



ALEXANDRIA
UNIVERSITY

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:

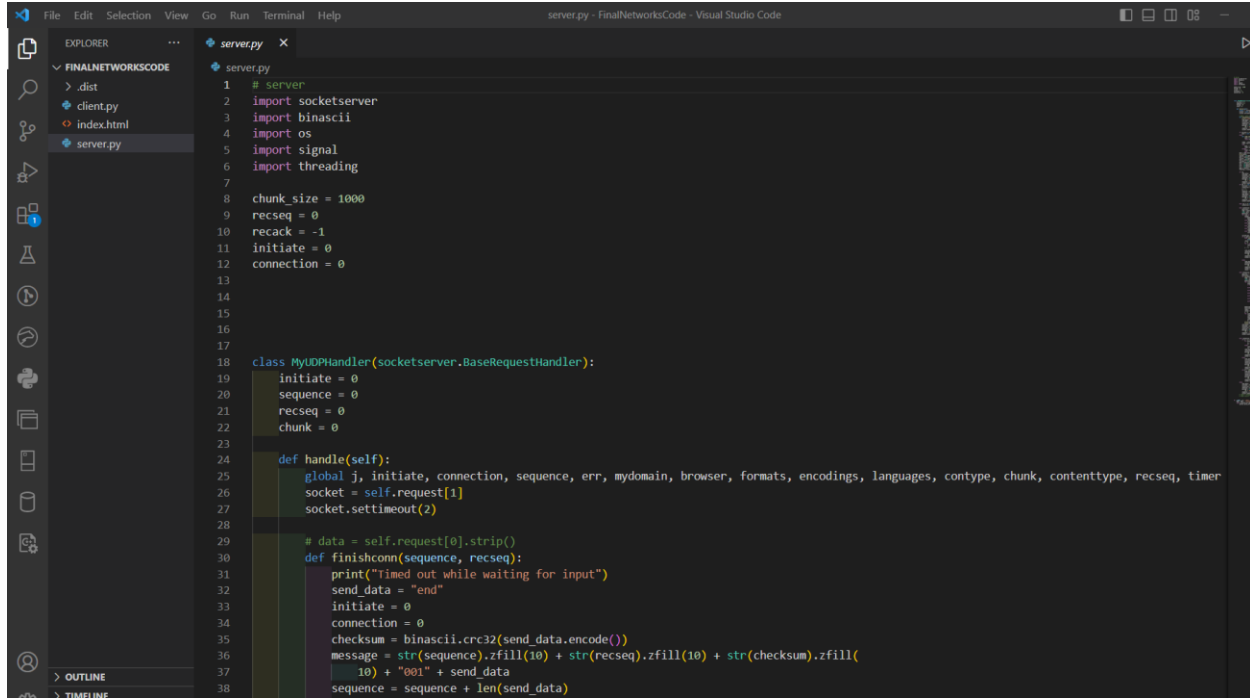
Part One : Main Code

Part Two : Bonus code

Part One : Main Code

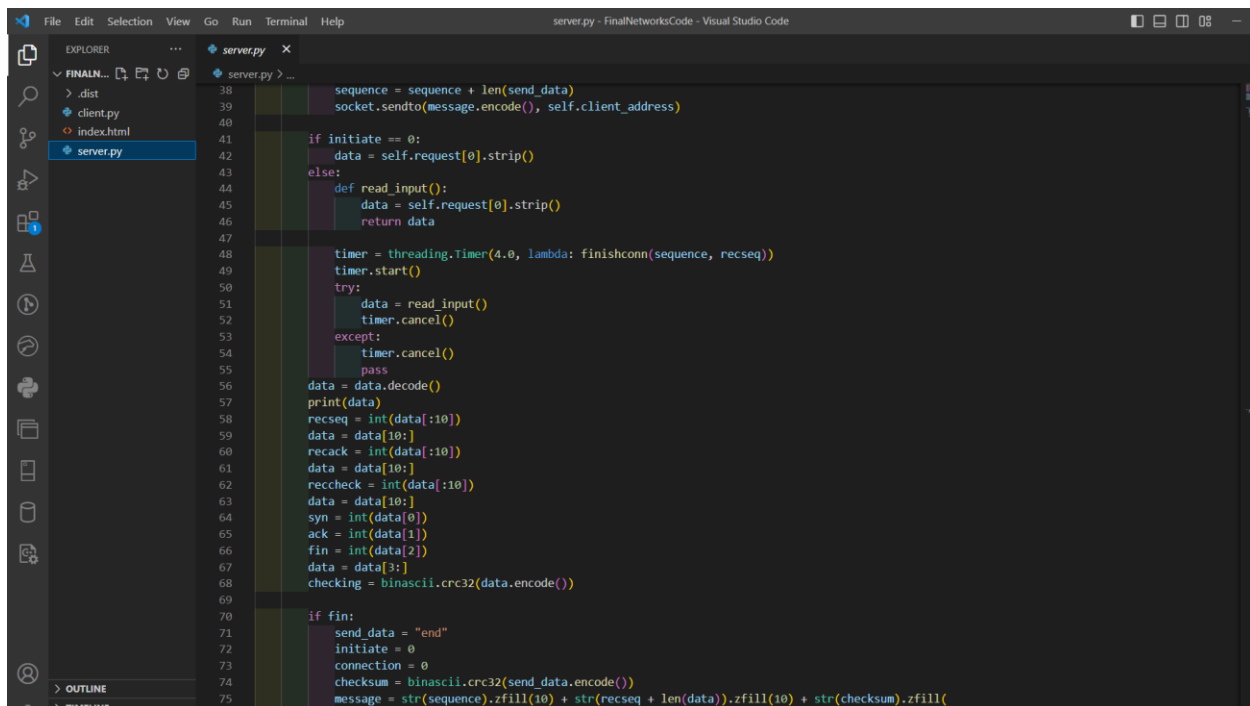
Code:

Server code



This screenshot shows the first 38 lines of the server.py file in Visual Studio Code. The Explorer sidebar on the left shows a project named 'FINALNETWORKCODE' with files: .dist, client.py, index.html, and server.py. The main editor displays the following code:

```
1 # server
2 import socketserver
3 import binascii
4 import os
5 import signal
6 import threading
7
8 chunk_size = 1000
9 recseq = 0
10 recack = -1
11 initiate = 0
12 connection = 0
13
14
15
16
17
18 class MyUDPHandler(socketserver.BaseRequestHandler):
19     initiate = 0
20     sequence = 0
21     recseq = 0
22     chunk = 0
23
24     def handle(self):
25         global j, initiate, connection, sequence, err, mydomain, browser, formats, encodings, languages, contype, chunk, contenttype, recseq, timer
26         socket = self.request[1]
27         socket.settimeout(2)
28
29         # data = self.request[0].strip()
30         def finishconn(sequence, recseq):
31             print("Timed out while waiting for input")
32             send_data = "end"
33             initiate = 0
34             connection = 0
35             checksum = binascii.crc32(send_data.encode())
36             message = str(sequence).zfill(10) + str(recseq).zfill(10) + str(checksum).zfill(
37                 10) + "001" + send_data
38             sequence = sequence + len(send_data)
```



