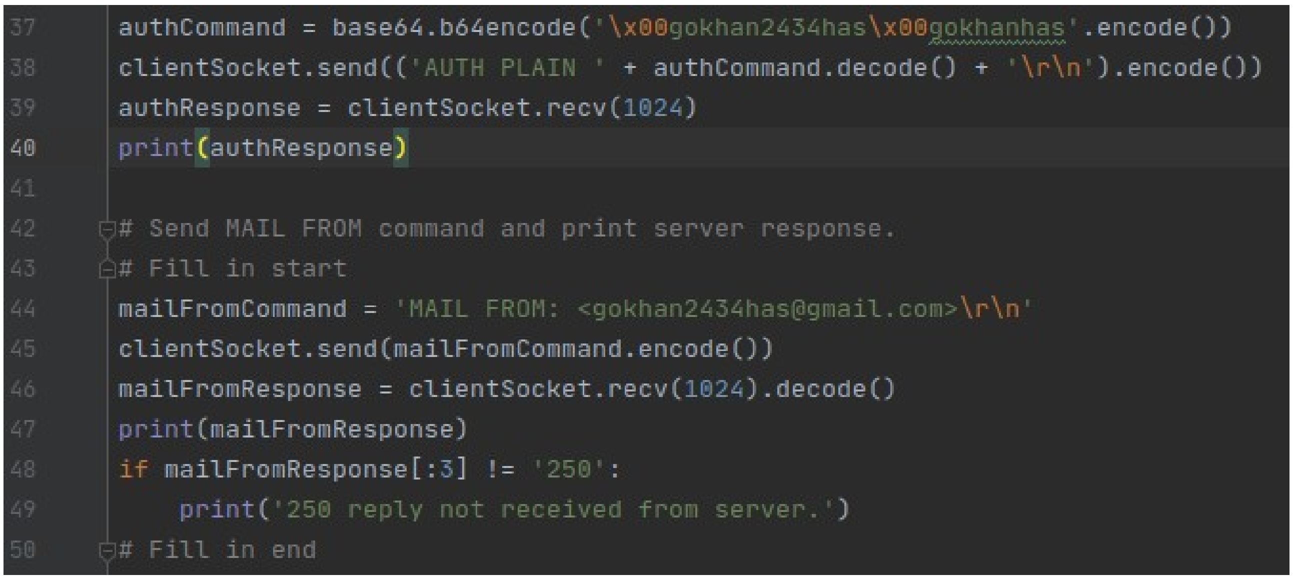


```python
### GOKHAN HAS - 161044067 ###
###### CSE 476 - LAB 03 ######

from socket import *
import ssl
import base64

msg = "\r\n I love computer networks!"
endmsg = "\r\n.\r\n"


# Choose a mail server (e.g. Google mail server) and call it mailserver
# Fill in start
mailserver = ('smtp.gmail.com', 465)    # The port TLS is 465
# Fill in end
```



