COMPUTER NETWORKS

Topic: Final Project

Due Date: 30-4-2023

Report BY:

Name	ID	LAB
Omar Waleed Mohamed Ashour	7058	G1 S2 L2
Shereen Mostafa Hassan Mabrouk	6844	G1 S2 L2
Hend Khaled Abdelhamid Mohamed Aly	6986	G1 S2 L2

Supervisor

DR. Kariem Banwan

Report Content:

Part One : Main Code
Part Two : Bonus code

Part One : Main Code

Code:

Server code

```python
# server
import socketserver
import binascii
import os
import signal
import threading

chunk_size = 1000
recseq = 0
recack = -1
initiate = 0
connection = 0




class MyUDPHandler(socketserver.BaseRequestHandler):
    initiate = 0
    sequence = 0
    recseq = 0
    chunk = 0

    def handle(self):
        global j, initiate, connection, sequence, err, mydomain, browser, formats, encodings, languages, contype, chunk, contenttype, recseq, timer
        socket = self.request[1]
        socket.settimeout(2)

        # data = self.request[0].strip()
        def finishconn(sequence, recseq):
            print("Timed out while waiting for input")
            send_data = "end"
            initiate = 0
            connection = 0
            checksum = binascii.crc32(send_data.encode())
            message = str(sequence).zfill(10) + str(recseq).zfill(10) + str(checksum).zfill(
                10) + "001" + send_data
            sequence = sequence + len(send_data)
```

```python
            sequence = sequence + len(send_data)
            socket.sendto(message.encode(), self.client_address)

        if initiate == 0:
            data = self.request[0].strip()
        else:
            def read_input():
                data = self.request[0].strip()
                return data

            timer = threading.Timer(4.0, lambda: finishconn(sequence, recseq))
            timer.start()
            try:
                data = read_input()
                timer.cancel()
            except:
                timer.cancel()
                pass
        data = data.decode()
        print(data)
        recseq = int(data[:10])
        data = data[10:]
        recack = int(data[:10])
        data = data[10:]
        reccheck = int(data[:10])
        data = data[10:]
        syn = int(data[0])
        ack = int(data[1])
        fin = int(data[2])
        data = data[3:]
        checking = binascii.crc32(data.encode())

        if fin:
            send_data = "end"
            initiate = 0
            connection = 0
            checksum = binascii.crc32(send_data.encode())
            message = str(sequence).zfill(10) + str(recseq + len(data)).zfill(10) + str(checksum).zfill(
```

server.py - FinalNetworksCode - Visual Studio Code

```python
            10) + "001" + send_data
        sequence = sequence + len(send_data)
        socket.sendto(message.encode(), self.client_address)
    if syn == 1 and ack == 0 and fin == 0:
        initiate = 1
        connection = 0
        j = -1
        err = 0
        sequence = 0
        initiate = 1
        send_data = "ok"
        checksum = binascii.crc32(send_data.encode())
        message = str(sequence).zfill(10) + str(recseq + len(data)).zfill(10) + str(checksum).zfill(
            10) + "110" + send_data
        sequence = sequence + len(send_data)
        socket.sendto(message.encode(), self.client_address)

    elif syn == 0 and ack == 1 and initiate == 1 and connection == 0 and fin == 0:
        connection = 1

    elif connection == 1 and data[:3].upper() == "GET" and fin == 0:
        datalines = data.split('\n')
        temp = datalines[0].split(" ")
        type, path, version = temp[0], temp[1], temp[2]
        datalines = datalines[1:]
        for line in datalines:
            header, attributes = line.split(" ", 1)
            if header == "Host:":
                mydomain = attributes
            elif header == "User-Agent:":
                browser = attributes
            elif header == "Accept:":
                formats = attributes.split(',')
            elif header == "Accept-Encoding:":
                encodings = attributes.split(',')
            elif header == "Accept-Language:":
                languages = attributes.split(',')
            elif header == "Connection:":
```

server.py - FinalNetworksCode - Visual Studio Code

```python
            elif header == "Connection:":
                contype = attributes

        current_dir = os.getcwd()
        if os.path.exists(current_dir + path):
            with open(current_dir + path, 'r') as file:
                file = file.read()
            chunk = [0 for i in range(int(len(file) / chunk_size) + 1)]
            j = 0
            for i in range(0, len(file), chunk_size):
                chunk[j] = file[i:i + chunk_size]
                j = j + 1
            j = -1

            send_data = "HTTP/1.1 200 OK\nContent-Type: text/html\nContent-Length:" + str(len(file)) + "\n"
            checksum = binascii.crc32(send_data.encode())
            message = str(sequence).zfill(10) + str(recseq + len(data)).zfill(10) + str(checksum).zfill(
                10) + "000" + send_data
            sequence = sequence + len(send_data)
            socket.sendto(message.encode(), self.client_address)

        else:
            send_data = "HTTP/1.1 404 Not_Found\nContent-Type: text/html; charset=utf-8\nContent-Length: 132\nConnection: close"
            checksum = binascii.crc32(send_data.encode())
            message = str(sequence).zfill(10) + str(recseq + len(data)).zfill(10) + str(checksum).zfill(
                10) + "001" + send_data
            sequence = sequence + len(send_data)
            socket.sendto(message.encode(), self.client_address)
            chunk = ""

    elif connection and data[:4].upper() == "POST" and fin == 0:
        splitter = ""
        datalines = data.split('\n')
        if (len(datalines) > 1):
            temp = datalines[0].split(" ")
            type, path, version = temp[0], temp[1], temp[2]
            datalines = datalines[1:3]
            for line in datalines:
```

```python
            datalines = datalines[1:3]
            for line in datalines:
                header, attributes = line.split(" ", 1)
                if header == "Host:":
                    mydomain = attributes
                elif header == "User-Agent:":
                    browser = attributes
                elif header == "Accept:":
                    formats = attributes.split(',')
                elif header == "Accept-Encoding:":
                    encodings = attributes.split(',')
                elif header == "Accept-Language:":
                    languages = attributes.split(',')
                elif header == "Connection:":
                    contype = attributes
                elif header == "Content-Type:":
                    contenttype = attributes.split(" boundary=")
                    splitter = contenttype[1]

            datalines = data.split('--' + splitter)
            datalines = datalines[1:-1]
            names = [0 for i in range(int(len(datalines)))]
            values = [0 for i in range(int(len(datalines)))]
            k = 0
            for line in datalines:
                temp = line.split('\n\n')
                values[k] = temp[-1].split('\n')[0]
                temp = temp[0].split("\"")
                names[k] = temp[1]
                k = k + 1
            print(names)
            print(values)
        send_data = "end"
        initiate = 0
        connection = 0
        checksum = binascii.crc32(send_data.encode())
        message = str(sequence).zfill(10) + str(recseq + len(data)).zfill(10) + str(checksum).zfill(
            10) + "001" + send_data
```

```python
            10) + "001" + send_data
        sequence = sequence + len(send_data)
        socket.sendto(message.encode(), self.client_address)


    elif connection and 'chunk' in globals():
        if (j + 1) < len(chunk) and fin == 0:
            if recack == sequence and checking == reccheck:
                j = j + 1
                err = 0
            else:
                sequence = sequence - len(chunk[j])
                err = err + 1
            send_data = chunk[j]
            checksum = binascii.crc32(send_data.encode())
            message = str(sequence).zfill(10) + str(recseq + len(data)).zfill(10) + str(checksum).zfill(
                10) + "000" + send_data
            sequence = sequence + len(send_data)
            socket.sendto(message.encode(), self.client_address)
        else:
            send_data = "end"
            initiate = 0
            connection = 0
            checksum = binascii.crc32(send_data.encode())
            message = str(sequence).zfill(10) + str(recseq + len(data)).zfill(10) + str(checksum).zfill(
                10) + "001" + send_data
            sequence = sequence + len(send_data)
            socket.sendto(message.encode(), self.client_address)
        if err > 100:
            send_data = "end"
            initiate = 0
            connection = 0
            checksum = binascii.crc32(send_data.encode())
            message = str(sequence).zfill(10) + str(recseq + len(data)).zfill(10) + str(checksum).zfill(
                10) + "001" + send_data
            sequence = sequence + len(send_data)
            socket.sendto(message.encode(), self.client_address)
    else:
```