This screenshot shows the continuation of the server.py file in Visual Studio Code, starting from line 38. The Explorer sidebar is the same as in the previous screenshot. The main editor displays the following code:

```
38         sequence = sequence + len(send_data)
39         socket.sendto(message.encode(), self.client_address)
40
41         if initiate == 0:
42             data = self.request[0].strip()
43         else:
44             def read_input():
45                 data = self.request[0].strip()
46                 return data
47
48         timer = threading.Timer(4.0, lambda: finishconn(sequence, recseq))
49         timer.start()
50         try:
51             data = read_input()
52             timer.cancel()
53         except:
54             timer.cancel()
55             pass
56         data = data.decode()
57         print(data)
58         recseq = int(data[:10])
59         data = data[10:]
60         recack = int(data[:10])
61         data = data[10:]
62         reccheck = int(data[:10])
63         data = data[10:]
64         syn = int(data[0])
65         ack = int(data[1])
66         fin = int(data[2])
67         data = data[3:]
68         checking = binascii.crc32(data.encode())
69
70         if fin:
71             send_data = "end"
72             initiate = 0
73             connection = 0
74             checksum = binascii.crc32(send_data.encode())
75             message = str(sequence).zfill(10) + str(recseq + len(data)).zfill(10) + str(checksum).zfill(
```

```

server.py - FinalNetworksCode - Visual Studio Code
EXPLORER
  FINAL...
    .dist
    client.py
    index.html
    server.py
  server.py
76         10) + "001" + send_data
77         sequence = sequence + len(send_data)
78         socket.sendto(message.encode(), self.client_address)
79     if syn == 1 and ack == 0 and fin == 0:
80         initiate = 1
81         connection = 0
82         j = -1
83         err = 0
84         sequence = 0
85         initiate = 1
86         send_data = "ok"
87         checksum = binascii.crc32(send_data.encode())
88         message = str(sequence).zfill(10) + str(recseq + len(data)).zfill(10) + str(checksum).zfill(
89             10) + "110" + send_data
90         sequence = sequence + len(send_data)
91         socket.sendto(message.encode(), self.client_address)
92
93     elif syn == 0 and ack == 1 and initiate == 1 and connection == 0 and fin == 0:
94         connection = 1
95
96     elif connection == 1 and data[:3].upper() == "GET" and fin == 0:
97         datalines = data.split('\n')
98         temp = datalines[0].split(" ")
99         type, path, version = temp[0], temp[1], temp[2]
100        datalines = datalines[1:]
101        for line in datalines:
102            header, attributes = line.split(" ", 1)
103            if header == "Host:":
104                mydomain = attributes
105            elif header == "User-Agent:":
106                browser = attributes
107            elif header == "Accept:":
108                formats = attributes.split(',')
109            elif header == "Accept-Encoding:":
110                encodings = attributes.split(',')
111            elif header == "Accept-Language:":
112                languages = attributes.split(',')
113            elif header == "Connection:":
114                continue = attributes

```

```

server.py - FinalNetworksCode - Visual Studio Code
EXPLORER
  FINAL...
    .dist
    client.py
    index.html
    server.py
  server.py
113        elif header == "Connection:":
114            continue = attributes
115
116        current_dir = os.getcwd()
117        if os.path.exists(current_dir + path):
118            with open(current_dir + path, 'r') as file:
119                file = file.read()
120                chunk = [0 for i in range(int(len(file) / chunk_size) + 1)]
121                j = 0
122                for i in range(0, len(file), chunk_size):
123                    chunk[j] = file[i:i + chunk_size]
124                    j = j + 1
125                j = -1
126
127                send_data = "HTTP/1.1 200 OK\nContent-Type: text/html\nContent-Length:" + str(len(file)) + "\n"
128                checksum = binascii.crc32(send_data.encode())
129                message = str(sequence).zfill(10) + str(recseq + len(data)).zfill(10) + str(checksum).zfill(
130                    10) + "000" + send_data
131                sequence = sequence + len(send_data)
132                socket.sendto(message.encode(), self.client_address)
133
134            else:
135                send_data = "HTTP/1.1 404 Not Found\nContent-Type: text/html; charset=utf-8\nContent-Length: 132\nConnection: close"
136                checksum = binascii.crc32(send_data.encode())
137                message = str(sequence).zfill(10) + str(recseq + len(data)).zfill(10) + str(checksum).zfill(
138                    10) + "001" + send_data
139                sequence = sequence + len(send_data)
140                socket.sendto(message.encode(), self.client_address)
141                chunk = ""
142
143        elif connection and data[:4].upper() == "POST" and fin == 0:
144            splitter = ""
145            datalines = data.split('\n')
146            if (len(datalines) > 1):
147                temp = datalines[0].split(" ")
148                type, path, version = temp[0], temp[1], temp[2]
149                datalines = datalines[1:]
150                for line in datalines:

```

```

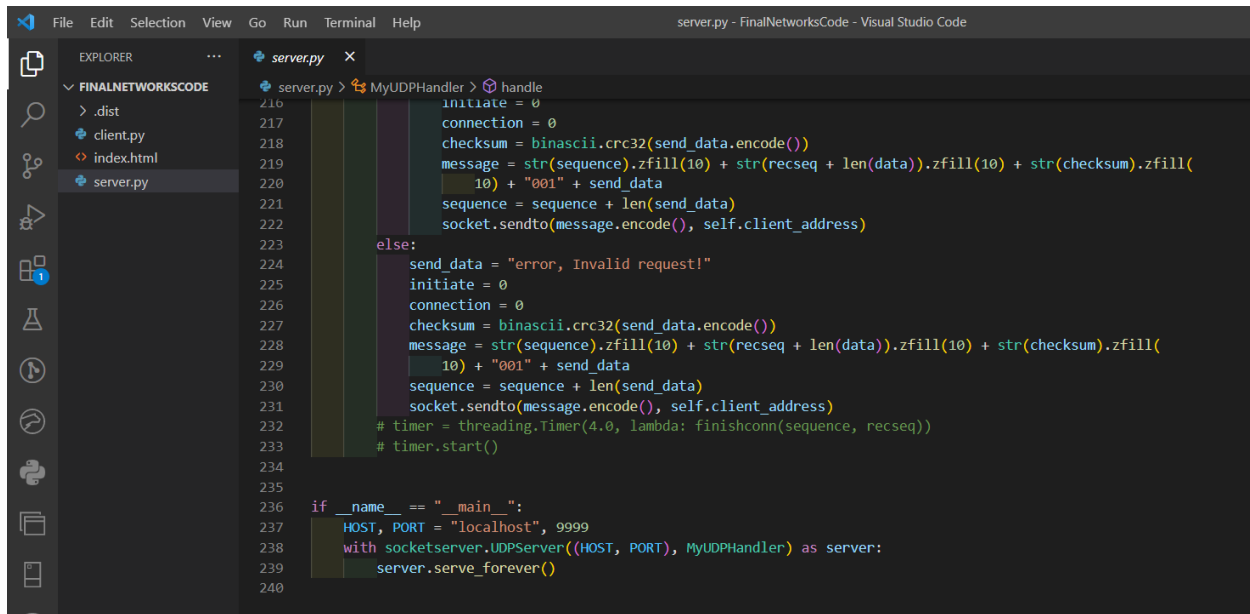
server.py - FinalNetworkCode - Visual Studio Code
EXPLORER
FINAL...
> .dist
client.py
index.html
server.py
server.py
149 datalines = datalines[1:3]
150 for line in datalines:
151     header, attributes = line.split(" ", 1)
152     if header == "Host:":
153         mydomain = attributes
154     elif header == "User-Agent:":
155         browser = attributes
156     elif header == "Accept:":
157         formats = attributes.split(',')
158     elif header == "Accept-Encoding:":
159         encodings = attributes.split(',')
160     elif header == "Accept-Language:":
161         languages = attributes.split(',')
162     elif header == "Connection:":
163         contype = attributes
164     elif header == "Content-Type:":
165         contenttype = attributes.split(" boundary=")
166         splitter = contenttype[1]
167
168 datalines = data.split('--' + splitter)
169 datalines = datalines[1:-1]
170 names = [0 for i in range(int(len(datalines)))]
171 values = [0 for i in range(int(len(datalines)))]
172 k = 0
173 for line in datalines:
174     temp = line.split('\n\n')
175     values[k] = temp[-1].split('\n')[0]
176     temp = temp[0].split('\n')
177     names[k] = temp[1]
178     k = k + 1
179 print(names)
180 print(values)
181 send_data = "end"
182 initiate = 0
183 connection = 0
184 checksum = binascii.crc32(send_data.encode())
185 message = str(sequence).zfill(10) + str(recseq + len(data)).zfill(10) + str(checksum).zfill(
186     10) + "001" + send_data
187
188 sequence = sequence + len(send_data)
189 socket.sendto(message.encode(), self.client_address)
190
191 elif connection and 'chunk' in globals():
192     if (j + 1) < len(chunk) and fin == 0:
193         if recack == sequence and checking == reccheck:
194             j = j + 1
195             err = 0
196         else:
197             sequence = sequence - len(chunk[j])
198             err = err + 1
199         send_data = chunk[j]
200         checksum = binascii.crc32(send_data.encode())
201         message = str(sequence).zfill(10) + str(recseq + len(data)).zfill(10) + str(checksum).zfill(
202             10) + "000" + send_data
203         sequence = sequence + len(send_data)
204         socket.sendto(message.encode(), self.client_address)
205     else:
206         send_data = "end"
207         initiate = 0
208         connection = 0
209         checksum = binascii.crc32(send_data.encode())
210         message = str(sequence).zfill(10) + str(recseq + len(data)).zfill(10) + str(checksum).zfill(
211             10) + "001" + send_data
212         sequence = sequence + len(send_data)
213         socket.sendto(message.encode(), self.client_address)
214     if err > 100:
215         send_data = "end"
216         initiate = 0
217         connection = 0
218         checksum = binascii.crc32(send_data.encode())
219         message = str(sequence).zfill(10) + str(recseq + len(data)).zfill(10) + str(checksum).zfill(
220             10) + "001" + send_data
221         sequence = sequence + len(send_data)
222         socket.sendto(message.encode(), self.client_address)
223     else:

```

```

server.py - FinalNetworkCode - Visual Studio Code
EXPLORER
FINALNETWORKCODE
> .dist
client.py
index.html
server.py
server.py
186     10) + "001" + send_data
187     sequence = sequence + len(send_data)
188     socket.sendto(message.encode(), self.client_address)
189
190
191 elif connection and 'chunk' in globals():
192     if (j + 1) < len(chunk) and fin == 0:
193         if recack == sequence and checking == reccheck:
194             j = j + 1
195             err = 0
196         else:
197             sequence = sequence - len(chunk[j])
198             err = err + 1
199         send_data = chunk[j]
200         checksum = binascii.crc32(send_data.encode())
201         message = str(sequence).zfill(10) + str(recseq + len(data)).zfill(10) + str(checksum).zfill(
202             10) + "000" + send_data
203         sequence = sequence + len(send_data)
204         socket.sendto(message.encode(), self.client_address)
205     else:
206         send_data = "end"
207         initiate = 0
208         connection = 0
209         checksum = binascii.crc32(send_data.encode())
210         message = str(sequence).zfill(10) + str(recseq + len(data)).zfill(10) + str(checksum).zfill(
211             10) + "001" + send_data
212         sequence = sequence + len(send_data)
213         socket.sendto(message.encode(), self.client_address)
214     if err > 100:
215         send_data = "end"
216         initiate = 0
217         connection = 0
218         checksum = binascii.crc32(send_data.encode())
219         message = str(sequence).zfill(10) + str(recseq + len(data)).zfill(10) + str(checksum).zfill(
220             10) + "001" + send_data
221         sequence = sequence + len(send_data)
222         socket.sendto(message.encode(), self.client_address)
223     else:

```



```
server.py - FinalNetworksCode - Visual Studio Code

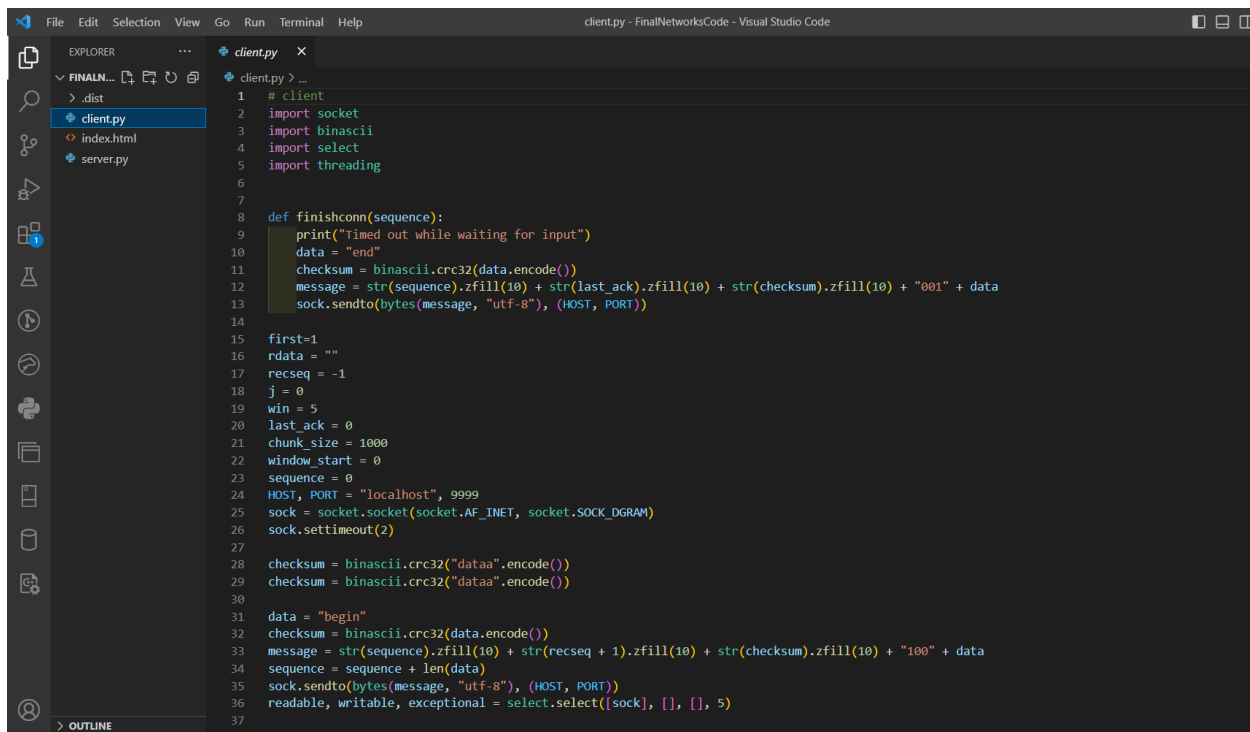
EXPLORER
  FINALNETWORKSCODE
    .dist
    client.py
    index.html
    server.py

server.py
  216
  217
  218
  219
  220
  221
  222
  223
  224
  225
  226
  227
  228
  229
  230
  231
  232
  233
  234
  235
  236
  237
  238
  239
  240

server.py > MyUDPHandler > handle
  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

EXPLORER
  FINALNETWORKSCODE
    .dist
    client.py
    index.html
    server.py

client.py
  1
  2
  3
  4
  5
  6
  7
  8
  9
  10
  11
  12
  13
  14
  15
  16
  17
  18
  19
  20
  21
  22
  23
  24
  25
  26
  27
  28
  29
  30
  31
  32
  33
  34
  35
  36
  37
  38

client.py > ...
  # 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
client.py
38
39 # if sock in readable:
40 #     received = str(sock.recv(chunk_size * 3), "utf-8")
41 # else:
42 #     print("No data received within 5 seconds")
43
44
45
46 def read_input():
47     received = str(sock.recv(chunk_size * 3), "utf-8")
48     return received
49
50
51 timer = threading.Timer(1.0, print(""))
52 timer.start()
53 try:
54     received = read_input()
55     timer.cancel()
56 except:
57     timer.cancel()
58     pass
59 if 'received' in locals():
60     # received = str(sock.recv(chunk_size * 3), "utf-8")
61     recseq = int(received[:10])
62     received = received[10:]
63     recack = int(received[:10])
64     received = received[10:]
65     reccheck = int(received[:10])
66     received = received[10:]
67     synflag = int(received[0])
68     ackflag = int(received[1])
69     finflag = int(received[2])
70     received = received[3:]
71
72 if synflag == 1 and ackflag == 1:
73     data = "start"
74     checksum = binascii.crc32(data.encode())
75     message = str(sequence).zfill(10) + str(recseq + len(received)).zfill(10) + str(checksum).zfill(10) + "010" + data