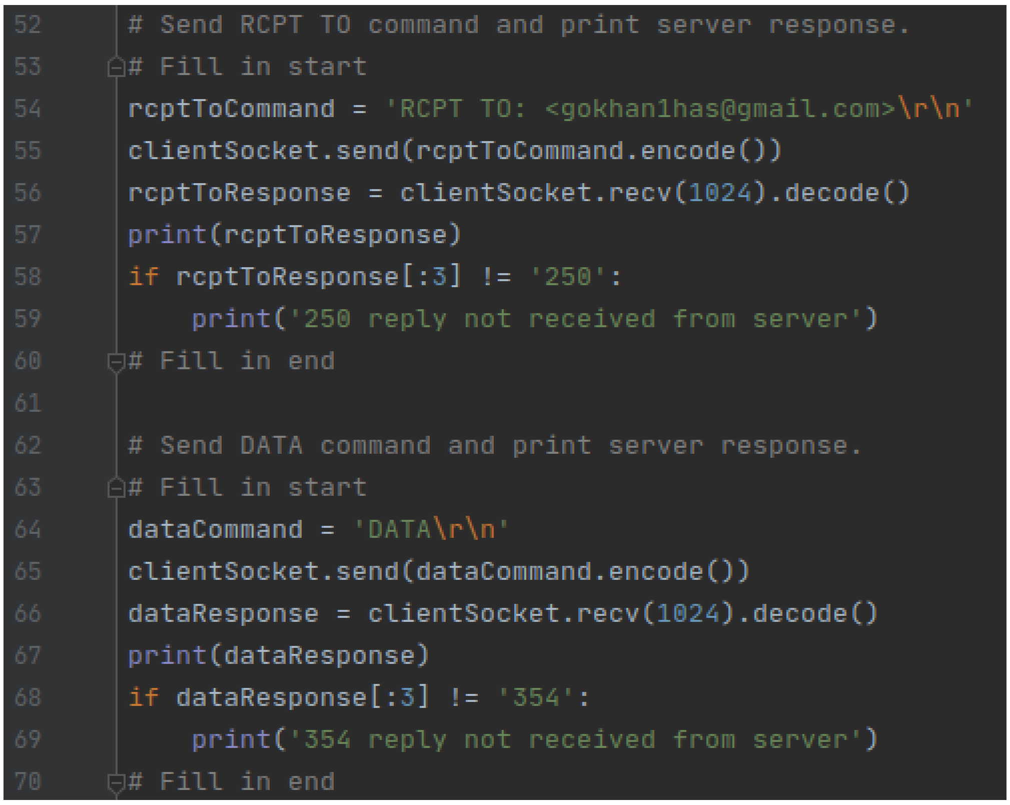
```python
# Create socket called clientSocket and establish a TCP connection with mailserver
# Fill in start
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket = ssl.wrap_socket(clientSocket)    # I used in TLS in this code.
clientSocket.connect(mailserver)
# Fill in end

recv = clientSocket.recv(1024).decode()
print(recv)
if recv[:3] != '220':
    print('220 reply not received from server.')

# Send HELO command and print server response.
heloCommand = 'HELO Alice\r\n'
clientSocket.send(bytes(heloCommand.encode()))
recv1 = clientSocket.recv(1024)
print(recv1)
if recv1[:3] != '250':
    print('250 reply not received from server.')
```

Then, in line 19, the TCP socket operation is started. SMTP needs a TCP socket. Because it is important whether the mail reaches the other party or not. Our mail server is set up with the help of ssl.wrap_socket () function and connect () function.



```
authCommand = base64.b64encode('\x00gokhan2434has\x00gokhanhas'.encode())
clientSocket.send(('AUTH PLAIN ' + authCommand.decode() + '\r\n').encode())
authResponse = clientSocket.recv(1024)
print(authResponse)

# Send MAIL FROM command and print server response.
# Fill in start
mailFromCommand = 'MAIL FROM: <gokhan2434has@gmail.com>\r\n'
clientSocket.send(mailFromCommand.encode())
mailFromResponse = clientSocket.recv(1024).decode()
print(mailFromResponse)
if mailFromResponse[:3] != '250':
    print('250 reply not received from server.')
# Fill in end
```