server.py - FinalNetworksCode - Visual Studio Code

server.py > MyUDPHandler > handle

```python
                initiate = 0
                connection = 0
                checksum = binascii.crc32(send_data.encode())
                message = str(sequence).zfill(10) + str(recseq + len(data)).zfill(10) + str(checksum).zfill(
                    10) + "001" + send_data
                sequence = sequence + len(send_data)
                socket.sendto(message.encode(), self.client_address)
            else:
                send_data = "error, Invalid request!"
                initiate = 0
                connection = 0
                checksum = binascii.crc32(send_data.encode())
                message = str(sequence).zfill(10) + str(recseq + len(data)).zfill(10) + str(checksum).zfill(
                    10) + "001" + send_data
                sequence = sequence + len(send_data)
                socket.sendto(message.encode(), self.client_address)
            # timer = threading.Timer(4.0, lambda: finishconn(sequence, recseq))
            # timer.start()


if __name__ == "__main__":
    HOST, PORT = "localhost", 9999
    with socketserver.UDPServer((HOST, PORT), MyUDPHandler) as server:
        server.serve_forever()
```

Client code

client.py - FinalNetworksCode - Visual Studio Code

client.py > ...

```python
# client
import socket
import binascii
import select
import threading


def finishconn(sequence):
    print("Timed out while waiting for input")
    data = "end"
    checksum = binascii.crc32(data.encode())
    message = str(sequence).zfill(10) + str(last_ack).zfill(10) + str(checksum).zfill(10) + "001" + data
    sock.sendto(bytes(message, "utf-8"), (HOST, PORT))

first=1
rdata = ""
recseq = -1
j = 0
win = 5
last_ack = 0
chunk_size = 1000
window_start = 0
sequence = 0
HOST, PORT = "localhost", 9999
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.settimeout(2)

checksum = binascii.crc32("dataa".encode())
checksum = binascii.crc32("dataa".encode())

data = "begin"
checksum = binascii.crc32(data.encode())
message = str(sequence).zfill(10) + str(recseq + 1).zfill(10) + str(checksum).zfill(10) + "100" + data
sequence = sequence + len(data)
sock.sendto(bytes(message, "utf-8"), (HOST, PORT))
readable, writable, exceptional = select.select([sock], [], [], 5)
```

client.py - FinalNetworksCode - Visual Studio Code

```python

# if sock in readable:
#     received = str(sock.recv(chunk_size * 3), "utf-8")
# else:
#     print("No data received within 5 seconds")



def read_input():
    received = str(sock.recv(chunk_size * 3), "utf-8")
    return received


timer = threading.Timer(1.0,print(""))
timer.start()
try:
    received = read_input()
    timer.cancel()
except:
    timer.cancel()
    pass
if 'received' in locals():
    # received = str(sock.recv(chunk_size * 3), "utf-8")
    recseq = int(received[:10])
    received = received[10:]
    recack = int(received[:10])
    received = received[10:]
    reccheck = int(received[:10])
    received = received[10:]
    synflag = int(received[0])
    ackflag = int(received[1])
    finflag = int(received[2])
    received = received[3:]

    if synflag == 1 and ackflag == 1:
        data = "start"
        checksum = binascii.crc32(data.encode())
        message = str(sequence).zfill(10) + str(recseq + len(received)).zfill(10) + str(checksum).zfill(10) + "010" + data
```

client.py - FinalNetworksCode - Visual Studio Code

```python
    if synflag == 1 and ackflag == 1:
        data = "start"
        checksum = binascii.crc32(data.encode())
        message = str(sequence).zfill(10) + str(recseq + len(received)).zfill(10) + str(checksum).zfill(10) + "010" + data
        last_ack = recseq + len(received)
        sequence = sequence + len(data)
        sock.sendto(bytes(message, "utf-8"), (HOST, PORT))
        data = input("Enter your Get request: ")
        err = 0
        while True:
            line = input()
            if line:
                data += "\n" + line
                err = 0
            else:
                data += "\n"
                err = err + 1
                if err > 1:
                    err = 0
                    break
        print("done")

        # data = "GET /index.html HTTP/1.1\nHost: example.com\nUser-Agent: Mozilla/5.0\nAccept-Language: en-us\nAccept-Encoding: gzip, de
        checksum = binascii.crc32(data.encode())
        message = str(sequence).zfill(10) + str(recseq + len(received)).zfill(10) + str(checksum).zfill(
            10) + "000" + data
        last_ack = recseq + len(received)
        sequence = sequence + len(data)
        sock.sendto(bytes(message, "utf-8"), (HOST, PORT))
    while True:

        def read_input():
            received = str(sock.recv(chunk_size * 3), "utf-8")
            return received
```

client.py - FinalNetworksCode - Visual Studio Code

```python
            timer.cancel()
            pass
        recseq = int(received[:10])
        received = received[10:]
        recack = int(received[:10])
        received = received[10:]
        reccheck = int(received[:10])
        received = received[10:]
        synflag = int(received[0])
        ackflag = int(received[1])
        finflag = int(received[2])
        received = received[3:]
        checksum = binascii.crc32(received.encode())
        if first:
            first=0
            print(received.split('\n')[0].split(' ')[-2:])
        if last_ack == recseq and checksum == reccheck:
            data = "ack"
            checksum = binascii.crc32(data.encode())
            message = str(sequence).zfill(10) + str(recseq + len(received)).zfill(10) + str(checksum).zfill(
                10) + "010" + data
            last_ack = recseq + len(received)
            sequence = sequence + len(data)
            sock.sendto(bytes(message, "utf-8"), (HOST, PORT))
            if received != "end":
                rdata = rdata + received

        else:
            data = "ack"
            checksum = binascii.crc32(data.encode())
            message = str(sequence).zfill(10) + str(last_ack).zfill(10) + str(checksum).zfill(10) + "010" + data
            sock.sendto(bytes(message, "utf-8"), (HOST, PORT))
        if finflag:
            print(rdata)
            print("connection dropped")
            break
else:
    print("connection time out!")
```

Index.html

index.html - FinalNetworksCode - Visual Studio Code

```html
<!DOCTYPE html>
<html lang="en">
<head>
<title>Page Title</title>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<style>
body {
  font-family: Arial, Helvetica, sans-serif;
}
</style>
</head>
<body>

<h1>My Website</h1>
<p>A website created by me.</p>

</body>
</html>
```

ILLUSTRATION OF CODE:

Server code:

The code starts by importing the necessary libraries and defining some global variables, including chunk_size, recseq, recack, initiate, and connection. Then, the MyUDPHandler class is defined, which overrides the handle() method of the BaseRequestHandler class. The handle() method is called every time a new request is received by the server.

The handle() method starts by setting a timeout for the socket object, which is used to receive data from the client. Then, it reads the incoming data from the client and decodes it. The decoded data is then parsed to extract the sequence number, acknowledgment number, checksum, SYN flag, ACK flag, FIN flag, and data payload.