```

```

client.py - FinalNetworksCode - Visual Studio Code
client.py
72 if synflag == 1 and ackflag == 1:
73     data = "start"
74     checksum = binascii.crc32(data.encode())
75     message = str(sequence).zfill(10) + str(recseq + len(received)).zfill(10) + str(checksum).zfill(10) + "010" + data
76     last_ack = recseq + len(received)
77     sequence = sequence + len(data)
78     sock.sendto(bytes(message, "utf-8"), (HOST, PORT))
79     data = input("Enter your Get request: ")
80     err = 0
81     while True:
82         line = input()
83         if line:
84             data += "\n" + line
85             err = 0
86         else:
87             data += "\n"
88             err = err + 1
89             if err > 1:
90                 err = 0
91                 break
92     print("done")
93
94 # data = "GET /index.html HTTP/1.1\nHost: example.com\nUser-Agent: Mozilla/5.0\nAccept-Language: en-us\nAccept-Encoding: gzip, de
95 checksum = binascii.crc32(data.encode())
96 message = str(sequence).zfill(10) + str(recseq + len(received)).zfill(10) + str(checksum).zfill(
97     10) + "000" + data
98 last_ack = recseq + len(received)
99 sequence = sequence + len(data)
100 sock.sendto(bytes(message, "utf-8"), (HOST, PORT))
101 while True:
102
103     def read_input():
104         received = str(sock.recv(chunk_size * 3), "utf-8")
105         return received
106

```

```
114         timer.cancel()
115         pass
116         recseq = int(received[:10])
117         received = received[10:]
118         recack = int(received[:10])
119         received = received[10:]
120         reccheck = int(received[:10])
121         received = received[10:]
122         synflag = int(received[0])
123         ackflag = int(received[1])
124         finflag = int(received[2])
125         received = received[3:]
126         checksum = binascii.crc32(received.encode())
127         if first:
128             first=0
129             print(received.split('\n')[0].split(' ')[-2:])
130         if last_ack == recseq and checksum == reccheck:
131             data = "ack"
132             checksum = binascii.crc32(data.encode())
133             message = str(sequence).zfill(10) + str(recseq + len(received)).zfill(10) + str(checksum).zfill(
134                 10) + "010" + data
135             last_ack = recseq + len(received)
136             sequence = sequence + len(data)
137             sock.sendto(bytes(message, "utf-8"), (HOST, PORT))
138             if received != "end":
139                 rdata = rdata + received
140
141         else:
142             data = "ack"
143             checksum = binascii.crc32(data.encode())
144             message = str(sequence).zfill(10) + str(last_ack).zfill(10) + str(checksum).zfill(10) + "010" + data
145             sock.sendto(bytes(message, "utf-8"), (HOST, PORT))
146         if finflag:
147             print(rdata)
148             print("connection dropped")
149             break
150     else:
151         print("connection time out!")
```