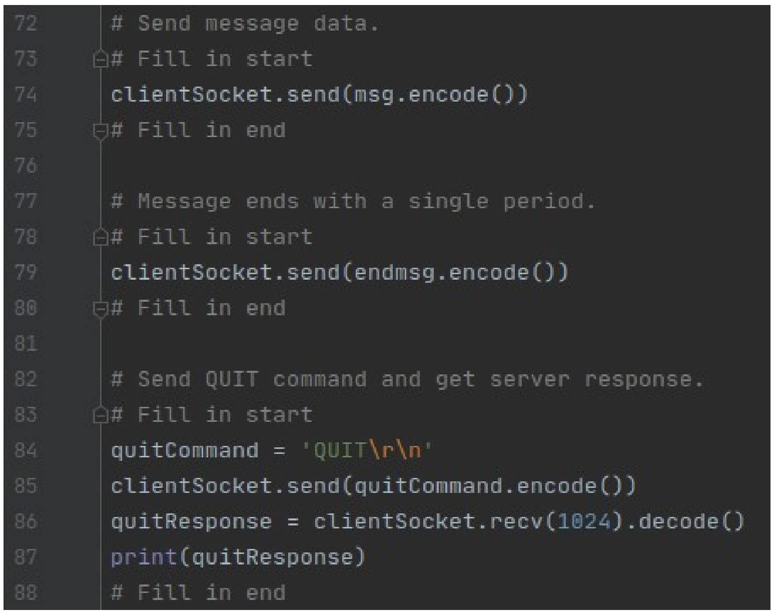
Here, the information from which e-mail to be sent is given. For example, I got the mail address gokhan2434has@gmail.com for this project. The password for this mail is "gokhanhas". Submission can be done here by changing this information. Then the MAIL FROM option is written with mailFromCommand. This information is also sent. If the code 250 returns, it means there was an error.



```
# Send RCPT TO command and print server response.
# Fill in start
rcptToCommand = 'RCPT TO: <gokhan1has@gmail.com>\r\n'
clientSocket.send(rcptToCommand.encode())
rcptToResponse = clientSocket.recv(1024).decode()
print(rcptToResponse)
if rcptToResponse[:3] != '250':
    print('250 reply not received from server')
# Fill in end

# Send DATA command and print server response.
# Fill in start
dataCommand = 'DATA\r\n'
clientSocket.send(dataCommand.encode())
dataResponse = clientSocket.recv(1024).decode()
print(dataResponse)
if dataResponse[:3] != '354':
    print('354 reply not received from server')
# Fill in end
```

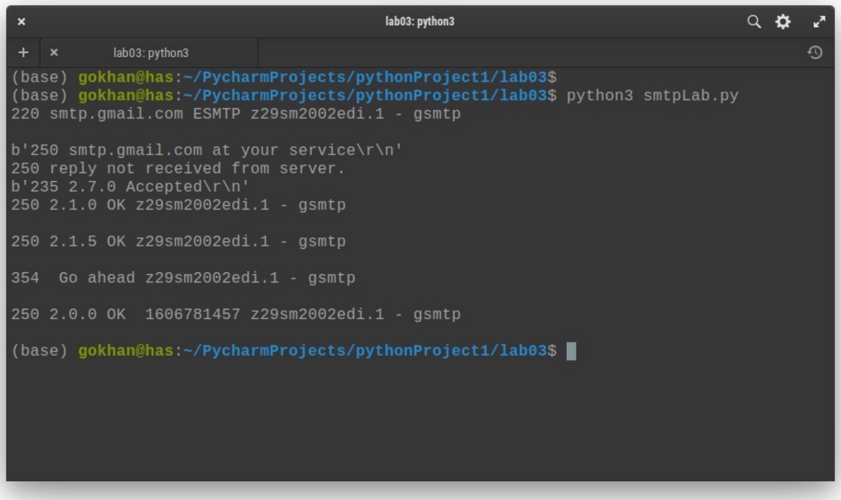
Likewise, with the RCPT TO part, it is determined to whom the mail will be sent. Here it will be sent to my other e-mail address, gokhan1has@gmail.com. This address can be sent by changing. If the 250 code returns from this block, an error has occurred.

Finally, it is time to send the DATA part. It is sent in the same way as above. If it fails only in the data section, if different from the above, 354 code is returned.