If the FIN flag is set, the server sends an "end" message to the client to indicate the end of the connection. If the SYN flag is set and the ACK flag is not set, the server sends an "ok" message to the client to establish a connection. If the SYN flag is not set and the ACK flag is set, the server sets the connection flag to 1 to indicate that a connection has been established.If the connection flag is set and the incoming data is a GET request, the server parses the request to extract the requested file path and other headers. If the file exists, the server reads the file and breaks it into chunks of size chunk_size. It then sends an HTTP response with the file content, content type, and content length to the client, along with the first chunk of data. If the file does not exist, the server sends a "404 Not Found" response to the client.

The server also includes a finishconn() function that is used to close the connection if no data is received from the client for a certain amount of time. The function sends an "end" message to the client and sets the initiate and connection flags to 0.

Finally, the server creates an instance of the MyUDPHandler class and binds it to a socket on a specific port. The server then enters a loop to listen for incoming requests and calls the handle() method for each request.

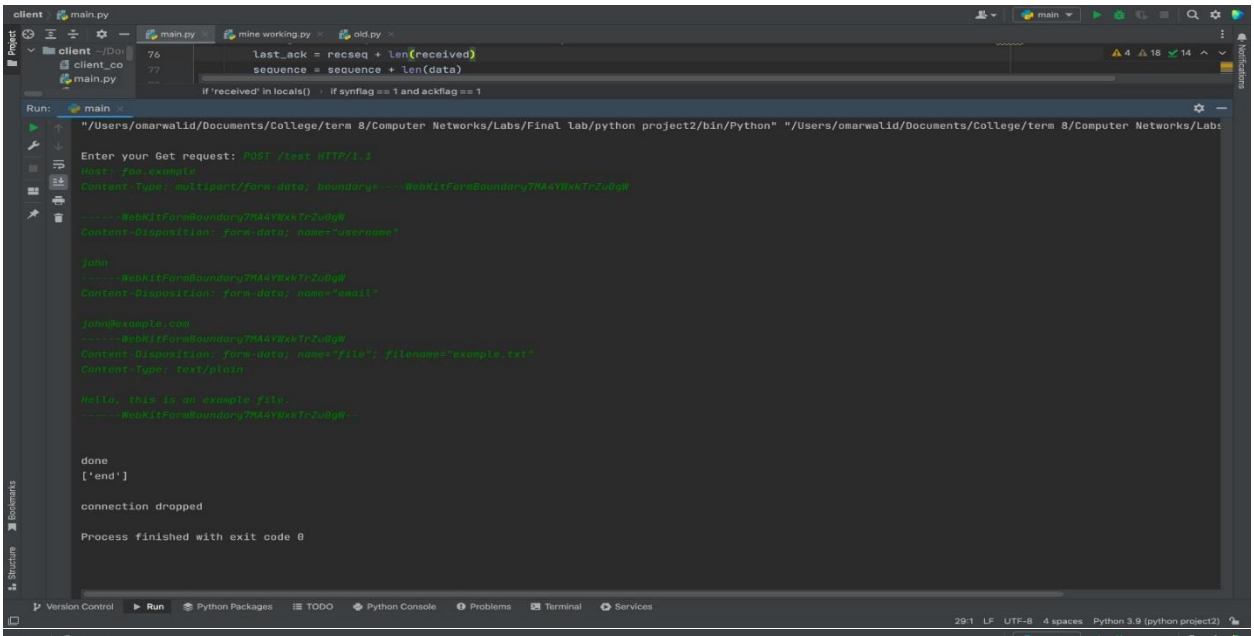
Client code:

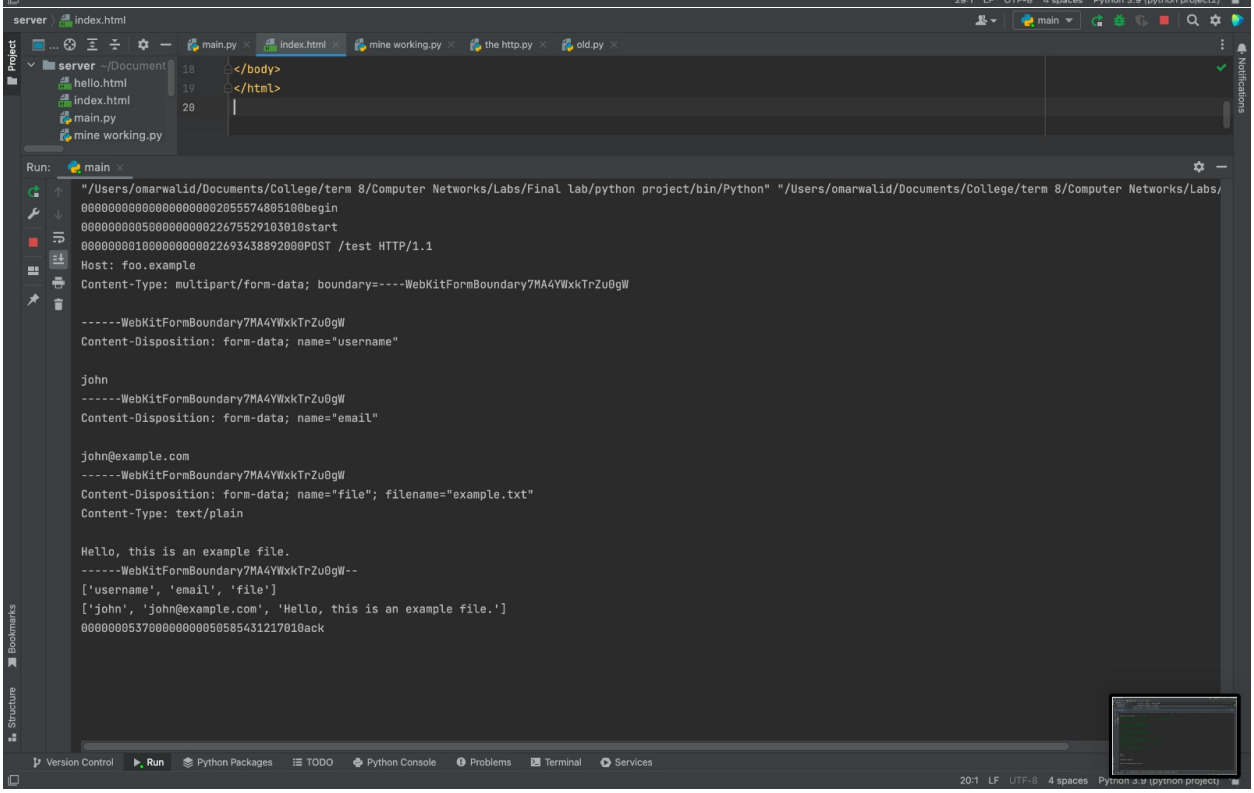
it sends an initial message to the server to establish a connection. The client then waits for a response from the server using a timer.Once a response is received, the client reads the input, extracts the relevant information, and sends an acknowledgment to the server. The client then enters a loop where it waits for further data from the server until it receives an "end" message.Inside the loop, the client reads the input from the server, extracts the relevant information, checks if the message is valid using a checksum, sends an acknowledgment to the server, and appends the received data to a buffer. If the message is not valid, it discards it.The client also has a function to finish the connection if it times out while waiting for input from the server.

