CSE 476 MOBILE COMMUNICATION NETWORKS

FINAL REPORT

2021.01.08

Ebru KARDAŞ

141044049

LAB 1: WEB SERVER

For this lab, it is wanted to develop a simple Web server in Python that is capable of processing only one request. The web server will capable of:

i. create a connection socket when contacted by a client (by browser);
ii. receive the HTTP request from this connection;
iii. parse the request to determine the specific file being requested;
iv. get the requested file from the server's file system;
v. create an HTTP response message consisting of the requested file preceded by header lines; and
vi. 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.

The code skeleton is given from the pdf as below:

```
#import socket module
from socket import *
serverSocket = socket(AF_INET, SOCK_STREAM)
#Prepare a server socket
#Fill in start #Fill in end

while True:
    #Establish the connection
    print 'Ready to serve...'
    connectionSocket, addr = #Fill in start #Fill in end
    try:
        message  = #Fill in start #Fill in end
        filename = message.split()[1]
        f = open(filename[1:])
        outputdata = #Fill in start #Fill in end
        #Send one HTTP header line into socket
        #Fill in start
        #Fill in end
        #Send the content of the requested file to the client
        for i in range(0, len(outputdata)):
            connectionSocket.send(outputdata[i])
        connectionSocket.close()
    except IOError:
        #Send response message for file not found
        #Fill in start #Fill in end
        #Close client socket
        #Fill in start #Fill in end
serverSocket.close()
```

Server code, the **#Fill in start #Fill in end** places are needed to be completed so this is how completed.

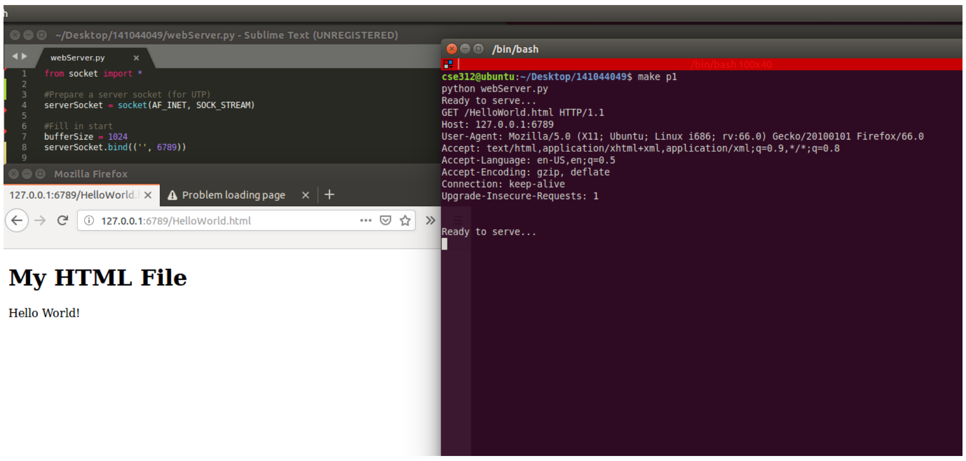
```python
from socket import *

#Prepare a server socket (for UTP)
serverSocket = socket(AF_INET, SOCK_STREAM)

#Fill in start
bufferSize = 1024
serverSocket.bind(('', 6789))

# listen one client each time
serverSocket.listen(1)

#Fill in end

while True:
    #Establish the connection
    print("Ready to serve...")

    #Accept the -one and only- client
    connectionSocket, addr = serverSocket.accept() #Fill in start #Fill in end

    try:

        message = connectionSocket.recv(bufferSize)

        #Fill in start

        # Print the message from client
        print(message)

        #Fill in end

        filename = message.split()[1]
        f = open(filename[1:])

        # Prepare the message to send
        outputdata = f.read() #Fill in start    #Fill in end

        #Send one HTTP header line into socket

        #Fill in start

        # Send to client that found
        connectionSocket.send("\nHTTP/1.x 200 OK\r\n\r\n")

        #Fill in end

        #Send the content of the requested file to the client
        for i in range(0, len(outputdata)):
            connectionSocket.send(outputdata[i])
        connectionSocket.close()
    except IOError:
        #Send response message for file not found

        #Fill in start

        print("404 NOT FOUND")
        # Typical ups 404 not found error

        connectionSocket.send("\nHTTP/1.x 404 Not Found\r\n\r\n")
        connectionSocket.send("<html><head></head><body><h1>404 Not Found</h1>
</body></html>\r\n")

        #Fill in end

        #Close client socket

        #Fill in start
        connectionSocket.close()
        #Fill in end
    except:

        print("Ups another issue happened")
        connectionSocket.close()


# 127.0.0.1:6789/HelloWorld.html
```

Output

Project is tested on VM, operating system Ubuntu 14.04 with browser Firefox with shown parameters.