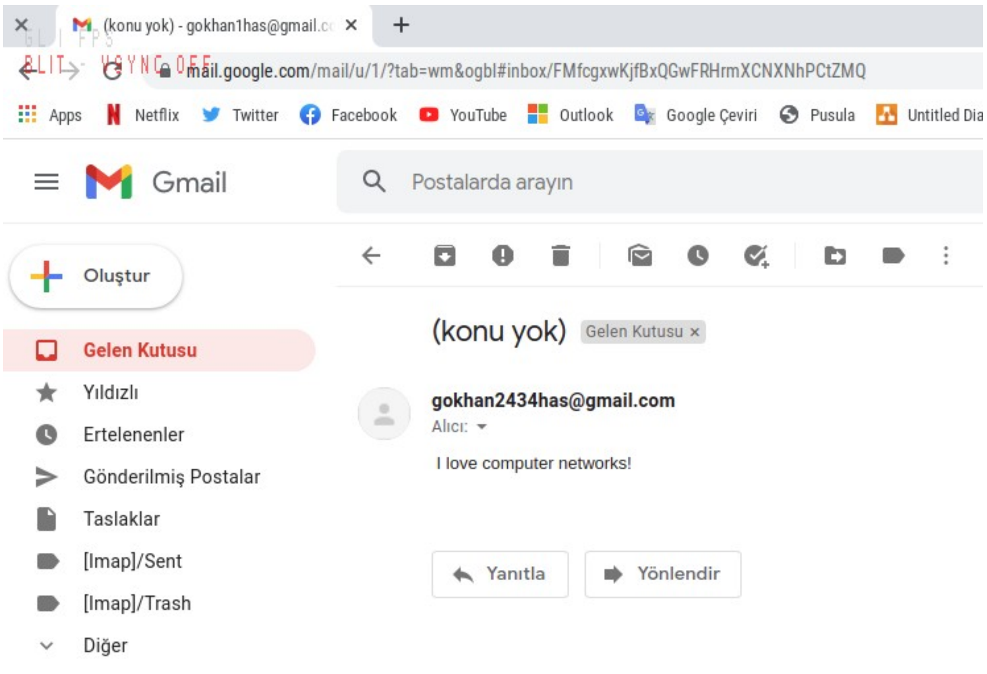
```
72      # Send message data.
73      # Fill in start
74      clientSocket.send(msg.encode())
75      # Fill in end
76
77      # Message ends with a single period.
78      # Fill in start
79      clientSocket.send(endmsg.encode())
80      # Fill in end
81
82      # Send QUIT command and get server response.
83      # Fill in start
84      quitCommand = 'QUIT\r\n'
85      clientSocket.send(quitCommand.encode())
86      quitResponse = clientSocket.recv(1024).decode()
87      print(quitResponse)
88      # Fill in end
```

The sent message and endmessage variables are sent. These operations take place in lines 74 and 79, respectively. Finally, the QUIT command is sent to the server. In this process, the other commands are similar to sending codes and the value returned from the server is written.



```
(base) gokhan@has:~/PycharmProjects/pythonProject1/lab03$
(base) gokhan@has:~/PycharmProjects/pythonProject1/lab03$ python3 smtpLab.py
220 smtp.gmail.com ESMTP z29sm2002edi.1 - gsmtp

b'250 smtp.gmail.com at your service\r\n'
250 reply not received from server.
b'235 2.7.0 Accepted\r\n'
250 2.1.0 OK z29sm2002edi.1 - gsmtp

250 2.1.5 OK z29sm2002edi.1 - gsmtp

354  Go ahead z29sm2002edi.1 - gsmtp

250 2.0.0 OK  1606781457 z29sm2002edi.1 - gsmtp

(base) gokhan@has:~/PycharmProjects/pythonProject1/lab03$
```

Figure 11: Execute the code

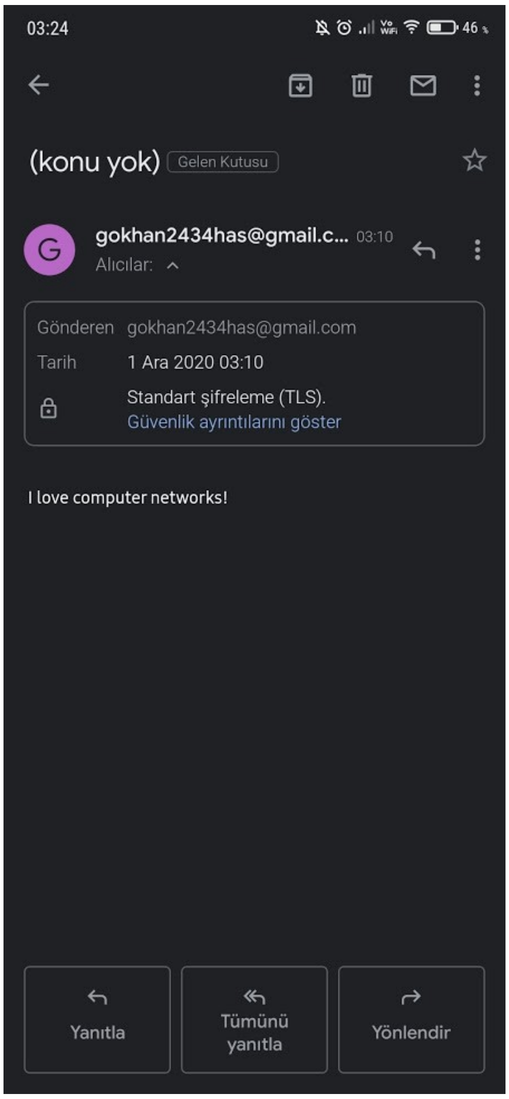


(konu yok) Gelen Kutusu

gokhan2434has@gmail.com
Alıcı:

I love computer networks!

Figure 12: Mail screenshot from web



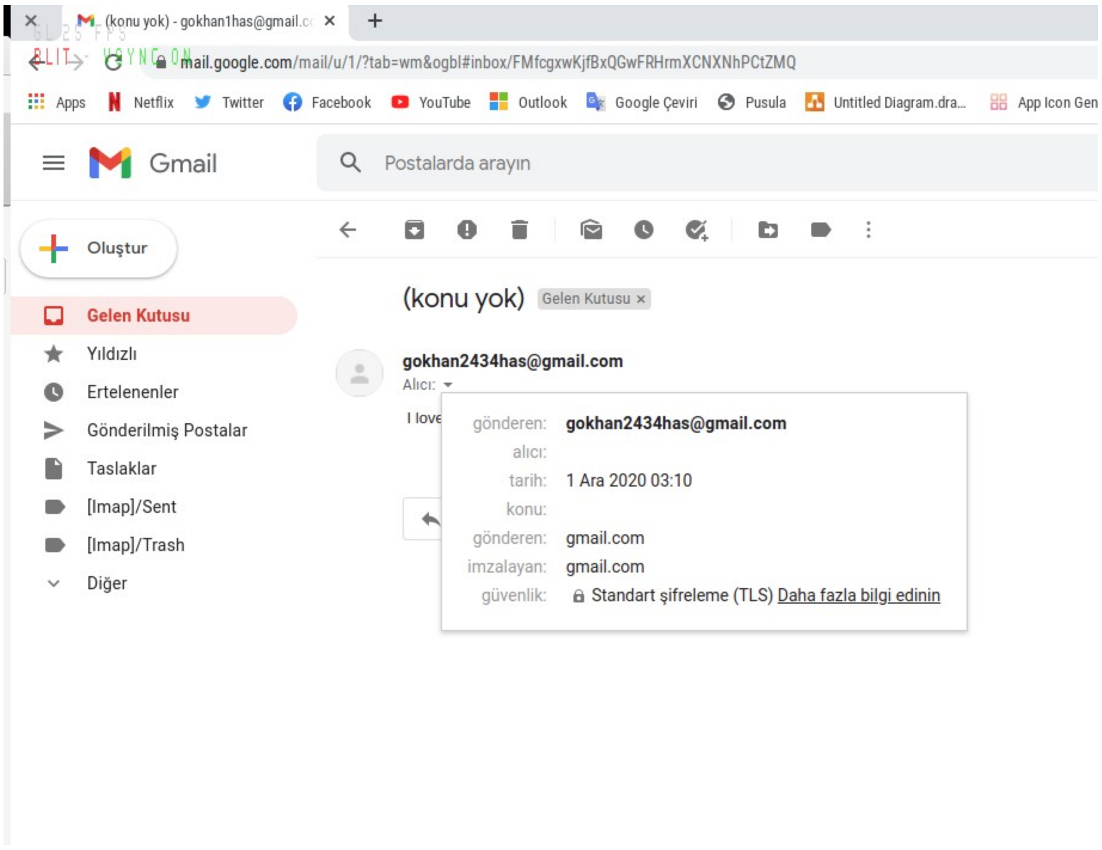
(konu yok) Gelen Kutusu

gokhan2434has@gmail.c... 03:10
Alıcılar:

Gönderen gokhan2434has@gmail.com
Tarih 1 Ara 2020 03:10
Standart şifreleme (TLS).
Güvenlik ayrıntılarını göster

I love computer networks!