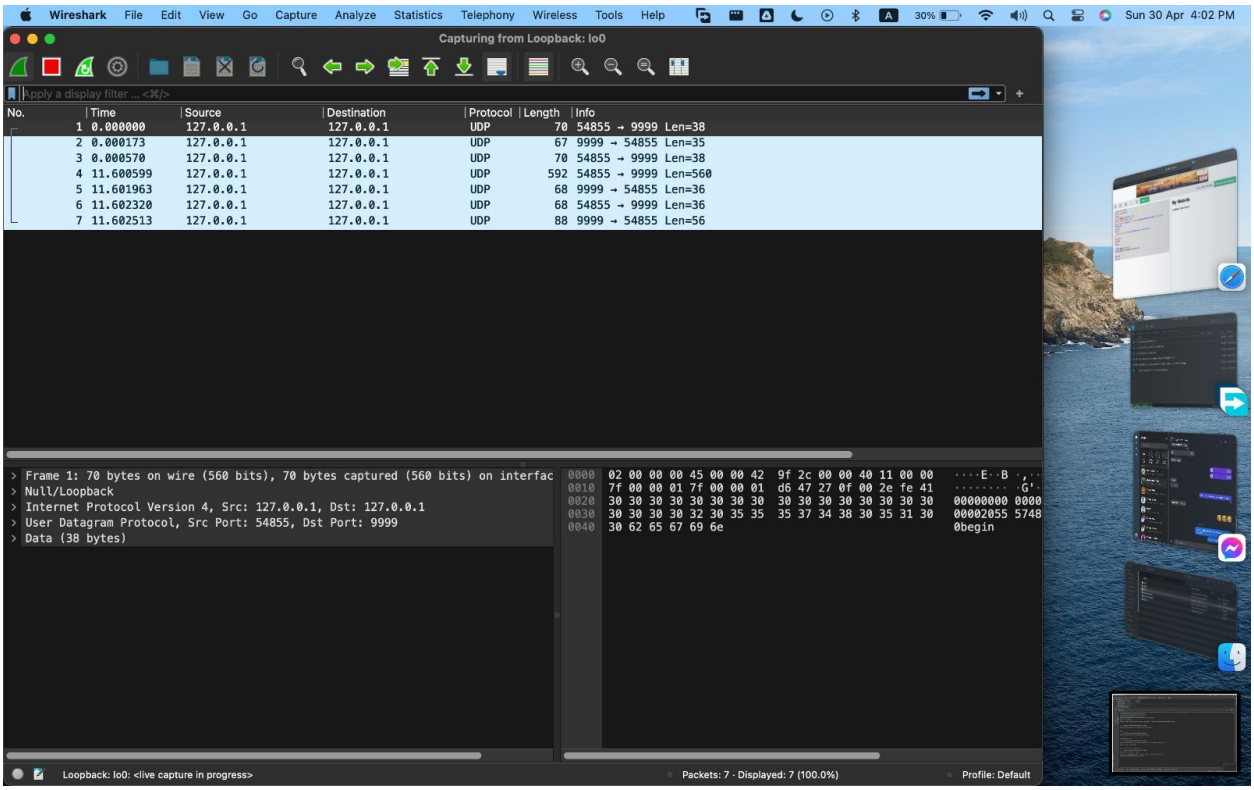
Test Cases wth wireshark:

Test case 1:

Post Request:







POST /test HTTP/1.1

Host: foo.example

Content-Type: multipart/form-data; boundary=----WebKitFormBoundary7MA4YWxkTrZu0gW

------WebKitFormBoundary7MA4YWxkTrZu0gW

Content-Disposition: form-data; name="username"

john

------WebKitFormBoundary7MA4YWxkTrZu0gW

Content-Disposition: form-data; name="email"

john@example.com

------WebKitFormBoundary7MA4YWxkTrZu0gW

Content-Disposition: form-data; name="file"; filename="example.txt"

Content-Type: text/plain

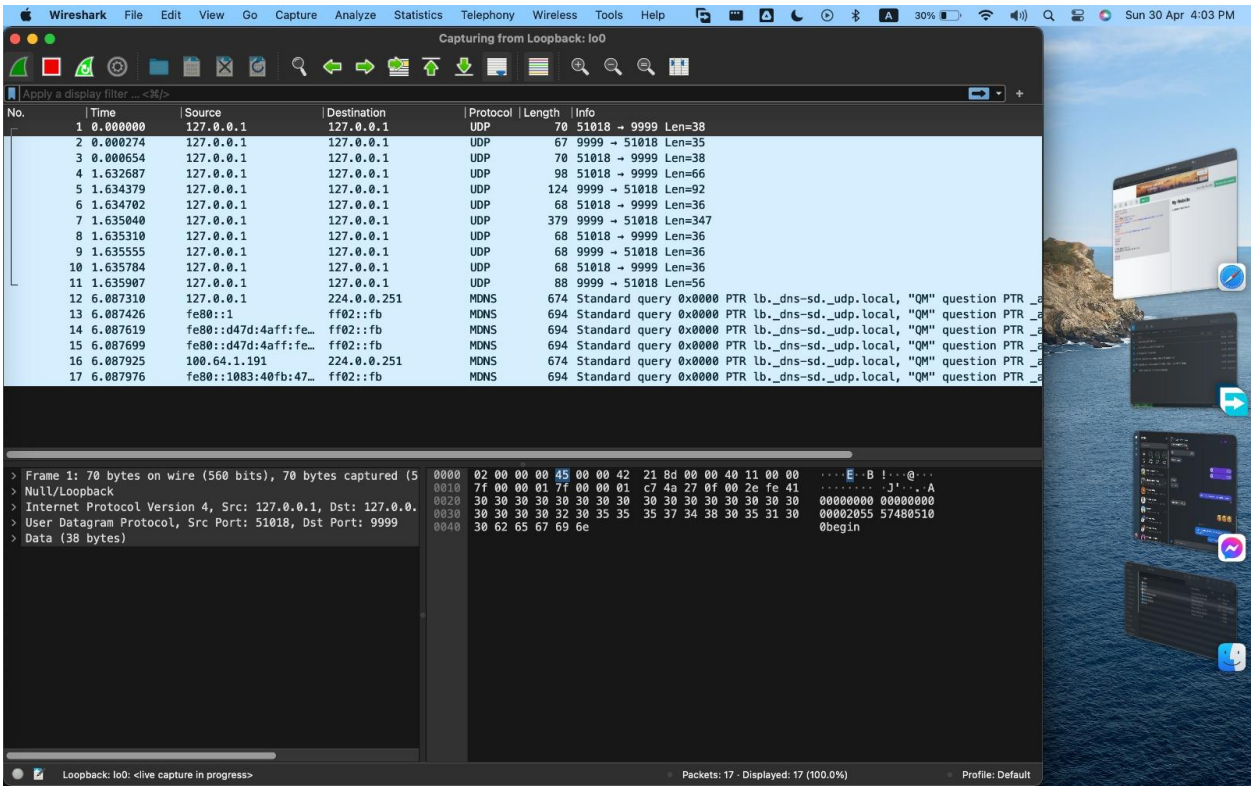
Hello, this is an example file.

------WebKitFormBoundary7MA4YWxkTrZu0gW--

Get Request 1

```
"/Users/omarwalid/Documents/College/term 8/Computer Networks/Labs/Final lab/python project2/bin/Python" "/Users/omarwalid/Documents/College/term 8/Computer Networks/Labs

Enter your Get request: GET /index.html 127.0.0.1 65432


done
['200', 'OK']
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length:314
<!DOCTYPE html>
<html lang="en">
<head>
<title>Page Title</title>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<style>
body {
  font-family: Arial, Helvetica, sans-serif;
}
</style>
</head>
<body>

<h1>My Website</h1>
<p>A website created by me.</p>

</body>
</html>

connection dropped

Process finished with exit code 0
```