Output/screenshots are shown below:

Test 1:


~/Desktop/141044049/webServer.py - Sublime Text (UNREGISTERED)

```python
from socket import *

#Prepare a server socket (for UTP)
serverSocket = socket(AF_INET, SOCK_STREAM)

#Fill in start
bufferSize = 1024
serverSocket.bind(('', 6789))

```

Mozilla Firefox — 127.0.0.1:6789/HelloWorld.html

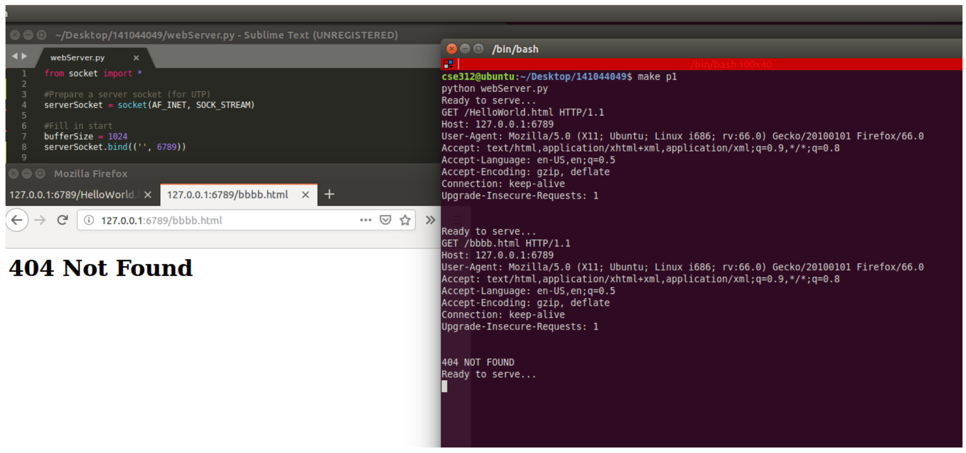
My HTML File

Hello World!

/bin/bash

```
cse312@ubuntu:~/Desktop/141044049$ make p1
python webServer.py
Ready to serve...
GET /HelloWorld.html HTTP/1.1
Host: 127.0.0.1:6789
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:66.0) Gecko/20100101 Firefox/66.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1


Ready to serve...
```

Test 2: Page Not Found


~/Desktop/141044049/webServer.py - Sublime Text (UNREGISTERED)

```python
from socket import *

#Prepare a server socket (for UTP)
serverSocket = socket(AF_INET, SOCK_STREAM)

#Fill in start
bufferSize = 1024
serverSocket.bind(('', 6789))

```

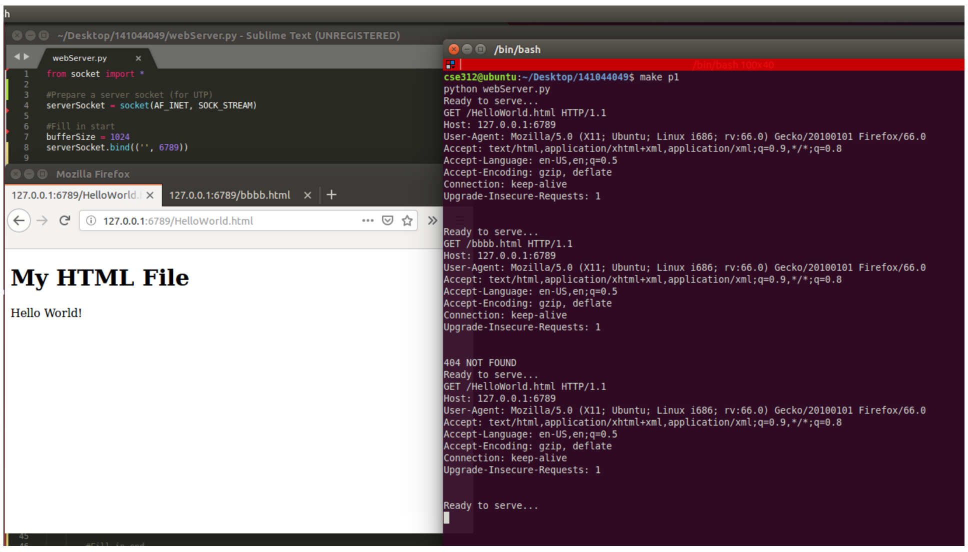
Mozilla Firefox — 127.0.0.1:6789/bbbb.html

404 Not Found

/bin/bash

```
cse312@ubuntu:~/Desktop/141044049$ make p1
python webServer.py
Ready to serve...
GET /HelloWorld.html HTTP/1.1
Host: 127.0.0.1:6789
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:66.0) Gecko/20100101 Firefox/66.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1


Ready to serve...
GET /bbbb.html HTTP/1.1
Host: 127.0.0.1:6789
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:66.0) Gecko/20100101 Firefox/66.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1


404 NOT FOUND
Ready to serve...
```

Test 3: Test 1 again


~/Desktop/141044049/webServer.py - Sublime Text (UNREGISTERED)

```python
from socket import *

#Prepare a server socket (for UTP)
serverSocket = socket(AF_INET, SOCK_STREAM)

#Fill in start
bufferSize = 1024
serverSocket.bind(('', 6789))

```

Mozilla Firefox — 127.0.0.1:6789/HelloWorld.html

My HTML File

Hello World!

/bin/bash

```
cse312@ubuntu:~/Desktop/141044049$ make p1
python webServer.py
Ready to serve...
GET /HelloWorld.html HTTP/1.1
Host: 127.0.0.1:6789
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:66.0) Gecko/20100101 Firefox/66.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1


Ready to serve...
GET /bbbb.html HTTP/1.1
Host: 127.0.0.1:6789
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:66.0) Gecko/20100101 Firefox/66.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1


404 NOT FOUND
Ready to serve...
GET /HelloWorld.html HTTP/1.1
Host: 127.0.0.1:6789
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:66.0) Gecko/20100101 Firefox/66.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1


Ready to serve...
```

LAB 2: UDP PINGER

For this lab, it is wanted us learn the basics of socket programming for UDP in Python, how to send and receive datagram packets using UDP sockets and also, how to set a proper socket timeout.