Figure 13: Mail screenshot from mobile and it can be seen TLS



(konu yok) Gelen Kutusu

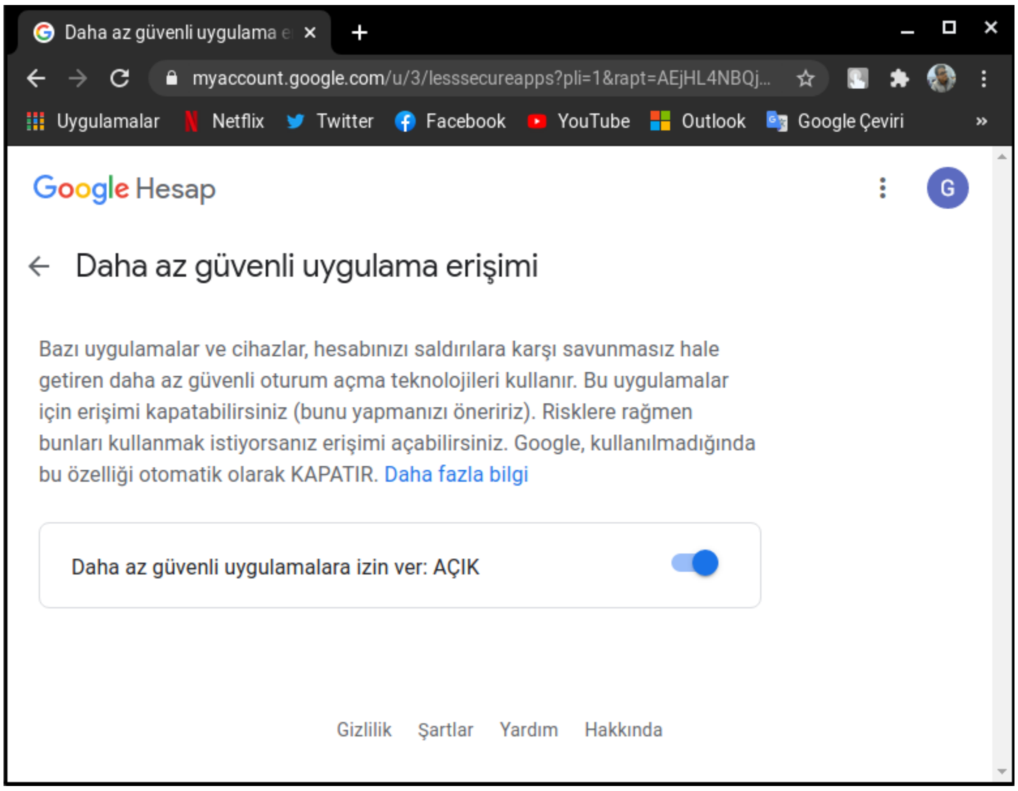
gokhan2434has@gmail.com

gönderen: gokhan2434has@gmail.com
alıcı:
tarih: 1 Ara 2020 03:10
konu:
gönderen: gmail.com
imzalayan: gmail.com
güvenlik: Standart şifreleme (TLS) Daha fazla bilgi edinin

Figure 14: Mail details screenshot from web

The optional first batch of Lab 3 was made as can be seen, but the second batch could not be made. For screenshots to be easily viewed, there are all screenshots in the extra folder.

IMPORTANT NOTE : In order to send mail from Gmail, the following setting must be turned on in the account to be sent.



Daha az güvenli uygulama erişimi

Bazı uygulamalar ve cihazlar, hesabınızı saldırılara karşı savunmasız hale getiren daha az güvenli oturum açma teknolojileri kullanır. Bu uygulamalar için erişimi kapatabilirsiniz (bunu yapmanızı öneririz). Risklere rağmen bunları kullanmak istiyorsanız erişimi açabilirsiniz. Google, kullanılmadığında bu özelliği otomatik olarak KAPATIR. Daha fazla bilgi

Daha az güvenli uygulamalara izin ver: AÇIK