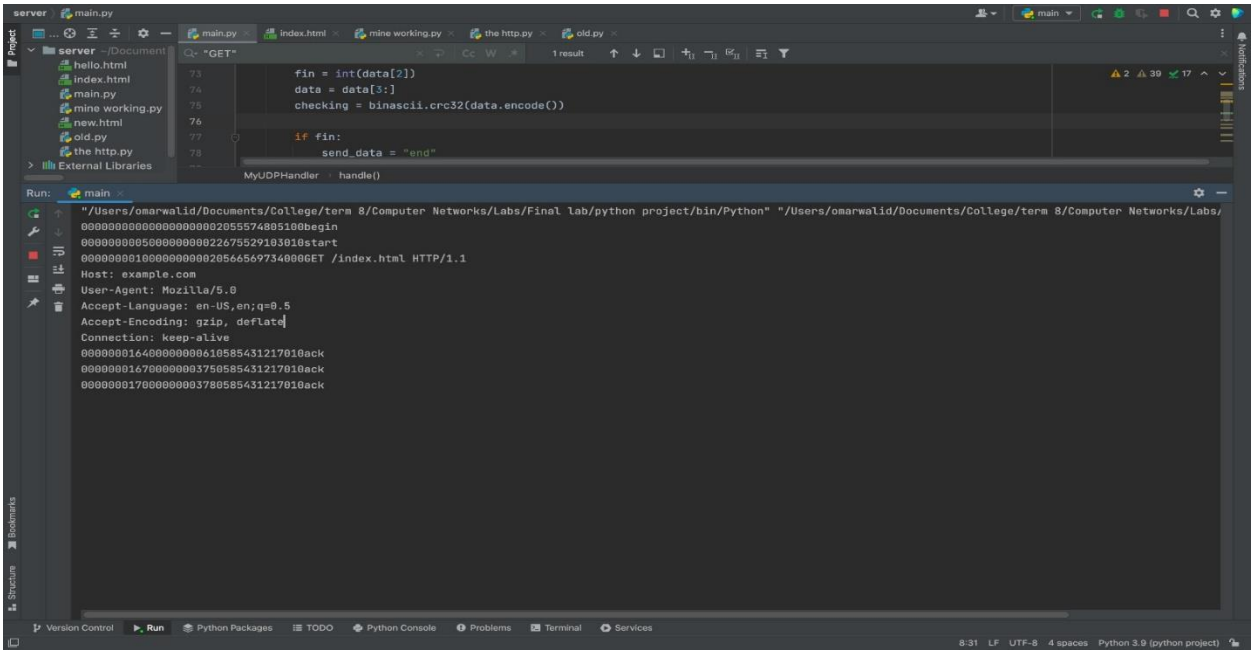
```
"/Users/omarwalid/Documents/College/term 8/Computer Networks/Labs/Final lab/python project/bin/Python" "/Users/omarwalid/Documents/College/term 8/Computer Networks/Labs/
0000000000000000000020555748051 00begin
00000000050000000002267552910301 0start
0000000010000000000023123616029000GET /index.html 127.0.0.1 65432
00000000430000000061058543121701 0ack
00000000460000000037505854312170 10ack
00000000490000000037805854312170 10ack
```