First studied a simple Internet ping server written in the Python, and implement a corresponding client. The programs use a simpler protocol, UDP to communicate with each other. The ping protocol allows a client machine to send a packet of data to a remote machine, and have the remote machine return the data back to the client unchanged (an action referred to as echoing). Among other uses, the ping protocol allows hosts to determine round-trip times to other machines.

It is given the complete code for the Ping server below:

```
# UDPPingerServer.py

# We will need the following module to generate randomized lost packets
import random
from socket import *
# Create a UDP socket

# Notice the use of SOCK_DGRAM for UDP packets
serverSocket = socket(AF_INET, SOCK_DGRAM)
# Assign IP address and port number to socket
serverSocket.bind(('', 12000))

while True:
    # Generate random number in the range of 0 to 10
    rand = random.randint(0, 10)
    # Receive the client packet along with the address it is coming from
    message, address = serverSocket.recvfrom(1024)
    # Capitalize the message from the client
    message = message.upper()
    # If rand is less is than 4, we consider the packet lost and do not respon
d
    if rand < 4:
        continue
    # Otherwise, the server responds
    serverSocket.sendto(message, address)
```

Ping client should send **10 pings** to the server. The client wait up to **one second for a reply**; if no reply is received within one second, the client program assumed that the packet was lost during transmission across the network.

Specifically, the client program

i. send the ping message using UDP
ii. print the response message from server, if any
iii. calculate and print the round trip time (RTT), in seconds, of each packet, if server responses
iv. otherwise, print "Request timed out"

During development, The UDPPingerServer.py is run on the machine, and tested the client by sending packets to *localhost* (or, 127.0.0.1).

The ping messages in this lab are formatted in a simple way. The client message is one line, consisting of ASCII characters in the following format:

Ping *sequence_number time*

where *sequence_number* starts **at 1** and progresses to 10 for each successive ping message sent by the client, and *time* is the time when the client sends the message.

Ping Client code is shown below:

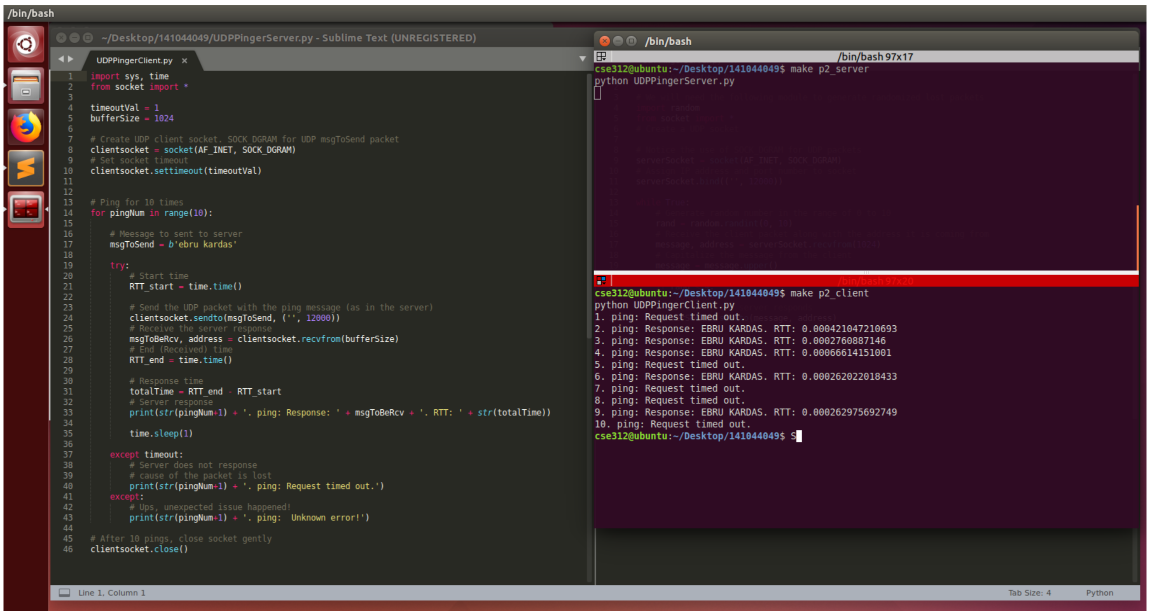
```python
import sys, time
from socket import *

timeoutVal = 1
bufferSize = 1024

# Create UDP client socket. SOCK_DGRAM for UDP msgToSend packet
clientsocket = socket(AF_INET, SOCK_DGRAM)
# Set socket timeout
clientsocket.settimeout(timeoutVal)


# Ping for 10 times
for pingNum in range(10):

    # Meesage to sent to server
    msgToSend = b'ebru kardas'

    try:
        # Start time
        RTT_start = time.time()

        # Send the UDP packet with the ping message (as in the server)
        clientsocket.sendto(msgToSend, ('', 12000))
        # Receive the server response
        msgToBeRcv, address = clientsocket.recvfrom(bufferSize)
        # End (Received) time
        RTT_end = time.time()
        # Response time
        totalTime = RTT_end - RTT_start
        # Server response
        print(str(pingNum+1) + '. ping: Response: ' + msgToBeRcv + '. RTT: ' + str(totalTime))

        time.sleep(1)

    except timeout:
        # Server does not response
        # cause of the packet is lost
        print(str(pingNum+1) + '. ping: Request timed out.')
    except:
        # Ups, unexpected issue happened!
        print(str(pingNum+1) + '. ping:  Unknown error!')

# After 10 pings, close socket gently
clientsocket.close()
```

Output

Project is tested on VM, operating system Ubuntu 14.04 with browser Firefox with shown parameters.