Index.html

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4 <title>Page Title</title>
5 <meta charset="UTF-8">
6 <meta name="viewport" content="width=device-width, initial-scale=1">
7 <style>
8 body {
9     font-family: Arial, Helvetica, sans-serif;
10 }
11 </style>
12 </head>
13 <body>
14
15 <h1>My Website</h1>
16 <p>A website created by me.</p>
17
18 </body>
19 </html>
20
```

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.

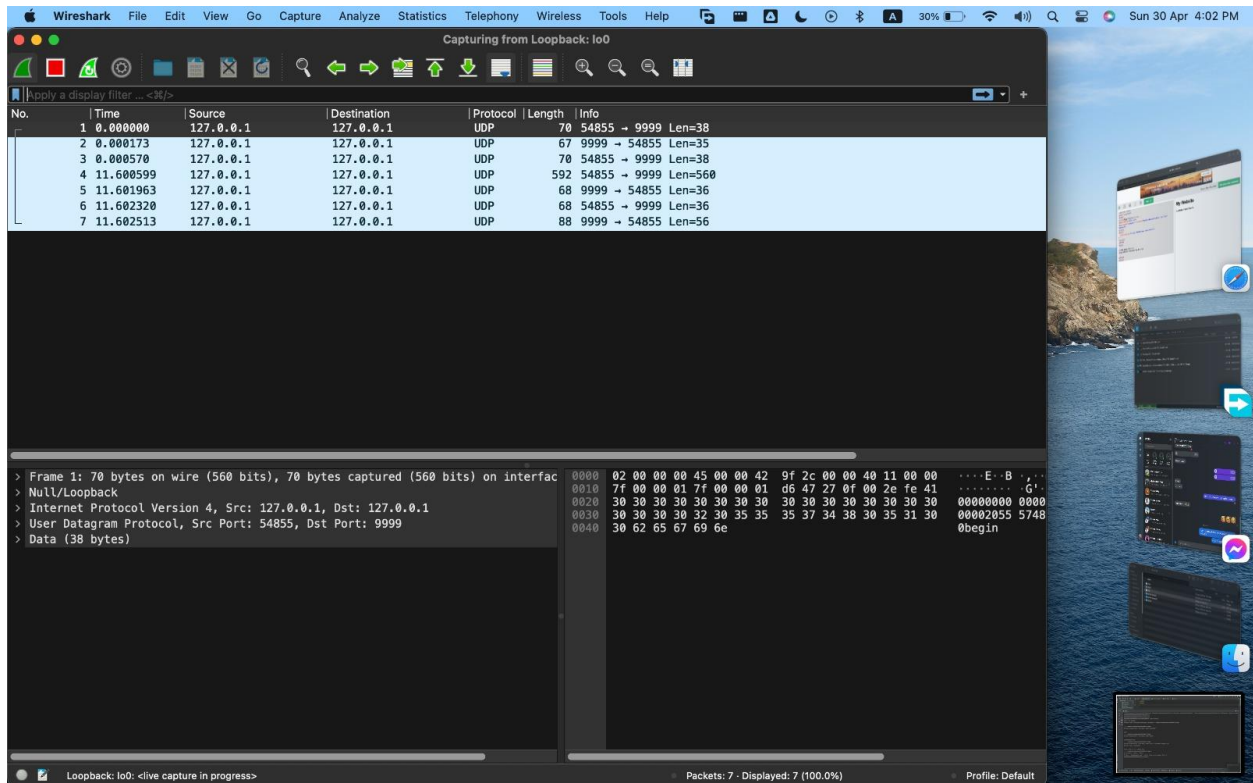
The server also includes a `finishconn()` function that is used to close the connection if no data is received from the client for a certain amount of time. The function sends an "end" message to the client and sets the `initiate` and `connection` flags to 0.

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.

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

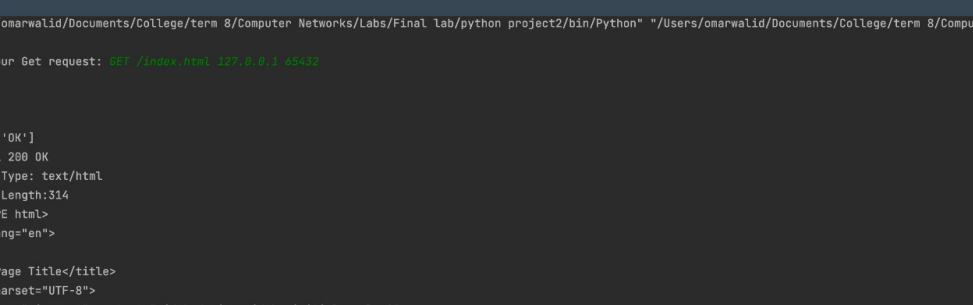
Content-Disposition: form-data; name="file"; filename="example.txt"

Content-Type: text/plain

Hello, this is an example file.

-----WebKitFormBoundary7MA4YWxkTrZu0gW--

Get Request 1



The screenshot shows a VS Code editor with a Python file named `main.py` open. The file contains a script that sends an HTTP GET request to a local server. The output of the script is displayed in the Run and Debug console. The output shows the server's response, which is an HTML page. The HTML page contains a title 'Page Title', a meta charset of 'UTF-8', a viewport meta tag, and a body with a font-family of 'Arial, Helvetica, sans-serif'. The body contains a heading 'My Website' and a paragraph 'A website created by me.'

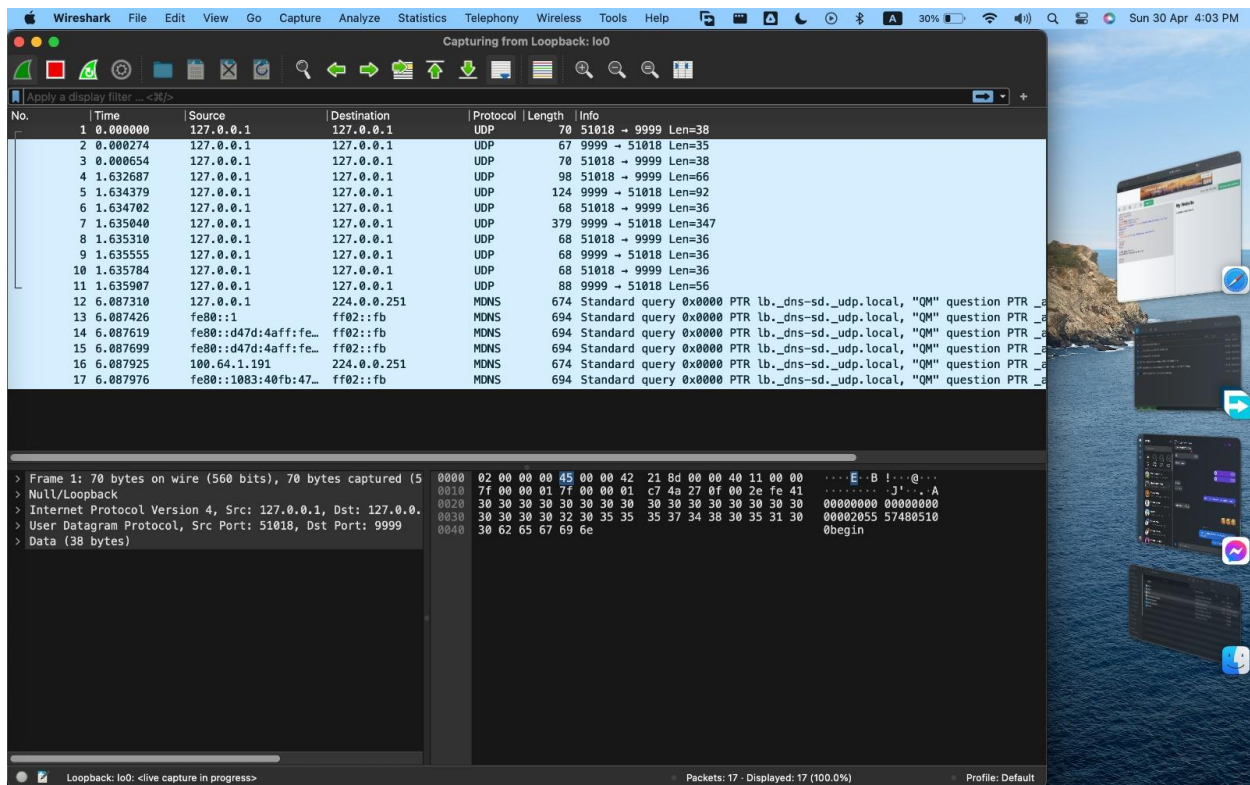
```

client main.py
main.py mine working.py old.py
if 'received' in locals(): if synflag == 1 and ackflag == 1

Run: main
/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
Version Control Run Python Packages TODO Python Console Problems Terminal Services
34:1 LF UTF-8 4 spaces Python 3.9 (python project2)

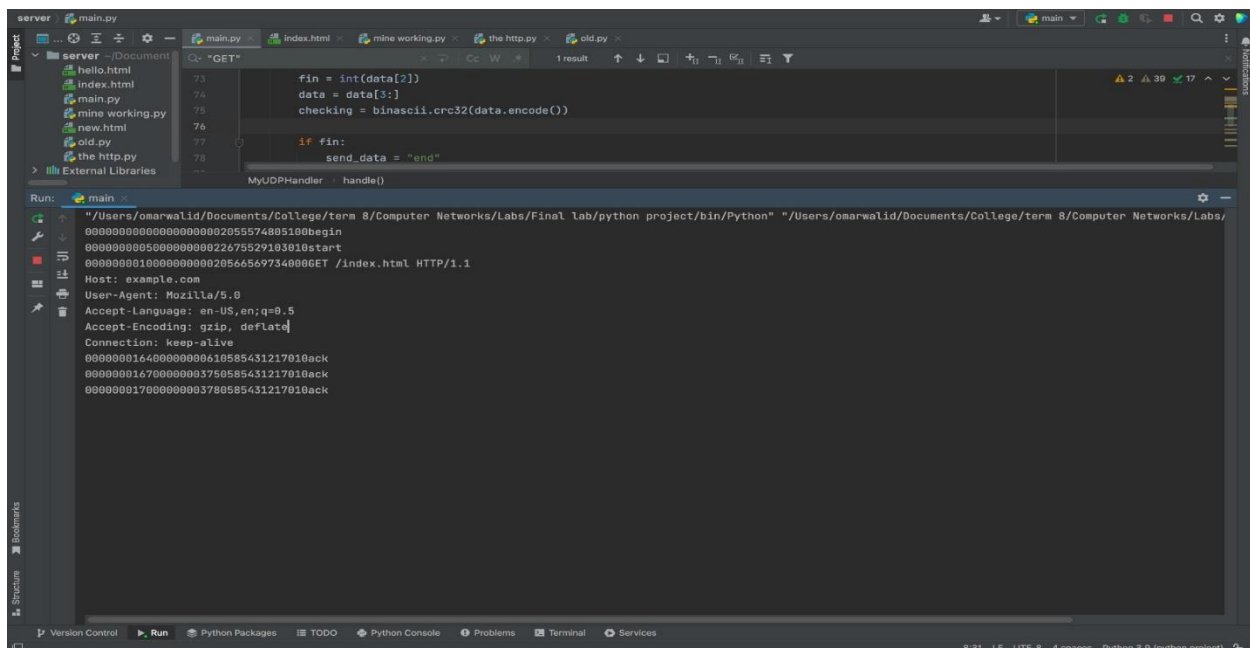
```

[illegible]



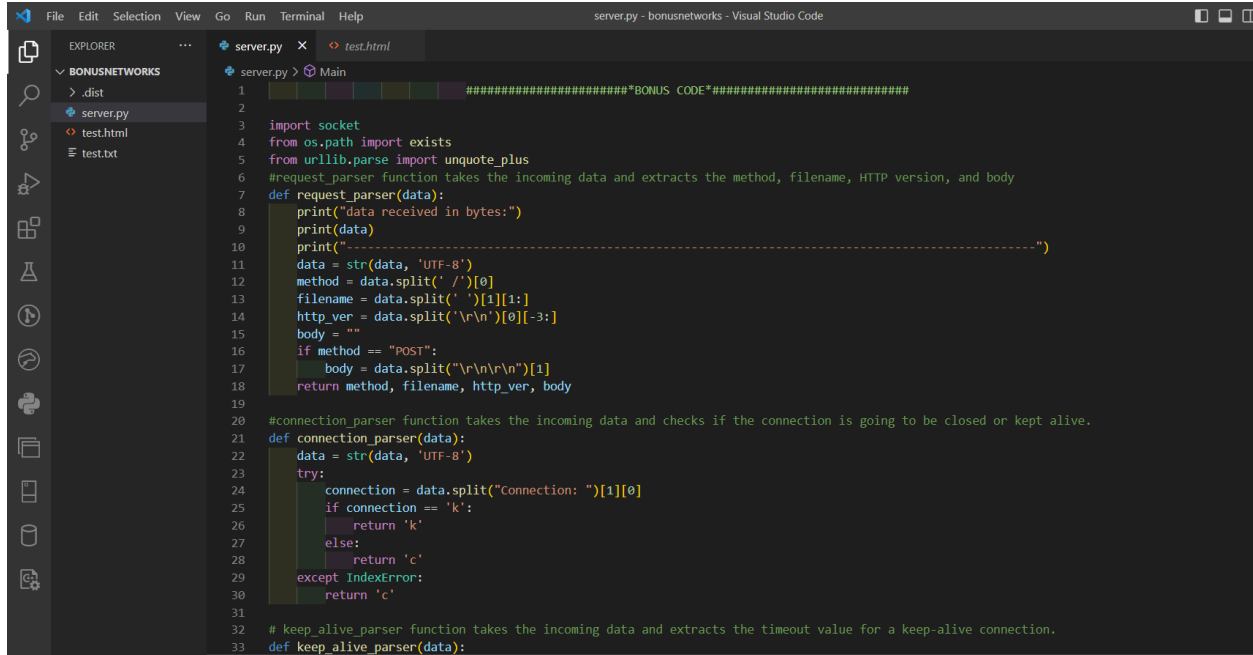
GET /index.html 127.0.0.1 65432

Get Request 2

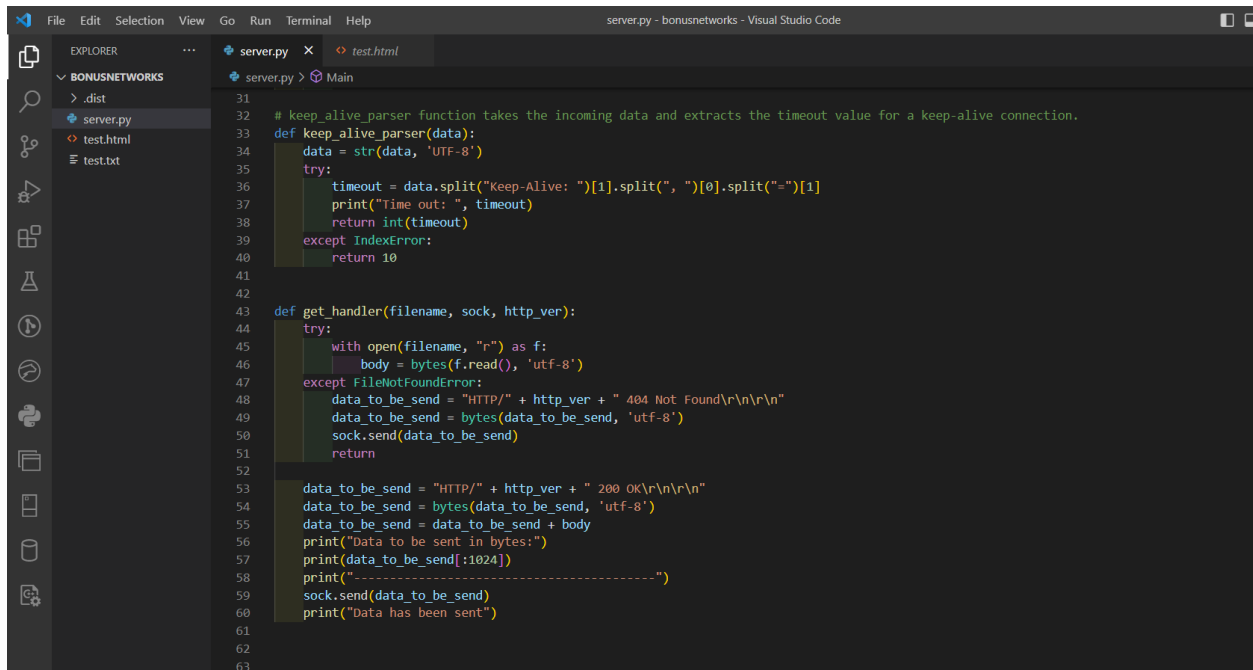


Part Two : Bonus code

Code:



```
1 #####BONUS CODE#####
2
3 import socket
4 from os.path import exists
5 from urllib.parse import unquote_plus
6 #request_parser function takes the incoming data and extracts the method, filename, HTTP version, and body
7 def request_parser(data):
8     print("data received in bytes:")
9     print(data)
10    print("-----")
11    data = str(data, 'UTF-8')
12    method = data.split(' ')[0]
13    filename = data.split(' ')[1][1:]
14    http_ver = data.split('\r\n')[0][-3:]
15    body = ""
16    if method == "POST":
17        body = data.split("\r\n\r\n")[1]
18    return method, filename, http_ver, body
19
20 #connection_parser function takes the incoming data and checks if the connection is going to be closed or kept alive.
21 def connection_parser(data):
22     data = str(data, 'UTF-8')
23     try:
24         connection = data.split("Connection: ")[1][0]
25         if connection == 'k':
26             return 'k'
27         else:
28             return 'c'
29     except IndexError:
30         return 'c'
31
32 # keep_alive_parser function takes the incoming data and extracts the timeout value for a keep-alive connection.
33 def keep_alive_parser(data):
```



```
31
32 # keep_alive_parser function takes the incoming data and extracts the timeout value for a keep-alive connection.
33 def keep_alive_parser(data):
34     data = str(data, 'UTF-8')
35     try:
36         timeout = data.split("Keep-Alive: ")[1].split(", ")[0].split("=")[1]
37         print("Time out: ", timeout)
38         return int(timeout)
39     except IndexError:
40         return 10
41
42
43 def get_handler(filename, sock, http_ver):
44     try:
45         with open(filename, "r") as f:
46             body = bytes(f.read(), 'utf-8')
47     except FileNotFoundError:
48         data_to_be_send = "HTTP/" + http_ver + " 404 Not Found\r\n\r\n"
49         data_to_be_send = bytes(data_to_be_send, 'utf-8')
50         sock.send(data_to_be_send)
51         return
52
53     data_to_be_send = "HTTP/" + http_ver + " 200 OK\r\n\r\n"
54     data_to_be_send = bytes(data_to_be_send, 'utf-8')
55     data_to_be_send = data_to_be_send + body
56     print("Data to be sent in bytes:")
57     print(data_to_be_send[:1024])
58     print("-----")
59     sock.send(data_to_be_send)
60     print("Data has been sent")
61
62
63
```



```

File Edit Selection View Go Run Terminal Help
server.py - bonusnetworks - Visual Studio Code

EXPLORER
BONUSNETWORKS
> .dist
server.py
test.html
test.txt

server.py
63
64 def post_handler(filename, body, sock, http_ver):
65     body = unquote_plus(body)
66
67     print("body to be written in file:")
68     print(body)
69     print("-----")
70     f = open(filename, "w+")
71     f.write(body)
72     print("body has been wrote")
73     f.close()
74     ok_message = "HTTP/" + http_ver + " 200 OK\r\nContent-Type: text/html\r\n\r\n"
75     ok_message = bytes(ok_message, 'utf-8')
76     sock.send(ok_message)
77     print("OK message has been sent")
78
79
80
81
82 def Main():
83     host = "127.0.0.1"
84     port = 65432
85     s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
86     s.bind((host, port))
87     print("-----")
88     print("socket binded to port", port)
89
90     s.listen()
91     print("socket is listening")
92     print("-----")
93

```

```

File Edit Selection View Go Run Terminal Help
server.py - bonusnetworks - Visual Studio Code

EXPLORER
BONUSNETWORKS
> .dist
server.py
test.html
test.txt

server.py
93
94 while True:
95     c, addr = s.accept()
96     print('Connected to:', addr[0], ':', addr[1])
97
98     while True:
99         data = b''
100         try:
101             rcv_data = []
102             rcv_data.append(c.recv(1024))
103             c.settimeout(0.5)
104             i = 0
105             while rcv_data[i]:
106                 data += rcv_data[i]
107                 i += 1
108             rcv_data.append(c.recv(1024))
109             c.settimeout(None)
110         except TimeoutError:
111             c.settimeout(None)
112
113         if not data:
114             break
115
116         method, filename, http_ver, body = request_parser(data)
117         print("method: " + method)
118         print("filename: " + filename)
119         print("http_ver: " + http_ver)
120         print("-----")
121

```

```

File Edit Selection View Go Run Terminal Help
server.py - bonusnetworks - Visual Studio Code

EXPLORER
BONUSNETWORKS
> .dist
server.py
test.html
test.txt

server.py
120
121     print("-----")
122
123     if method == "POST":
124         print("in POST")
125         post_handler(filename, body, c, http_ver)
126         print("-----")
127
128     elif method == "GET":
129         print("in GET")
130         is_exist = exists(filename)
131         if is_exist:
132             print("file exists")
133             print("-----")
134             get_handler(filename, c, http_ver)
135             print("-----")
136         else:
137             print("file not exists")
138             error_message = "HTTP/" + http_ver + " 404 Not Found\r\n\r\n"
139             error_message = bytes(error_message, 'utf-8')
140             c.send(error_message)
141             print("Error message has been sent")
142             print("-----")
143
144     if http_ver == "1.1":
145         connection = connection_parser(data)
146         if connection == 'k':
147             timeout = keep_alive_parser(data)
148             print("Waiting... 1.1, keep-alive")
149             c.settimeout(timeout)
150         else:
151             print("Not waiting... 1.1, close")
152             break
153     else:
154         print("Not waiting... 1.0")
155         break
156
157

```

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. connection 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

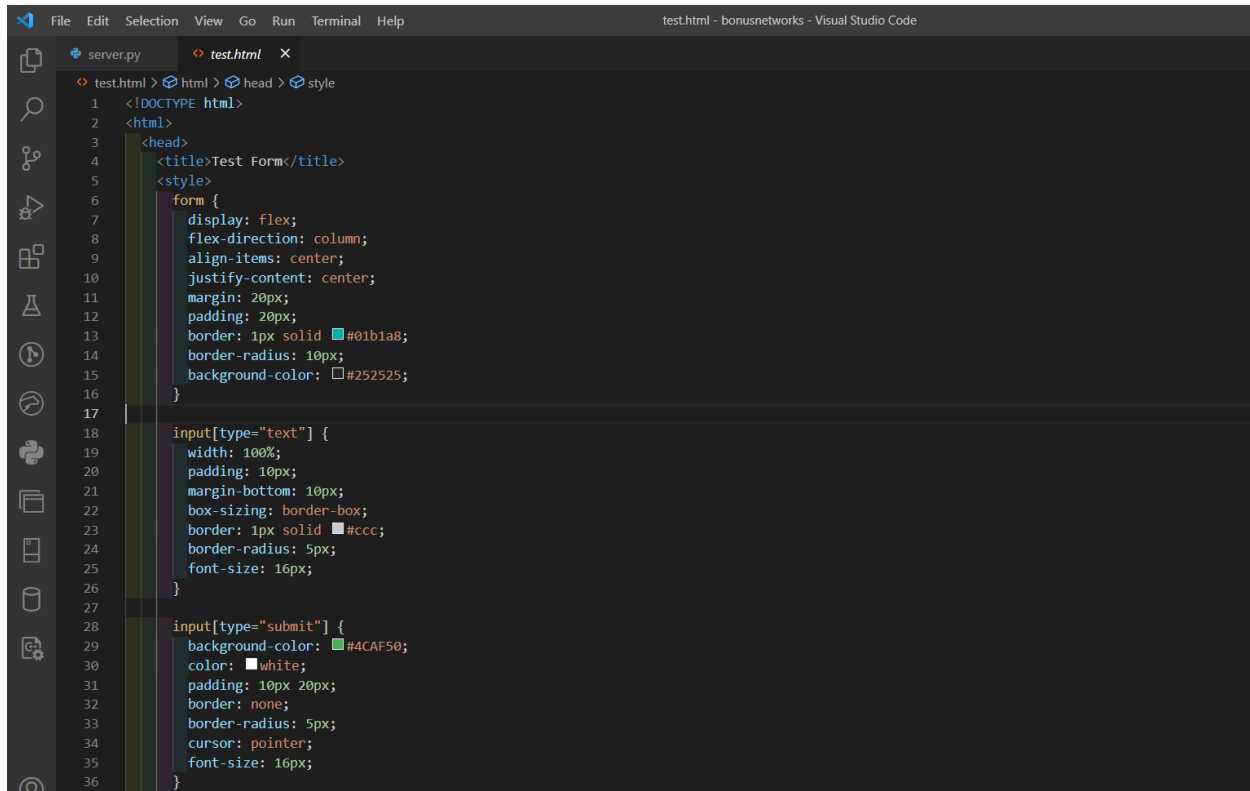
Then the main function

The main function first of all we initialized the host 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.

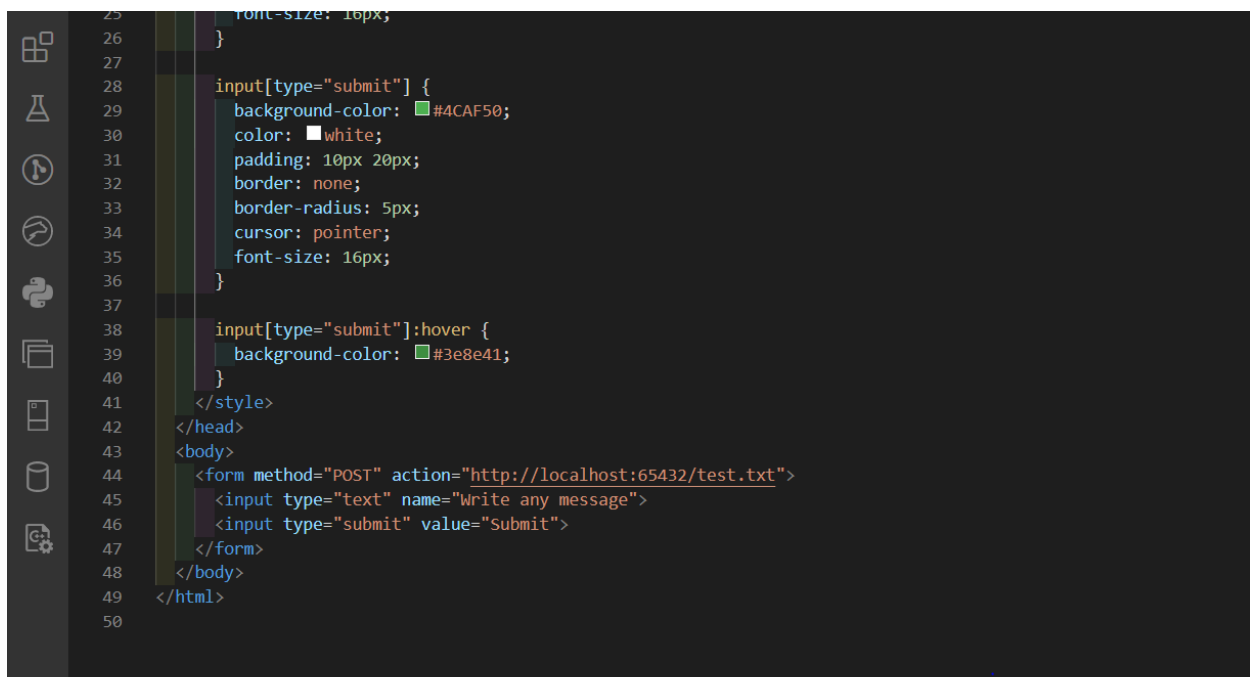
Then we call request parser to extract the method and file name and http version then we create if loop to take action either the method is GET or POST

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



```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <title>Test Form</title>
5 <style>
6   form {
7     display: flex;
8     flex-direction: column;
9     align-items: center;
10    justify-content: center;
11    margin: 20px;
12    padding: 20px;
13    border: 1px solid #01b1a8;
14    border-radius: 10px;
15    background-color: #252525;
16  }
17
18  input[type="text"] {
19    width: 100%;
20    padding: 10px;
21    margin-bottom: 10px;
22    box-sizing: border-box;
23    border: 1px solid #ccc;
24    border-radius: 5px;
25    font-size: 16px;
26  }
27
28  input[type="submit"] {
29    background-color: #4CAF50;
30    color: white;
31    padding: 10px 20px;
32    border: none;
33    border-radius: 5px;
34    cursor: pointer;
35    font-size: 16px;
36  }
```



```
25 font-size: 16px;
26 }
27
28 input[type="submit"] {
29   background-color: #4CAF50;
30   color: white;
31   padding: 10px 20px;
32   border: none;
33   border-radius: 5px;
34   cursor: pointer;
35   font-size: 16px;
36 }
37
38 input[type="submit"]:hover {
39   background-color: #3e8e41;
40 }
41 </style>
42 </head>
43 <body>
44   <form method="POST" action="http://localhost:65432/test.txt">
45     <input type="text" name="Write any message">
46     <input type="submit" value="Submit">
47   </form>
48 </body>
49 </html>
50
```

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:

```

42
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL LIGHTTRUN SQL CONSOLE
PS C:\Users\Shereen\Documents\bonusnetworks> python server.py

-----
socket binded to port 65432
socket is listening
-----
Connected to : 127.0.0.1 : 56393
data received in bytes:
b'GET /test.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-mobile: ?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/112.0.0.0 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-Fetch-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: test.html
http_ver: 1.1
-----
in GET
file exists
-----
Data to be sent in bytes:
b'HTTP/1.1 200 OK\r\n\r\n<!DOCTYPE html>\n<html>\n  <head>\n    <title>Test Form</title>\n    <style>\n      form {\n        display: flex;\n        flex-direction: column;\n        align-items: center;\n        justify-content: center;\n        margin: 20px;\n        padding: 20px;\n        border: 1px solid #01b1a8;\n        border-radius: 10px;\n        background-color: #252525;\n      }\n      input[type="text"] {\n        width: 100%;\n        padding: 10px;\n        margin-bottom: 10px;\n        border: 1px solid #ccc;\n        border-radius: 5px;\n        font-size: 16px;\n      }\n      input[type="submit"] {\n        background-color: #4CAF50;\n        color: white;\n        padding: 10px 20px;\n        border: none;\n        border-radius: 5px;\n        cursor: pointer;\n        font-size: 16px;\n      }\n      input[type="submit"]:hover {\n        background-color: #3e8e41;\n      }\n    </style>\n  </head>\n  <body>\n    <form method="POST" action="http://localhost:65432/test.txt">\n      <input
-----
Data has been sent

```

<http://localhost:65432/test.html>

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL LIGHTRUN SQL CONSOLE
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: application/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,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-fetch-site: same-origin\r\nsec-fetch-mode: navigate\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...
> .dist
server.py
test.html
test.txt
1 Write any message=shereen
```

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
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-mobile: ?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/112.0.0.0 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-fetch-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

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

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