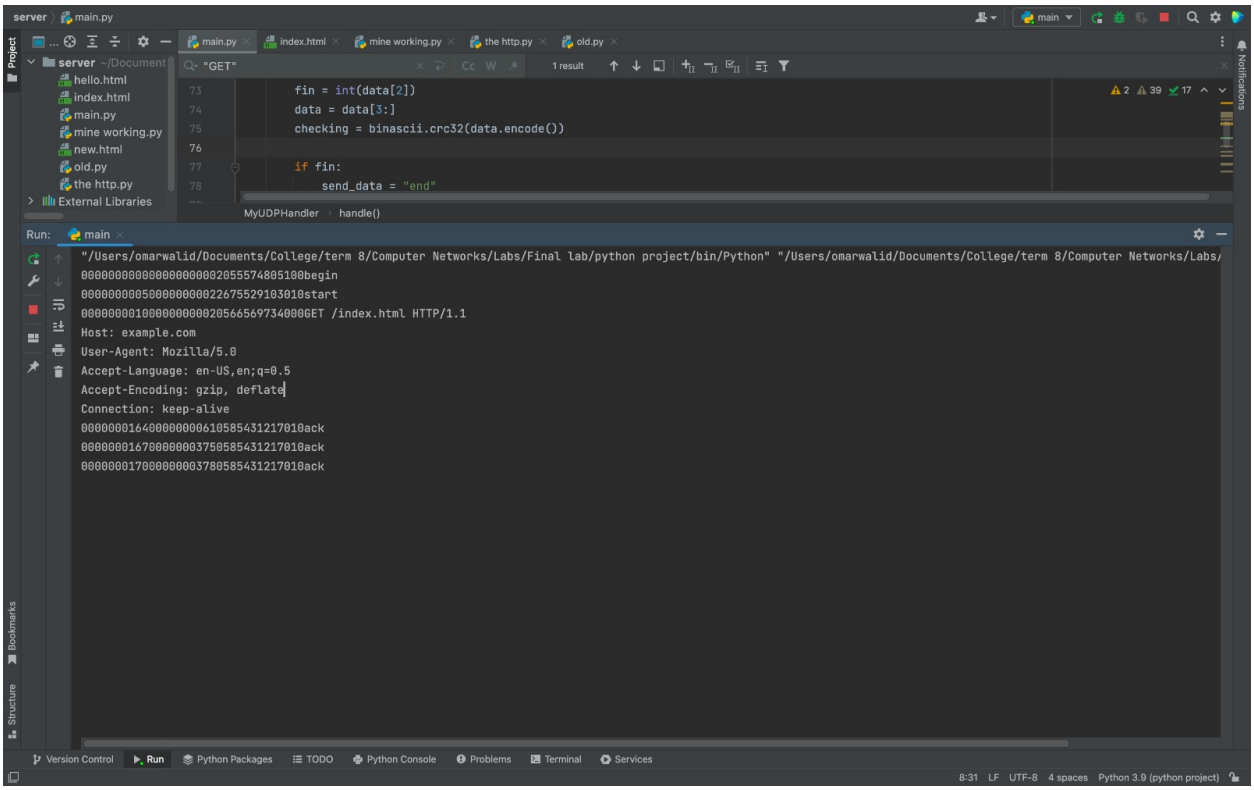


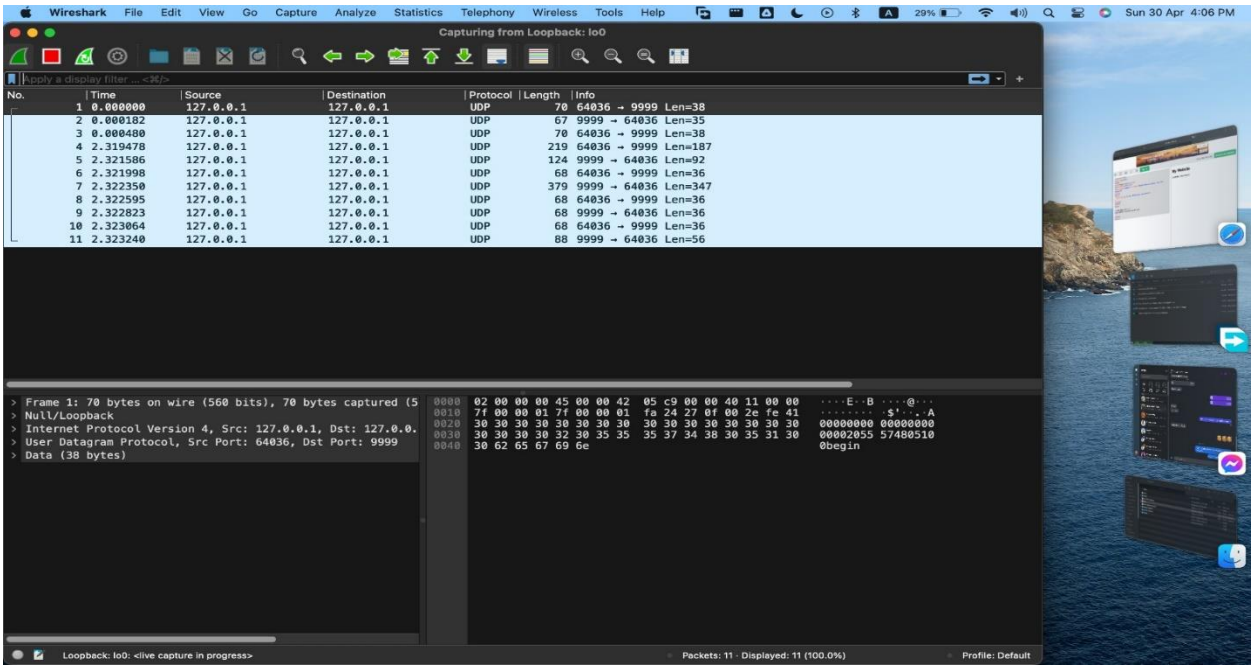
GET /index.html 127.0.0.1 65432

……

Get Request 2







GET /index.html HTTP/1.1

Host: example.com

User-Agent: Mozilla/5.0

Accept-Language: en-US,en;q=0.5

Accept-Encoding: gzip, deflate

Connection: keep-alive

Part Two : Bonus code

Code:

```python
                    ######################*BONUS CODE*##########################

import socket
from os.path import exists
from urllib.parse import unquote_plus
#request_parser function takes the incoming data and extracts the method, filename, HTTP version, and body
def request_parser(data):
    print("data received in bytes:")
    print(data)
    print("-----------------------------------------------------------------------------------------------")
    data = str(data, 'UTF-8')
    method = data.split(' /')[0]
    filename = data.split(' ')[1][1:]
    http_ver = data.split('\r\n')[0][-3:]
    body = ""
    if method == "POST":
        body = data.split("\r\n\r\n")[1]
    return method, filename, http_ver, body

#connection_parser function takes the incoming data and checks if the connection is going to be closed or kept alive.
def connection_parser(data):
    data = str(data, 'UTF-8')
    try:
        connection = data.split("Connection: ")[1][0]
        if connection == 'k':
            return 'k'
        else:
            return 'c'
    except IndexError:
        return 'c'

# keep_alive_parser function takes the incoming data and extracts the timeout value for a keep-alive connection.
def keep_alive_parser(data):
```

```python

# keep_alive_parser function takes the incoming data and extracts the timeout value for a keep-alive connection.
def keep_alive_parser(data):
    data = str(data, 'UTF-8')
    try:
        timeout = data.split("Keep-Alive: ")[1].split(", ")[0].split("=")[1]
        print("Time out: ", timeout)
        return int(timeout)
    except IndexError:
        return 10


def get_handler(filename, sock, http_ver):
    try:
        with open(filename, "r") as f:
            body = bytes(f.read(), 'utf-8')
    except FileNotFoundError:
        data_to_be_send = "HTTP/" + http_ver + " 404 Not Found\r\n\r\n"
        data_to_be_send = bytes(data_to_be_send, 'utf-8')
        sock.send(data_to_be_send)
        return

    data_to_be_send = "HTTP/" + http_ver + " 200 OK\r\n\r\n"
    data_to_be_send = bytes(data_to_be_send, 'utf-8')
    data_to_be_send = data_to_be_send + body
    print("Data to be sent in bytes:")
    print(data_to_be_send[:1024])
    print("----------------------------------------")
    sock.send(data_to_be_send)
    print("Data has been sent")
```

```python

def post_handler(filename, body, sock, http_ver):
    body = unquote_plus(body)


    print("body to be written in file:")
    print(body)
    print("----------------------------------------")
    f = open(filename, "w+")
    f.write(body)
    print("body has been wrote")
    f.close()
    ok_message = "HTTP/" + http_ver + " 200 OK\r\nContent-Type: text/html\r\n\r\n"
    ok_message = bytes(ok_message, 'utf-8')
    sock.send(ok_message)
    print("OK message has been sent")



def Main():
    host = "127.0.0.1"
    port = 65432
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.bind((host, port))
    print("------------------------------------------------------------------------------------------")
    print("socket binded to port", port)

    s.listen()
    print("socket is listening")
    print("------------------------------------------------------------------------------------------")

```

```python
    while True:
        c, addr = s.accept()
        print('Connected to :', addr[0], ':', addr[1])

        while True:
            data = b''
            try:
                recv_data = []
                recv_data.append(c.recv(1024))
                c.settimeout(0.5)
                i = 0
                while recv_data[i]:
                    data += recv_data[i]
                    i += 1
                    recv_data.append(c.recv(1024))
                c.settimeout(None)
            except TimeoutError:
                c.settimeout(None)

            if not data:
                break

            method, filename, http_ver, body = request_parser(data)
            print("method: " + method)
            print("filename: " + filename)
            print("http_ver: " + http_ver)
            print("------------------------------------------------------------------------------------------")
```

```python
            print("------------------------------------------------------------------------------------------")

            if method == "POST":
                print("in POST")
                post_handler(filename, body, c, http_ver)
                print(
                    "------------------------------------------------------------------------------------------")

            elif method == "GET":
                print("in GET")
                is_exist = exists(filename)
                if is_exist:
                    print("file exists")
                    print("----------------------------------------")
                    get_handler(filename, c, http_ver)
                    print(
                        "------------------------------------------------------------------------------------------")
                else:
                    print("file not exists")
                    error_message = "HTTP/" + http_ver + " 404 Not Found\r\n\r\n"
                    error_message = bytes(error_message, 'utf-8')
                    c.send(error_message)
                    print("Error message has been sent")
                    print(
                        "------------------------------------------------------------------------------------------")

            if http_ver == "1.1":
                connection = connection_parser(data)
                if connection == 'k':
                    timeout = keep_alive_parser(data)
                    print("Waiting... 1.1, keep-alive")
                    c.settimeout(timeout)
                else:
                    print("Not waiting... 1.1, close")
                    break
            else :
                print("Not waiting... 1.0")
                break
```

ILLUSTRATION OF CODE:

We import tcp to use it to make the connection of HTTP with the web browser

We create five functions:

1. Request parser to extract the method and filename and http version

2. conncetion parser takes the incoming data and checks if the connection is going to be closed or kept alive

3.keep alive parser takes the incoming data and extracts the timeout value for a keep-alive connection.

4.get handler to handle the get request coming

5. post handler to handle post request coming

Then the main function

The main function first of all we intiallized the hoost and port then

the server creates a new socket object to handle the communication with the client. The accept() method is used to accept incoming client connections and return a new socket object for the server to communicate with that specific client. The new socket object c is used to send and receive data with the client, while the original socket s continues listening for incoming connections.

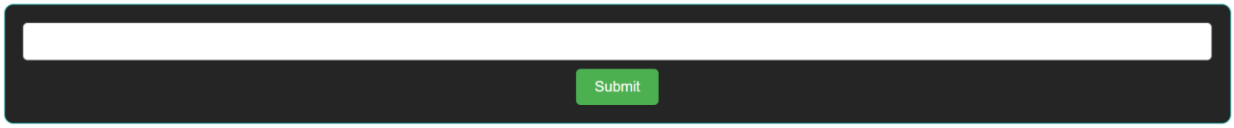
Then we call request parser to extract the method and file name and http version then we create if loop to takke action either the method is GET or POST

then we create if loop to see the http version

We create html file and put it in the same directory of server also we create a form inside it to submit the data and we put css to make it in good shape

```html
<!DOCTYPE html>
<html>
  <head>
    <title>Test Form</title>
    <style>
      form {
        display: flex;
        flex-direction: column;
        align-items: center;
        justify-content: center;
        margin: 20px;
        padding: 20px;
        border: 1px solid #01b1a8;
        border-radius: 10px;
        background-color: #252525;
      }

      input[type="text"] {
        width: 100%;
        padding: 10px;
        margin-bottom: 10px;
        box-sizing: border-box;
        border: 1px solid #ccc;
        border-radius: 5px;
        font-size: 16px;
      }

      input[type="submit"] {
        background-color: #4CAF50;
        color: white;
        padding: 10px 20px;
        border: none;
        border-radius: 5px;
        cursor: pointer;
        font-size: 16px;
      }
```

```html
        font-size: 16px;
      }

      input[type="submit"] {
        background-color: #4CAF50;
        color: white;
        padding: 10px 20px;
        border: none;
        border-radius: 5px;
        cursor: pointer;
        font-size: 16px;
      }

      input[type="submit"]:hover {
        background-color: #3e8e41;
      }
    </style>
  </head>
  <body>
    <form method="POST" action="http://localhost:65432/test.txt">
      <input type="text" name="Write any message">
      <input type="submit" value="Submit">
    </form>
  </body>
</html>
```

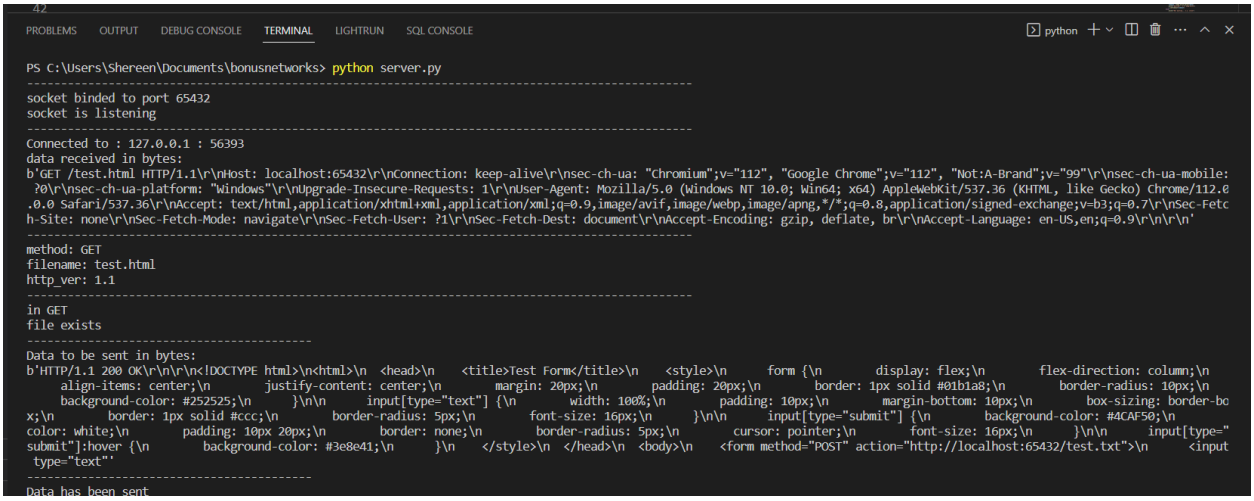


How the code works:

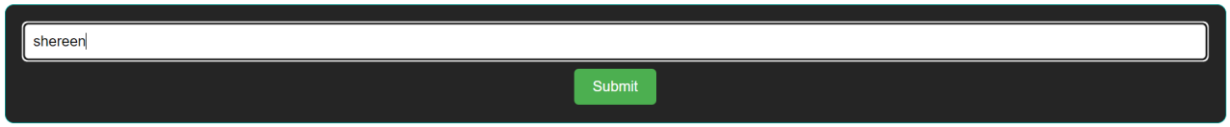
First of all we run the server then we open any browser and write the link of local host then the html page link .. if it found the page link it return get ok if no it return 404 not found
then if the page link is correct we entered any message and click submit so it returned it on text file and the send message is post ok 200

Lets see test cases

Test Cases:



http://localhost:65432/test.html



```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   LIGHTRUN   SQL CONSOLE                                   python

Waiting... 1.1, keep-alive
Connected to : 127.0.0.1 : 56563
data received in bytes:
b'POST /test.txt HTTP/1.1\r\nHost: localhost:65432\r\nConnection: keep-alive\r\nContent-Length: 25\r\nCache-Control: max-age=0\r\nsec-ch-ua: "Chromium";v="112", "Google Chrome";
v="112", "Not:A-Brand";v="99"\r\nsec-ch-ua-mobile: ?0\r\nsec-ch-ua-platform: "Windows"\r\nUpgrade-Insecure-Requests: 1\r\nOrigin: http://localhost:65432\r\nContent-Type: applica
tion/x-www-form-urlencoded\r\nUser-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/112.0.0.0 Safari/537.36\r\nAccept: text/html,ap
plication/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7\r\nSec-Fetch-Site: same-origin\r\nSec-Fetch-Mode: nav
igate\r\nSec-Fetch-User: ?1\r\nSec-Fetch-Dest: document\r\nReferer: http://localhost:65432/test.html\r\nAccept-Encoding: gzip, deflate, br\r\nAccept-Language: en-US,en;q=0.9\r\n
\r\nWrite+any+message=shereen'
--------------------------------------------------------------------------------
method: POST
filename: test.txt
http_ver: 1.1
--------------------------------------------------------------------------------
in POST
body to be written in file:
Write any message=shereen
-----------------------------------------
body has been wrote
--------------------------------------------------------------------------------
Waiting... 1.1, keep-alive
Connected to : 127.0.0.1 : 56564
```

as shown in the previous test case example the data is Get and post successfully

```
EXPLORER                server.py    test.txt
BONUS...                test.txt
  .dist                   1   Write any message=shereen
  server.py
  test.html
  test.txt
```

and it created test.txt text file and wrote the message sent in it

Lets see if I put a nonexisting link

For example : http://localhost:65432/shereen.html

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   LIGHTRUN   SQL CONSOLE                                   python