Output/screenshots are shown below:

Test 1:



/bin/bash

~/Desktop/141044049/UDPPingerServer.py - Sublime Text (UNREGISTERED)

UDPPingerClient.py

```python
import sys, time
from socket import *

timeoutVal = 1
bufferSize = 1024

# Create UDP client socket. SOCK_DGRAM for UDP msgToSend packet
clientsocket = socket(AF_INET, SOCK_DGRAM)
# Set socket timeout
clientsocket.settimeout(timeoutVal)


# Ping for 10 times
for pingNum in range(10):

    # Meesage to sent to server
    msgToSend = b'ebru kardas'

    try:
        # Start time
        RTT_start = time.time()

        # Send the UDP packet with the ping message (as in the server)
        clientsocket.sendto(msgToSend, ('', 12000))
        # Receive the server response
        msgToBeRcv, address = clientsocket.recvfrom(bufferSize)
        # End (Received) time
        RTT_end = time.time()

        # Response time
        totalTime = RTT_end - RTT_start
        # Server response
        print(str(pingNum+1) + '. ping: Response: ' + msgToBeRcv + '. RTT: ' + str(totalTime))

        time.sleep(1)

    except timeout:
        # Server does not response
        # cause of the packet is lost
        print(str(pingNum+1) + '. ping: Request timed out.')
    except:
        # Ups, unexpected issue happened!
        print(str(pingNum+1) + '. ping:  Unknown error!')

# After 10 pings, close socket gently
clientsocket.close()
```

Line 1, Column 1

Tab Size: 4

Python

/bin/bash

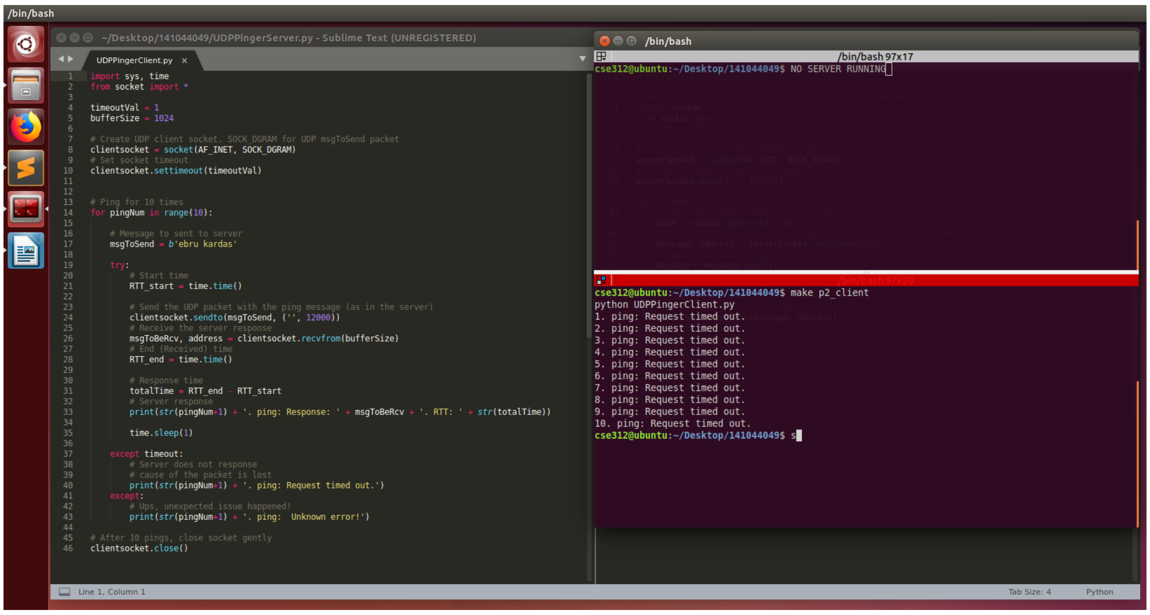
/bin/bash 97x17

```
cse312@ubuntu:~/Desktop/141044049$ make p2_server
python UDPPingerServer.py
```

/bin/bash 97x20

```
cse312@ubuntu:~/Desktop/141044049$ make p2_client
python UDPPingerClient.py
1. ping: Request timed out.
2. ping: Response: EBRU KARDAS. RTT: 0.000421047210693
3. ping: Response: EBRU KARDAS. RTT: 0.0002760887146
4. ping: Response: EBRU KARDAS. RTT: 0.00066614151001
5. ping: Request timed out.
6. ping: Response: EBRU KARDAS. RTT: 0.000262022018433
7. ping: Request timed out.
8. ping: Request timed out.
9. ping: Response: EBRU KARDAS. RTT: 0.000262975692749
10. ping: Request timed out.
cse312@ubuntu:~/Desktop/141044049$ S
```

Test 2: Server not running



/bin/bash

~/Desktop/141044049/UDPPingerServer.py - Sublime Text (UNREGISTERED)