Waiting... 1.1, keep-alive
Connected to : 127.0.0.1 : 56855
Connected to : 127.0.0.1 : 56856
Connected to : 127.0.0.1 : 56934
data received in bytes:
b'GET /shereen.html HTTP/1.1\r\nHost: localhost:65432\r\nConnection: keep-alive\r\nsec-ch-ua: "Chromium";v="112", "Google Chrome";v="112", "Not:A-Brand";v="99"\r\nsec-ch-ua-mobi
le: ?0\r\nsec-ch-ua-platform: "Windows"\r\nUpgrade-Insecure-Requests: 1\r\nUser-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/11
2.0.0.0 Safari/537.36\r\nAccept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7\r\nSec-F
etch-Site: none\r\nSec-Fetch-Mode: navigate\r\nSec-Fetch-User: ?1\r\nSec-Fetch-Dest: document\r\nAccept-Encoding: gzip, deflate, br\r\nAccept-Language: en-US,en;q=0.9\r\n\r\n'
--------------------------------------------------------------------------------
method: GET
filename: shereen.html
http_ver: 1.1
--------------------------------------------------------------------------------
in GET
file not exists
Error message has been sent
--------------------------------------------------------------------------------
Waiting... 1.1, keep-alive
Connected to : 127.0.0.1 : 56935
```

It shown that this file not exists

Wireshark Output:

Adapter for loopback traffic capture

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

tcp.port==65432

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	::1	::1	TCP	76	56559 → 65432 [SYN] Seq=0 Win=65535 Len=0 MSS=65475 WS=256 SACK_PERM
2	0.000014	::1	::1	TCP	64	65432 → 56559 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
3	0.001386	::1	::1	TCP	76	56560 → 65432 [SYN] Seq=0 Win=65535 Len=0 MSS=65475 WS=256 SACK_PERM
4	0.001402	::1	::1	TCP	64	65432 → 56560 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
5	0.262383	::1	::1	TCP	76	56561 → 65432 [SYN] Seq=0 Win=65535 Len=0 MSS=65475 WS=256 SACK_PERM
6	0.262427	::1	::1	TCP	64	65432 → 56561 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
7	0.308725	127.0.0.1	127.0.0.1	TCP	56	56562 → 65432 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
8	0.308794	127.0.0.1	127.0.0.1	TCP	56	65432 → 56562 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
9	0.308900	127.0.0.1	127.0.0.1	TCP	44	56562 → 65432 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
10	0.309033	127.0.0.1	127.0.0.1	TCP	56	56563 → 65432 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
11	0.309093	127.0.0.1	127.0.0.1	TCP	56	65432 → 56563 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
12	0.309124	127.0.0.1	127.0.0.1	TCP	44	56563 → 65432 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
13	0.309391	127.0.0.1	127.0.0.1	TCP	44	65432 → 56398 [FIN, ACK] Seq=1 Ack=1 Win=10233 Len=0
14	0.309422	127.0.0.1	127.0.0.1	TCP	44	56398 → 65432 [ACK] Seq=1 Ack=2 Win=10233 Len=0
15	0.309461	127.0.0.1	127.0.0.1	HTTP	712	GET /test.html HTTP/1.1
16	0.309481	127.0.0.1	127.0.0.1	TCP	44	65432 → 56562 [ACK] Seq=1 Ack=669 Win=2619648 Len=0
17	0.572875	127.0.0.1	127.0.0.1	TCP	56	56564 → 65432 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
18	0.572930	127.0.0.1	127.0.0.1	TCP	56	65432 → 56564 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
19	0.572992	127.0.0.1	127.0.0.1	TCP	44	56564 → 65432 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
20	0.828328	127.0.0.1	127.0.0.1	TCP	1168	65432 → 56562 [PSH, ACK] Seq=1 Ack=669 Win=2619648 Len=1124 [TCP segment of a reassembled PDU]
21	0.828364	127.0.0.1	127.0.0.1	TCP	44	56562 → 65432 [ACK] Seq=669 Ack=1125 Win=2618624 Len=0
22	10.839948	127.0.0.1	127.0.0.1	HTTP	44	HTTP/1.1 200 OK
23	10.839993	127.0.0.1	127.0.0.1	TCP	44	56562 → 65432 [ACK] Seq=669 Ack=1126 Win=2618624 Len=0
24	10.840217	127.0.0.1	127.0.0.1	TCP	44	56562 → 65432 [FIN, ACK] Seq=669 Ack=1126 Win=2618624 Len=0
25	10.840270	127.0.0.1	127.0.0.1	TCP	44	65432 → 56562 [ACK] Seq=1126 Ack=670 Win=2619648 Len=0
26	28.636042	127.0.0.1	127.0.0.1	HTTP	914	POST /test.txt HTTP/1.1 (application/x-www-form-urlencoded)
27	28.636071	127.0.0.1	127.0.0.1	TCP	44	65432 → 56563 [ACK] Seq=1 Ack=871 Win=2619648 Len=0
28	29.152577	127.0.0.1	127.0.0.1	TCP	88	65432 → 56563 [PSH, ACK] Seq=1 Ack=871 Win=2619648 Len=44 [TCP segment of a reassembled PDU]
29	29.152698	127.0.0.1	127.0.0.1	TCP	44	56563 → 65432 [ACK] Seq=871 Ack=45 Win=2619648 Len=0
30	39.156865	127.0.0.1	127.0.0.1	HTTP	44	HTTP/1.1 200 OK
31	39.156925	127.0.0.1	127.0.0.1	TCP	44	56563 → 65432 [ACK] Seq=871 Ack=46 Win=2619648 Len=0
32	39.157243	127.0.0.1	127.0.0.1	TCP	44	56563 → 65432 [FIN, ACK] Seq=871 Ack=46 Win=2619648 Len=0
33	39.157292	127.0.0.1	127.0.0.1	TCP	44	65432 → 56563 [ACK] Seq=46 Ack=872 Win=2619648 Len=0

Adapter for loopback traffic capture

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

tcp.port==65432

No.	Time	Source	Destination	Protocol	Length	Info
1844	5077.150235	::1	::1	TCP	64	65432 → 56933 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
1845	5077.196242	127.0.0.1	127.0.0.1	TCP	56	56934 → 65432 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
1846	5077.196410	127.0.0.1	127.0.0.1	TCP	56	65432 → 56934 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
1847	5077.196578	127.0.0.1	127.0.0.1	TCP	44	56934 → 65432 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
1848	5077.196874	127.0.0.1	127.0.0.1	TCP	56	56935 → 65432 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
1849	5077.196986	127.0.0.1	127.0.0.1	TCP	56	65432 → 56935 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
1850	5077.197110	127.0.0.1	127.0.0.1	TCP	44	56935 → 65432 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
1851	5077.197121	127.0.0.1	127.0.0.1	TCP	44	65432 → 56856 [FIN, ACK] Seq=1 Ack=2 Win=2619648 Len=0
1852	5077.197210	127.0.0.1	127.0.0.1	TCP	44	56856 → 65432 [ACK] Seq=2 Ack=2 Win=2619648 Len=0
1853	5077.197848	127.0.0.1	127.0.0.1	HTTP	715	GET /shereen.html HTTP/1.1
1854	5077.197906	127.0.0.1	127.0.0.1	TCP	44	65432 → 56934 [ACK] Seq=1 Ack=672 Win=2619648 Len=0
1855	5077.460214	127.0.0.1	127.0.0.1	TCP	56	56936 → 65432 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
1856	5077.460358	127.0.0.1	127.0.0.1	TCP	56	65432 → 56936 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
1857	5077.460495	127.0.0.1	127.0.0.1	TCP	44	56936 → 65432 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
1858	5077.715861	127.0.0.1	127.0.0.1	TCP	70	65432 → 56934 [PSH, ACK] Seq=1 Ack=672 Win=2619648 Len=26 [TCP segment of a reassembled PDU]
1859	5077.715919	127.0.0.1	127.0.0.1	TCP	44	56934 → 65432 [ACK] Seq=672 Ack=27 Win=2619648 Len=0
1860	5087.731804	127.0.0.1	127.0.0.1	HTTP	44	HTTP/1.1 404 Not Found
1861	5087.731852	127.0.0.1	127.0.0.1	TCP	44	56934 → 65432 [ACK] Seq=672 Ack=28 Win=2619648 Len=0
1862	5087.732129	127.0.0.1	127.0.0.1	TCP	44	56934 → 65432 [FIN, ACK] Seq=672 Ack=28 Win=2619648 Len=0
1863	5087.732189	127.0.0.1	127.0.0.1	TCP	44	65432 → 56934 [ACK] Seq=28 Ack=673 Win=2619648 Len=0
1876	5122.204298	127.0.0.1	127.0.0.1	TCP	45	[TCP Keep-Alive] 56935 → 65432 [ACK] Seq=0 Ack=1 Win=2619648 Len=1
1877	5122.204330	127.0.0.1	127.0.0.1	TCP	56	[TCP Window Update] 65432 → 56935 [ACK] Seq=1 Ack=1 Win=2619648 Len=0 SLE=0 SRE=1
1878	5122.471309	127.0.0.1	127.0.0.1	TCP	45	[TCP Keep-Alive] 56936 → 65432 [ACK] Seq=0 Ack=1 Win=2619648 Len=1
1879	5122.471325	127.0.0.1	127.0.0.1	TCP	56	[TCP Window Update] 65432 → 56936 [ACK] Seq=1 Ack=1 Win=2619648 Len=0 SLE=0 SRE=1
1884	5167.219318	127.0.0.1	127.0.0.1	TCP	45	[TCP Keep-Alive] 56935 → 65432 [ACK] Seq=0 Ack=1 Win=2619648 Len=1