UDPPingerClient.py

```python
import sys, time
from socket import *

timeoutVal = 1
bufferSize = 1024

# Create UDP client socket. SOCK_DGRAM for UDP msgToSend packet
clientsocket = socket(AF_INET, SOCK_DGRAM)
# Set socket timeout
clientsocket.settimeout(timeoutVal)


# Ping for 10 times
for pingNum in range(10):

    # Meesage to sent to server
    msgToSend = b'ebru kardas'

    try:
        # Start time
        RTT_start = time.time()

        # Send the UDP packet with the ping message (as in the server)
        clientsocket.sendto(msgToSend, ('', 12000))
        # Receive the server response
        msgToBeRcv, address = clientsocket.recvfrom(bufferSize)
        # End (Received) time
        RTT_end = time.time()

        # Response time
        totalTime = RTT_end - RTT_start
        # Server response
        print(str(pingNum+1) + '. ping: Response: ' + msgToBeRcv + '. RTT: ' + str(totalTime))

        time.sleep(1)

    except timeout:
        # Server does not response
        # cause of the packet is lost
        print(str(pingNum+1) + '. ping: Request timed out.')
    except:
        # Ups, unexpected issue happened!
        print(str(pingNum+1) + '. ping:  Unknown error!')

# After 10 pings, close socket gently
clientsocket.close()
```

Line 1, Column 1 — Tab Size: 4 — Python

/bin/bash

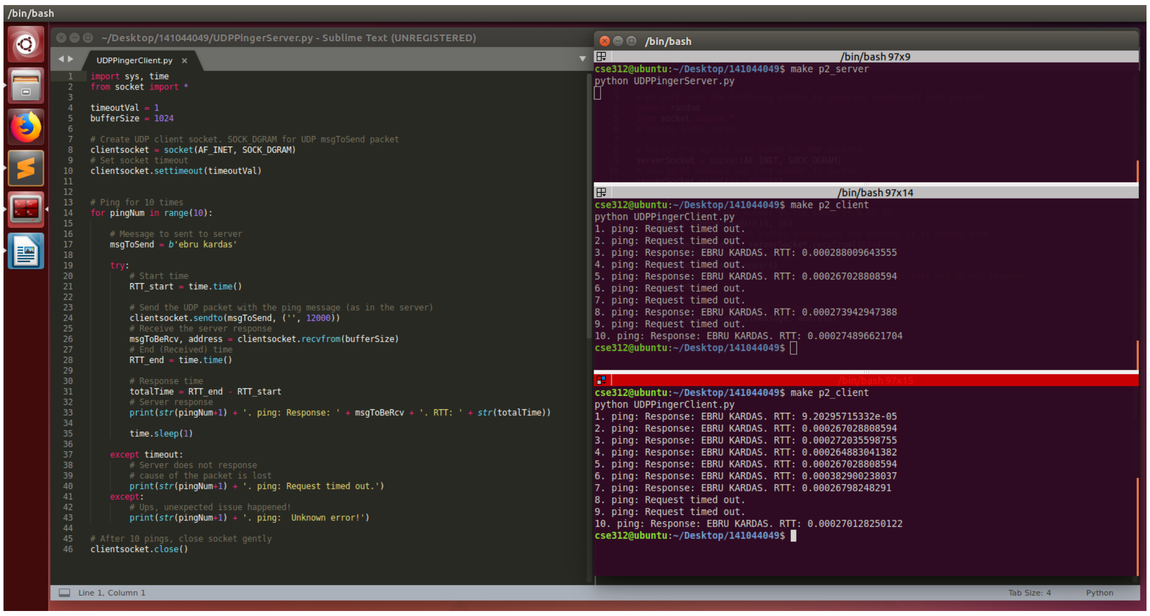
/bin/bash 97x17

```
cse312@ubuntu:~/Desktop/141044049$ NO SERVER RUNNING
```

/bin/bash 97x20

```
cse312@ubuntu:~/Desktop/141044049$ make p2_client
python UDPPingerClient.py
1. ping: Request timed out.
2. ping: Request timed out.
3. ping: Request timed out.
4. ping: Request timed out.
5. ping: Request timed out.
6. ping: Request timed out.
7. ping: Request timed out.
8. ping: Request timed out.
9. ping: Request timed out.
10. ping: Request timed out.
cse312@ubuntu:~/Desktop/141044049$ s
```

Test 3: 2 clients running at the same time



/bin/bash

~/Desktop/141044049/UDPPingerServer.py - Sublime Text (UNREGISTERED)

UDPPingerClient.py

```python
import sys, time
from socket import *

timeoutVal = 1
bufferSize = 1024

# Create UDP client socket. SOCK_DGRAM for UDP msgToSend packet
clientsocket = socket(AF_INET, SOCK_DGRAM)
# Set socket timeout
clientsocket.settimeout(timeoutVal)


# Ping for 10 times
for pingNum in range(10):

    # Meesage to sent to server
    msgToSend = b'ebru kardas'

    try:
        # Start time
        RTT_start = time.time()

        # Send the UDP packet with the ping message (as in the server)
        clientsocket.sendto(msgToSend, ('', 12000))
        # Receive the server response
        msgToBeRcv, address = clientsocket.recvfrom(bufferSize)
        # End (Received) time
        RTT_end = time.time()

        # Response time
        totalTime = RTT_end - RTT_start
        # Server response
        print(str(pingNum+1) + '. ping: Response: ' + msgToBeRcv + '. RTT: ' + str(totalTime))

        time.sleep(1)

    except timeout:
        # Server does not response
        # cause of the packet is lost
        print(str(pingNum+1) + '. ping: Request timed out.')
    except:
        # Ups, unexpected issue happened!
        print(str(pingNum+1) + '. ping:  Unknown error!')

# After 10 pings, close socket gently
clientsocket.close()
```

Line 1, Column 1 — Tab Size: 4 — Python

/bin/bash

/bin/bash 97x9

```
cse312@ubuntu:~/Desktop/141044049$ make p2_server
python UDPPingerServer.py
```

/bin/bash 97x14

```
cse312@ubuntu:~/Desktop/141044049$ make p2_client
python UDPPingerClient.py
1. ping: Request timed out.
2. ping: Request timed out.
3. ping: Response: EBRU KARDAS. RTT: 0.000288009643555
4. ping: Request timed out.
5. ping: Response: EBRU KARDAS. RTT: 0.000267028808594
6. ping: Request timed out.
7. ping: Request timed out.
8. ping: Response: EBRU KARDAS. RTT: 0.000273942947388
9. ping: Request timed out.
10. ping: Response: EBRU KARDAS. RTT: 0.000274896621704
cse312@ubuntu:~/Desktop/141044049$
```

/bin/bash 97x15

```
cse312@ubuntu:~/Desktop/141044049$ make p2_client
python UDPPingerClient.py
1. ping: Response: EBRU KARDAS. RTT: 9.20295715332e-05
2. ping: Response: EBRU KARDAS. RTT: 0.000267028808594
3. ping: Response: EBRU KARDAS. RTT: 0.000272035598755
4. ping: Response: EBRU KARDAS. RTT: 0.000264883041382
5. ping: Response: EBRU KARDAS. RTT: 0.000267028808594
6. ping: Response: EBRU KARDAS. RTT: 0.000382900238037
7. ping: Response: EBRU KARDAS. RTT: 0.00026798248291
8. ping: Request timed out.
9. ping: Request timed out.
10. ping: Response: EBRU KARDAS. RTT: 0.000270128250122
cse312@ubuntu:~/Desktop/141044049$
```

LAB 3: MAIL CLIENT (SMTP)

By the end of this lab, you will have acquired a better understanding of SMTP protocol. You will also gain experience in implementing a standard protocol using Python.

Your task is to develop a simple mail client that sends email to any recipient. Your client will need to connect to a mail server, dialogue with the mail server using the SMTP protocol, and send an email message to the mail server. Python provides a module, called smtplib, which has built in methods to send mail using SMTP protocol. However, we will not be using this module in this lab, because it hide the details of SMTP and socket programming.

In order to limit spam, some mail servers do not accept TCP connection from arbitrary sources. For the experiment described below, you may want to try connecting both to your university mail server and to a popular Webmail server, such as a AOL mail server. You may also try making your connection both from your home and from your university campus.

Skeleton Python Code for the Mail Client

```
from socket import *
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
mailserver = #Fill in start #Fill in end
# Create socket called clientSocket and establish a TCP connection with mailserver
#Fill in start #Fill in end
recv = clientSocket.recv(1024)
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(heloCommand)
recv1 = clientSocket.recv(1024)
print recv1
if recv1[:3] != '250':
        print '250 reply not received from server.'
# Send MAIL FROM command and print server response.
# Fill in start # Fill in end
# Send RCPT TO command and print server response.
# Fill in start # Fill in end
# Send DATA command and print server response.
# Fill in start # Fill in end
# Send message data.
# Fill in start # Fill in end
# Message ends with a single period.
# Fill in start # Fill in end
# Send QUIT command and get server response.
# Fill in start # Fill in end
```

```python
from socket import *

import ssl
import base64
from email.mime.text import MIMEText

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
mailserver = 'smtp.gmail.com'#Fill in start
mailPort = 587                  # port integer 587 25 465
endmsg2 = '\r\n'

#Fill in end
# Create socket called clientSocket and establish a TCP connection with mailse
rver
#Fill in start
# Create socket called clientSocket and establish
# a TCP connection with gmailserver
clientSocket = socket(AF_INET, SOCK_STREAM) # IPv4 internet protocols
clientSocket.connect((mailserver,mailPort))
#Fill in end
recv = clientSocket.recv(1024)
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(heloCommand.encode())
recv = clientSocket.recv(1024).decode()
print recv
if recv[:3] != '250':
    print '250 reply not received from server.'
# Send MAIL FROM command and print server response.
# Fill in start

tlsCommand = 'STARTTLS\r\n'
clientSocket.send(tlsCommand.encode())
recv = clientSocket.recv(1024).decode()
print recv
if recv[:3] != '220':
    print '220 reply not received from server.'
# Add SSL after the STARTTLS command
clientSocket = ssl.wrap_socket(clientSocket)

authCommand = 'AUTH LOGIN\r\n'
clientSocket.write(authCommand.encode())
recv = clientSocket.recv(1024).decode()
print recv
```

```
if recv[:3] != '334':
    print '334 reply not received from server.'

mail = "deneme1530@gmail.com"
psw = "EK123456"

clientSocket.write((str(base64.b64encode(mail.encode())).encode("utf-8")
                + endmsg2).encode())
recv = clientSocket.recv(1024).decode()
print recv
if recv[:3] != '334':
    print '334 reply not received from server.'

clientSocket.write((str(base64.b64encode(psw.encode())).encode("utf-8")
                + endmsg2).encode())
recv = clientSocket.recv(1024).decode()
print recv
if recv[:3] != '235':
    print '235 reply not received from server.'

mailFromCmd = 'MAIL FROM: <' + mail + '>' + endmsg2
clientSocket.send(mailFromCmd.encode())
recv = clientSocket.recv(1024).decode()
print recv
if recv[:3] != '250':   # if first 3 chars is not '250'
    print 'ERROR: 7 250 reply not received from server.\n'
# Fill in end
# Send RCPT TO command and print server response.
# Fill in start
cmd = ('RCPT TO: <' + mail + '>\r\n').encode()
clientSocket.send(cmd)
recv = clientSocket.recv(1024)
print recv
if recv[:3] != '250':   # if first 3 chars is not '250'
    print '250 reply not received from server.\n'
# Fill in end
# Send DATA command and print server response.
# Fill in start
dataCommand = 'Data\r\n'
clientSocket.send(dataCommand.encode())
recv = clientSocket.recv(1024).decode()
print recv
if recv[:3] != '354':   # if first 3 chars is not '250'
    print '354 reply not received from server.\n'
# Fill in end
# Send message data.
# Fill in start

# for message
```

```
clientSocket.send(msg.encode() + endmsg.encode())
#for image
#imgFName = 'https://www.network.com.tr/assets/v2/img/Network.jpg'
#imageMsg = MIMEText('<html><body><h1>Hello</h1>' +
#                    '<p><img src="'+ imgFName + '"></p>' +
#                    '</body></html>', 'html', 'utf-8')
#clientSocket.send(imageMsg.as_string().encode() + endmsg.encode())
#print imageMsg.as_string().encode()


# Fill in end
# Message ends with a single period.
# Fill in start
recv = clientSocket.recv(1024).decode()
print recv
if recv[:3] != '250':   # if first 3 chars is not '250'
    print '250 reply not received from server.\n'

# Fill in end
# Send QUIT command and get server response.
# Fill in start
quitCommand = 'Quit\r\n'
clientSocket.send(quitCommand.encode())
recv = clientSocket.recv(1024).decode()
print recv
if recv[:3] != '221':   # if first 3 chars is not '250'
    print '221 reply not received from server.\n'
# Fill in end

exit()

# text and image
```

Problems

- **501 5.1.7 Invalid address**
  The server disagrees with you about whether the address is valid.
  sender = "<kisiselgorsel@gmail.com>"
  psw = "D1996b18"
  #'MAIL FROM:<\x00ekardas@gtu.edu.tr\x00D1996b18>\r\n'*
  mailFromCmd = 'MAIL FROM:' + sender + "\x00" + psw + endmsg

- **530-5.7.0 Authentication Required.**
  'AUTH LOGIN' command is needed
- Even whitespaces in commands effects connection like the whitespace after '<' char.
  'RCPT TO: < x@gmail.com>\r\n'

Solutions

- Optional Exercise 1: Add TLS/SSL commands to your existing ones.

```
tlsCommand = 'STARTTLS\r\n'
clientSocket.send(tlsCommand.encode())
recv = clientSocket.recv(1024).decode()
print recv
if recv[:3] != '220':
    print '220 reply not received from server.'
# Add SSL after the STARTTLS command
clientSocket = ssl.wrap_socket(clientSocket)
```

- Optional Exercise 2:
  For image:
  from email.mime.text import MIMEText

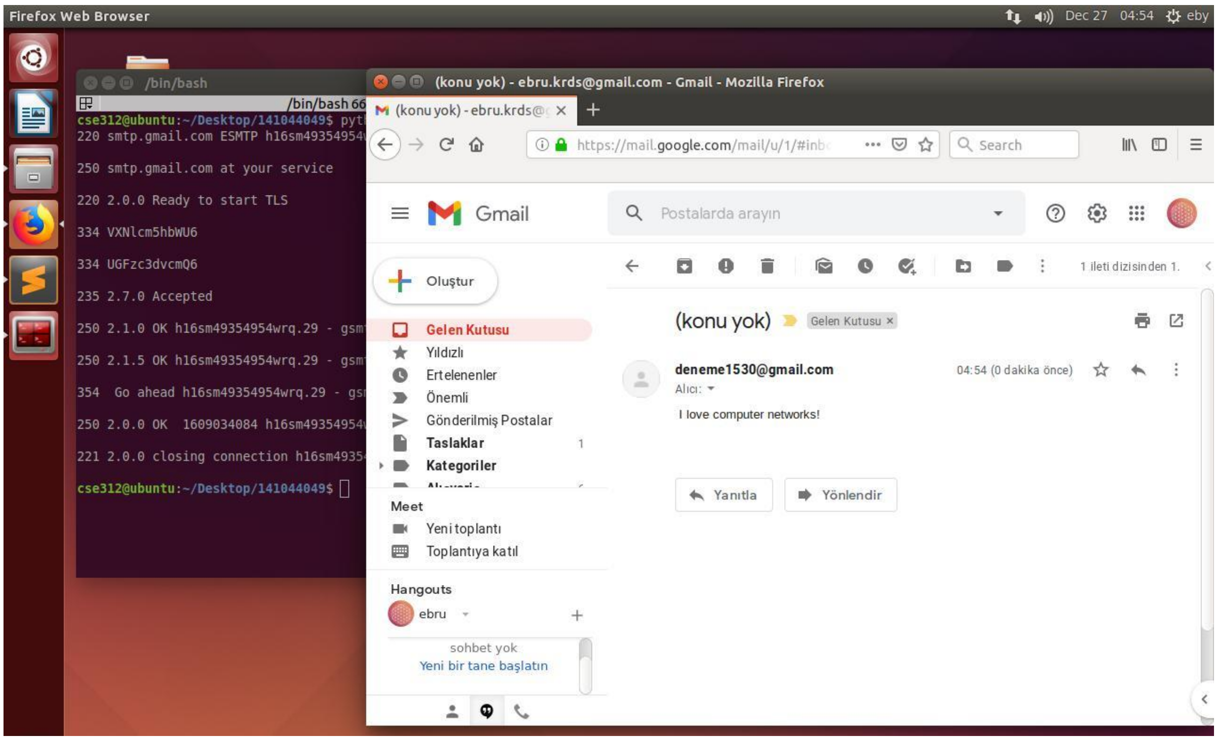
```
#for image
imgFName = 'https://www.network.com.tr/assets/v2/img/Network.jpg'
imageMsg = MIMEText('<html><body><h1>Hello</h1>' +
                    '<p><img src="'+ imgFName + '"></p>' +
                    '</body></html>', 'html', 'utf-8')
clientSocket.send(imageMsg.as_string().encode() + endmsg.encode())
#print imageMsg.as_string().encode()
```

Output

Project is tested on VM, operating system Ubuntu 14.04 with shown parameters.

Output/screenshots are shown below:

Message sent:



```
cse312@ubuntu:~/Desktop/141044049$ pyt
220 smtp.gmail.com ESMTP h16sm49354954

250 smtp.gmail.com at your service

220 2.0.0 Ready to start TLS

334 VXNlcm5hbWU6

334 UGFzc3dvcmQ6

235 2.7.0 Accepted

250 2.1.0 OK h16sm49354954wrq.29 - gsm

250 2.1.5 OK h16sm49354954wrq.29 - gsm

354  Go ahead h16sm49354954wrq.29 - gs

250 2.0.0 OK  1609034084 h16sm49354954

221 2.0.0 closing connection h16sm4935

cse312@ubuntu:~/Desktop/141044049$
```

(konu yok) - ebru.krds@gmail.com - Gmail - Mozilla Firefox

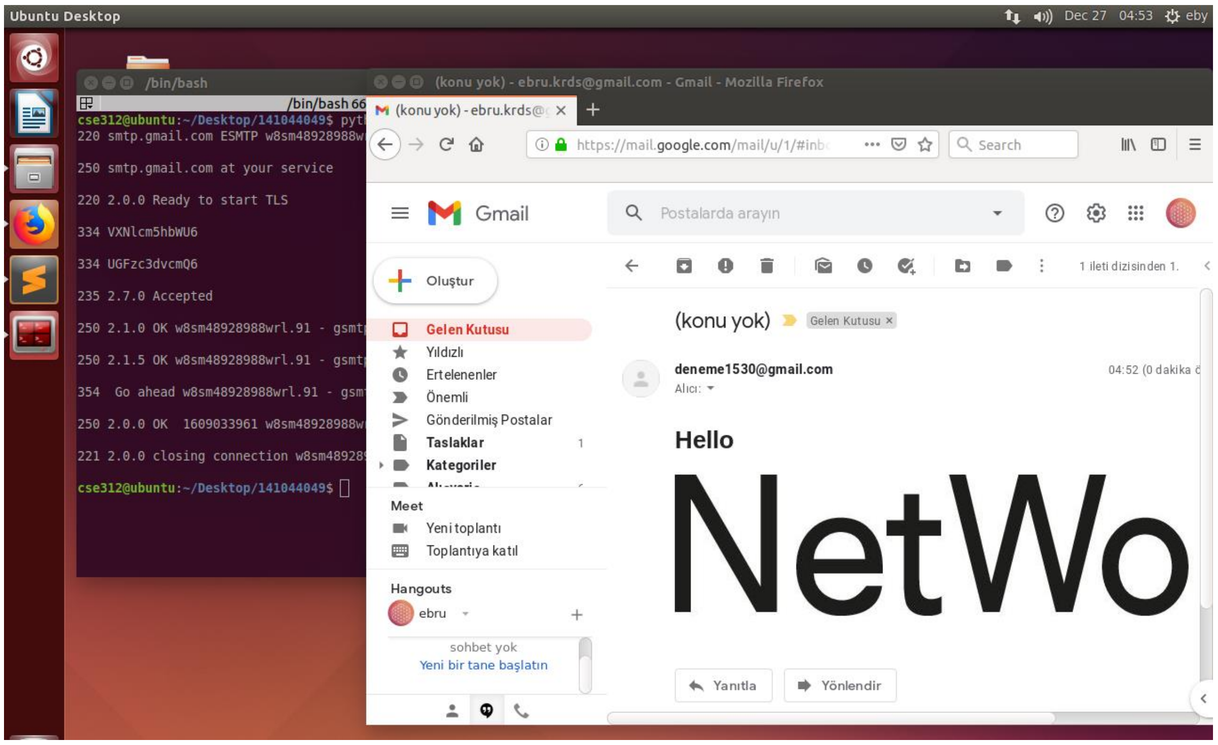
https://mail.google.com/mail/u/1/#inb

(konu yok) Gelen Kutusu ×

deneme1530@gmail.com 04:54 (0 dakika önce)
Alıcı:

I love computer networks!

Image sent:



```
cse312@ubuntu:~/Desktop/141044049$ pyt
220 smtp.gmail.com ESMTP w8sm48928988w

250 smtp.gmail.com at your service

220 2.0.0 Ready to start TLS

334 VXNlcm5hbWU6

334 UGFzc3dvcmQ6

235 2.7.0 Accepted

250 2.1.0 OK w8sm48928988wrl.91 - gsmt

250 2.1.5 OK w8sm48928988wrl.91 - gsmt

354  Go ahead w8sm48928988wrl.91 - gsm

250 2.0.0 OK  1609033961 w8sm48928988w

221 2.0.0 closing connection w8sm48928

cse312@ubuntu:~/Desktop/141044049$
```

(konu yok) - ebru.krds@gmail.com - Gmail - Mozilla Firefox

https://mail.google.com/mail/u/1/#inb

(konu yok) Gelen Kutusu ×

deneme1530@gmail.com 04:52 (0 dakika
Alıcı: