Faculty of Engineering & Technology Electrical & Computer Engineering Department

ENCS 3320
Computer Networks

Project #1 - Report

Prebread by:

Ghazi Haj Qassem	1210778	sec 3
Diaa Badaha	1210478	sec 1
Omar Husain	1212739	sec 3

Instructor:
Dr. Abdelkarim Awad

Date: 22-12-2023

Abstract

The project is is mainly focused on the use of the application layer protocols, primarily HTTP and DNS. In addition to that, socket programming is employed to create basic client and server programs, including a web server. Some commands that are entered through the terminal may be useful in gaining knowledge about the path to a destination or some information about it. Wireshark, a common packet tracer application, is also employed to capture some packets, mainly DNS messages.

Table of Contents

Table of figures

Part I

Point 1

The following commands are given with their brief description:

- ping: A command used to test the reachability of a host by computing the round-trip time, time to live (TTL) of packets and statistics for packet loss.
- tracert: A command used to trace the route from a source to a destination by sending three packets along each hop (to each device on the path) and computing the round-trip time for each packet.
- nslookup: A command used to query the DNS server for DNS records.
- telnet: A text-based network protocol for 8-bit bidirectional communication for remote access.

Point 2.1

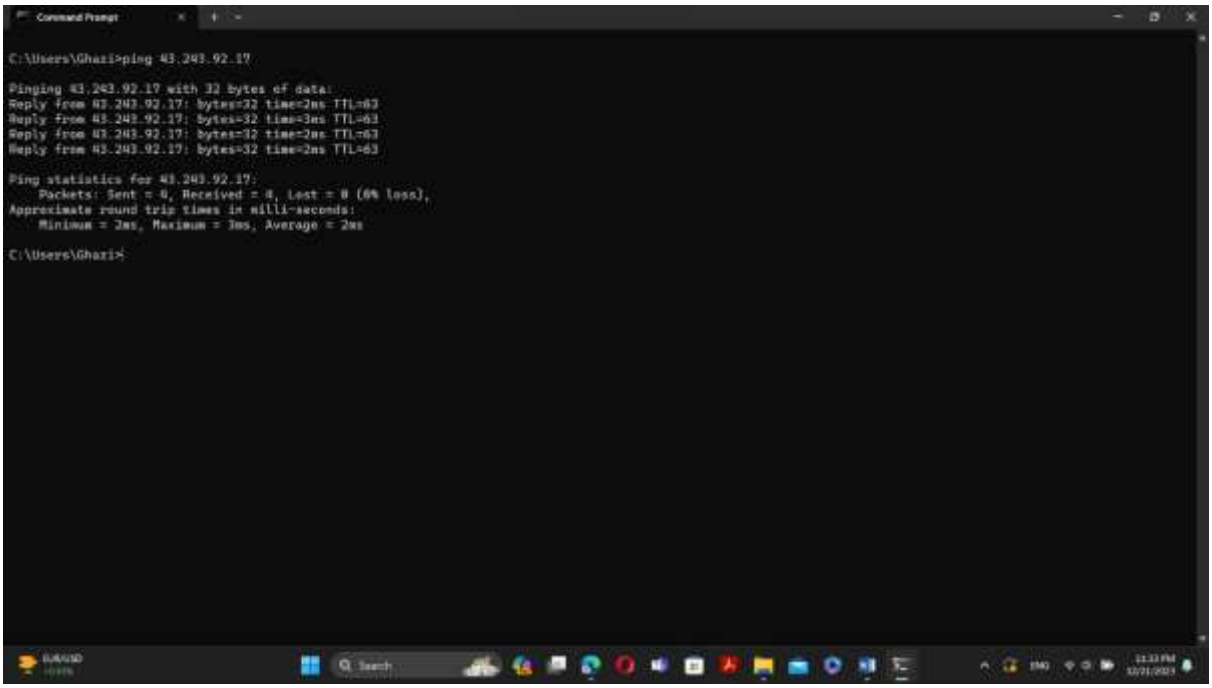


Figure 1: results of pinging a device in the same network

As seen, after I pinged my smartphone, the results show that 4 packets, each 32 bytes in size, were sent to my smart phone, and none was lost. The time 2-3 ms displayed is the round-trip time, which is the time required for a packet to reach the destination added to the time required to receive a reply from the destination to acknowledge the reception of packets. TTL is the time to live for packet, which is the number of network devices along the path that the

packet can pass through. If a packet travels beyond TTL, then it is discarded to prevent it from indefinitely circulating in the network. Thus, a packet here is allowed to pass through 63 devices.

Point 2.2

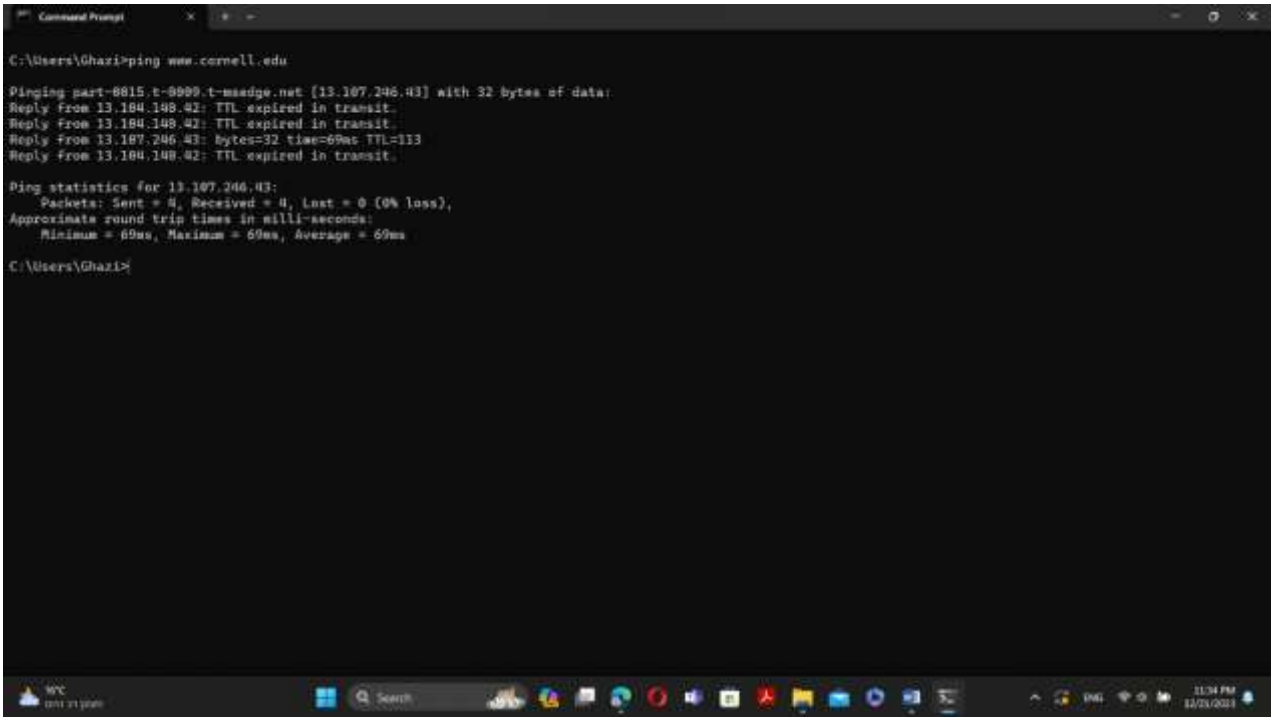


Figure 2: results of pinging www.cornell.edu

In contrast to pinging a device in the same network, pinging www.cornell.edu , which is a more distant location, yielded higher time results. Three packets were discarded, since they passed through devices more than their specified TTL. However, one packet arrived, and its round-trip time (RTT) is 69 ms, and TTL = 113. This means that it took a lot of time and passed through much more devices compared than the device in the same network that was pinged. The results show that the destination is likely to be in the USA, since RTT is 35 times the RTT that resulted from pinging a device in the same network, and “TTL expired in transit” is yet another indicator. The IP address was also checked to confirm the case, and it turned out to be in the USA.

Point 2.3

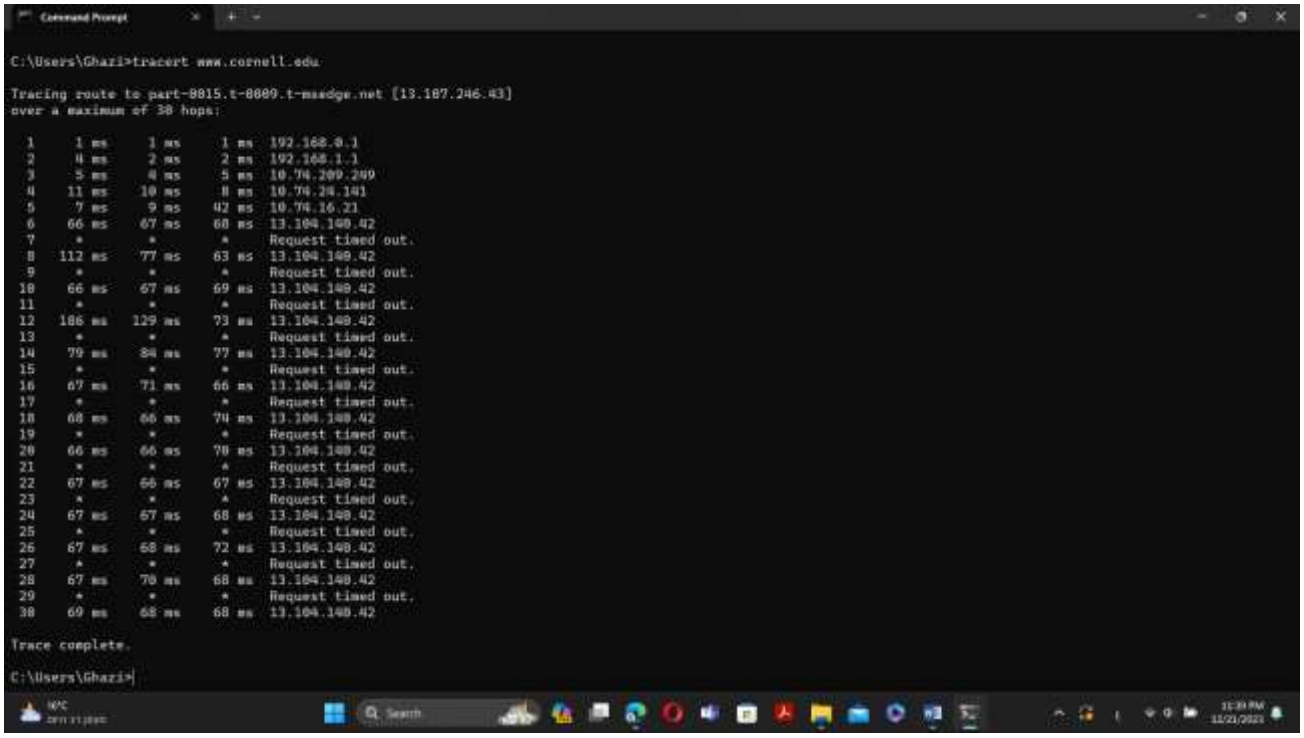


Figure 3: Results of using the command "tracert www.cornell.edu"

As shown above, the packets took 30 hops to reach the destination, meaning in the last hop, the three packets had to pass through 30 network devices to reach the destination host. The first column represents the hop number. The next three columns contain the round-trip time (RTT) for each of three packets sent in this hop. The last column is the destination (be it a host or a network device) that each of the three packets in a certain hop are sent to. Some hops display “*” and “Request timed out.”. This means that the router did not respond with a message within the specified time limit in the three cases. Another thing to note is that in the last 25 hops, the destination IP addresses are the same. This may be a security measure to obscure IP addresses of each device as having the same IP address of the host destination.

Point 2.4

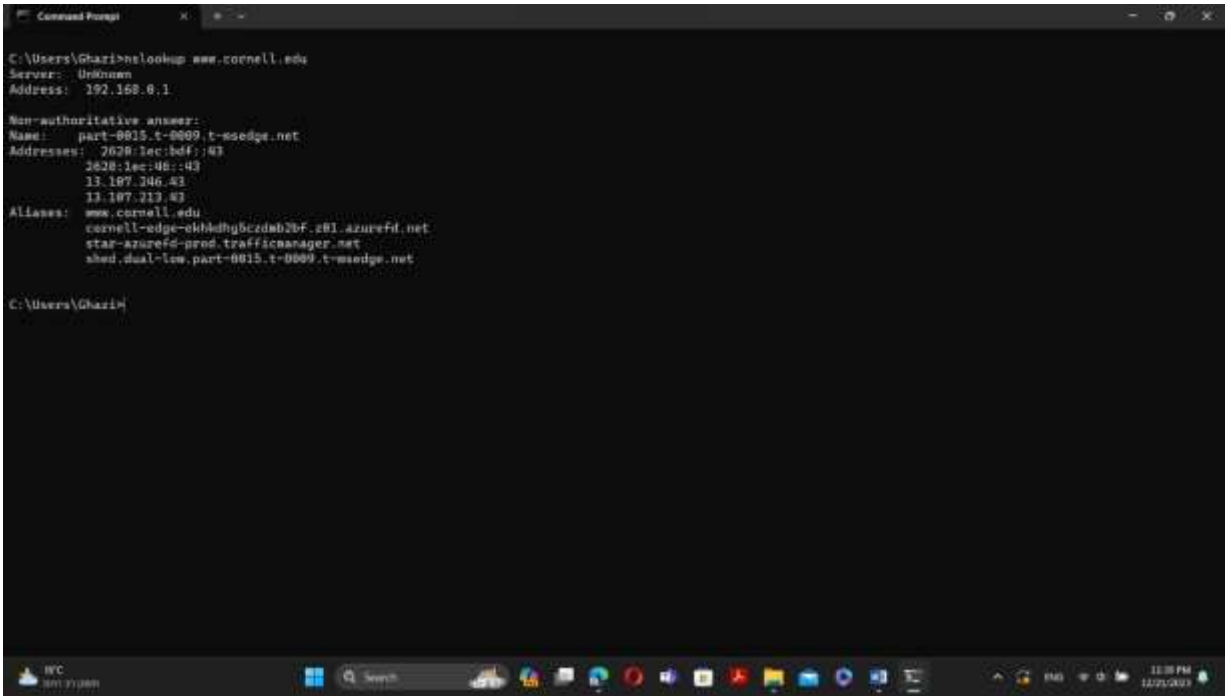


Figure 4: results of using the command "nslookup www.cornell.edu"

It is seen that the DNS server sends two types of DNS records, A and CNAME. An A record is for the IP addresses of that hostname, while a CNAME record is for aliases of that hostname. These records came from a non-authoritative DNS server. An authoritative DNS server is the kind of server that stores these types of records that are mapped to this hostname. Records maybe were cached at this non-authoritative server, since the answer was obtained from it instead of an authoritative one.

Point 3

The following captured DNS messages are as a result of pinging www.cornell.edu.

The below captured message using Wireshark shows some DNS query, requesting the IP address mapped to URL www.cornell.edu as a result of being pinged. Of course, it queries the DNS server, which turns out to be a DNS server inside my local network, for an A record, which stores the IP address of that hostname.

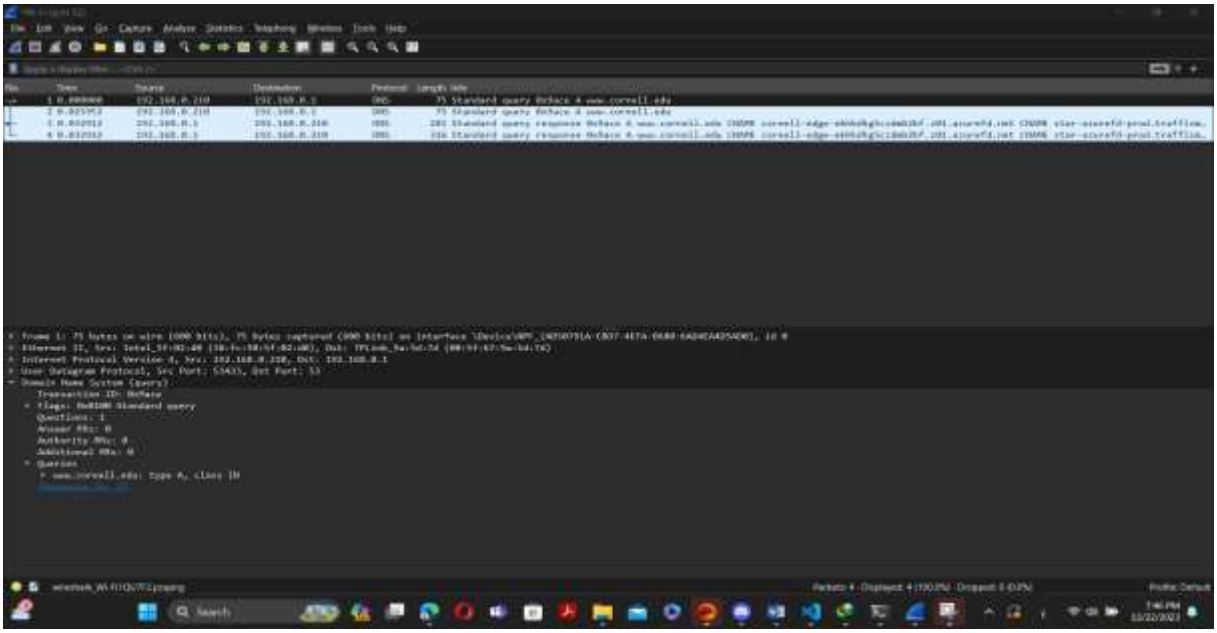


Figure 5: DNS query message captured by Wireshark

The below DNS query response message contains two types of records, namely A and CNAME. These hold an IP address and an alias, respectively. This was shown in the ping results, where the addresses and aliases were displayed.

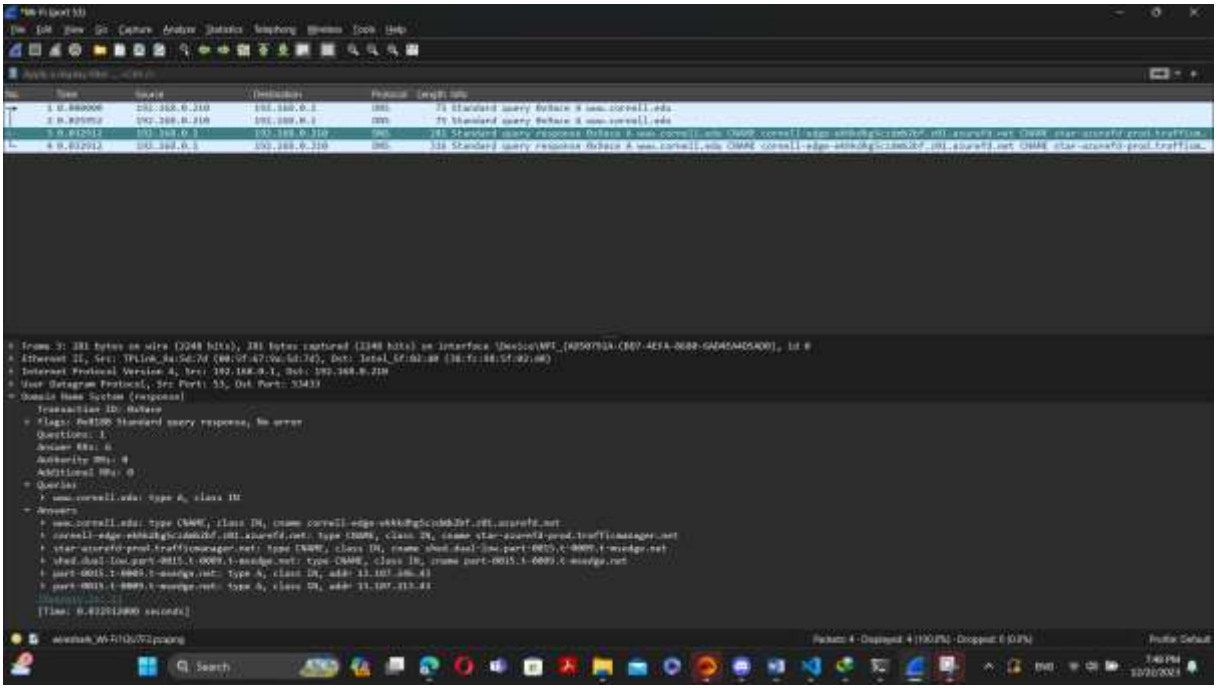


Figure 6: DNS query response message captured by Wireshark

Part II

Tests

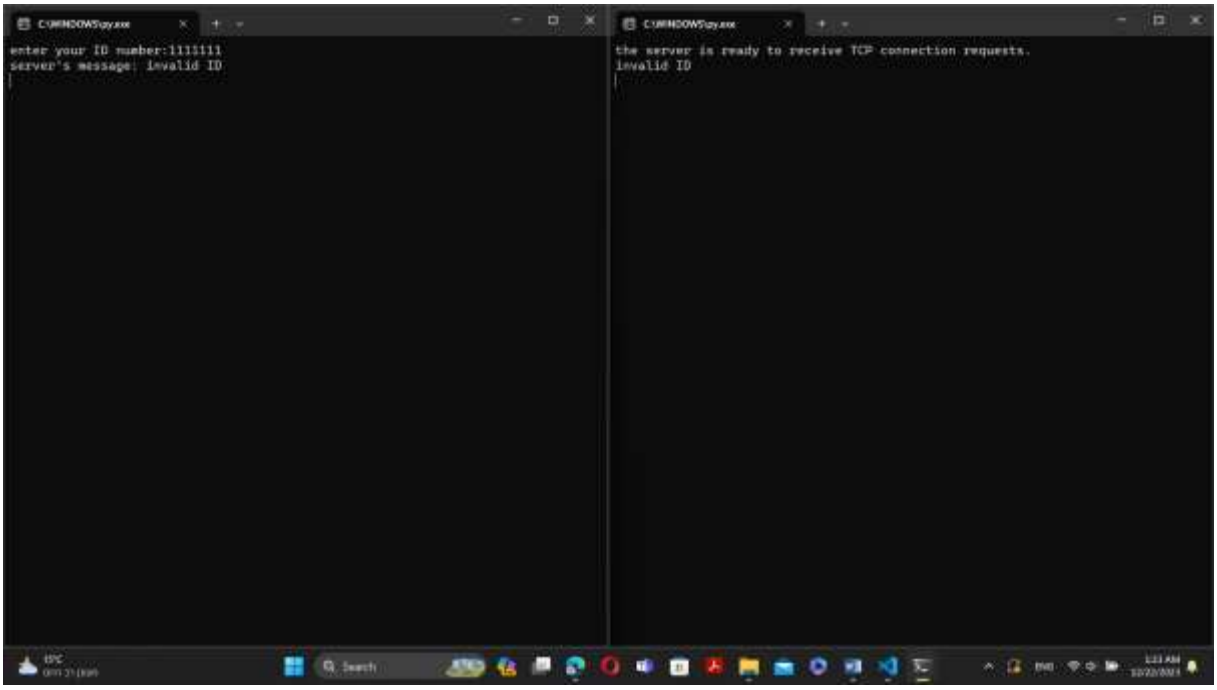


Figure 7: Testing the server and client in part 2 when entering an invalid ID

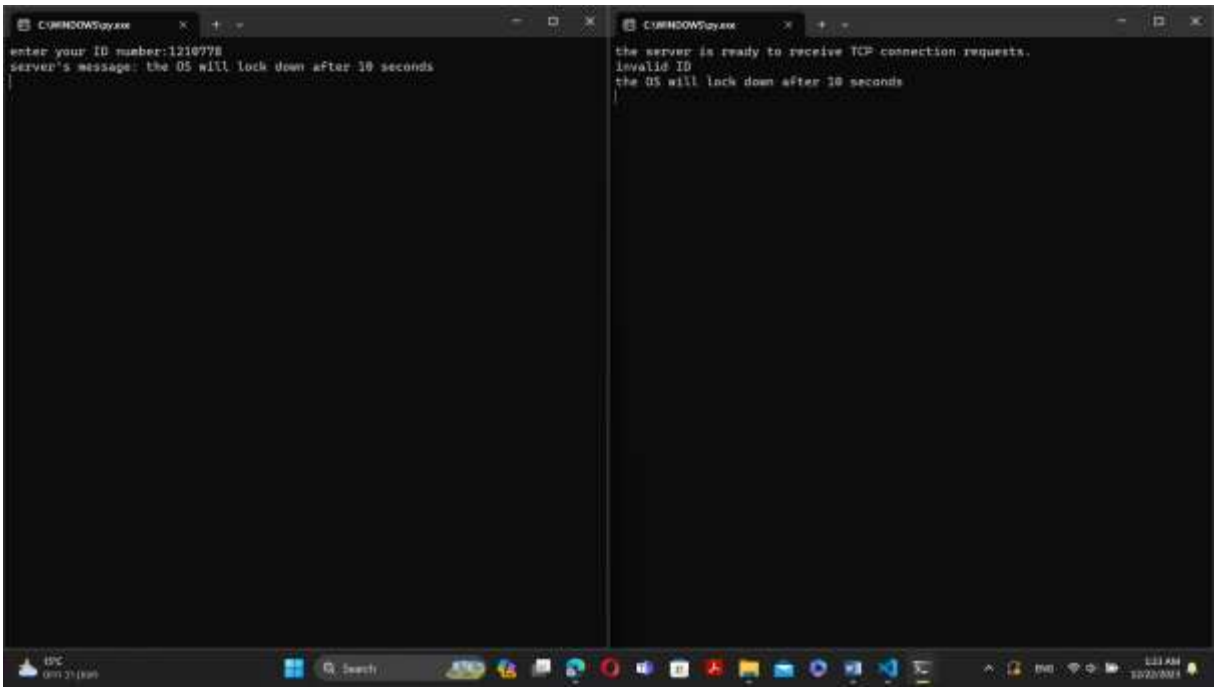


Figure 8: Testing the server and client in part 2 when entering a valid ID

Part III

Point 0

The Content-Type entity-header field indicates the media type of the entity-body sent to the recipient or, in the case of the HEAD method, the media type that would have been sent had the request been a GET. And we summarized it in the web page as shown in figure 11.

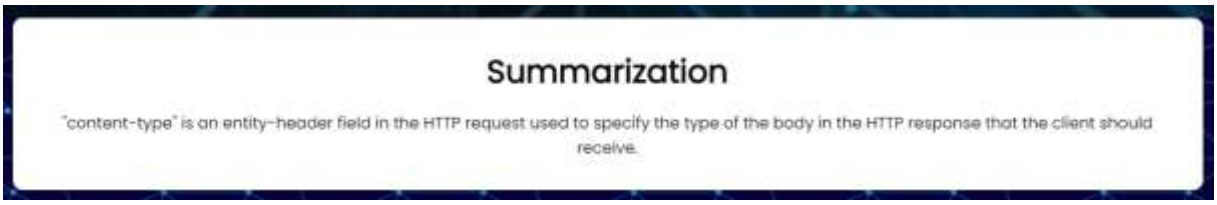


Figure 9: Summarization of point 0

Point 1

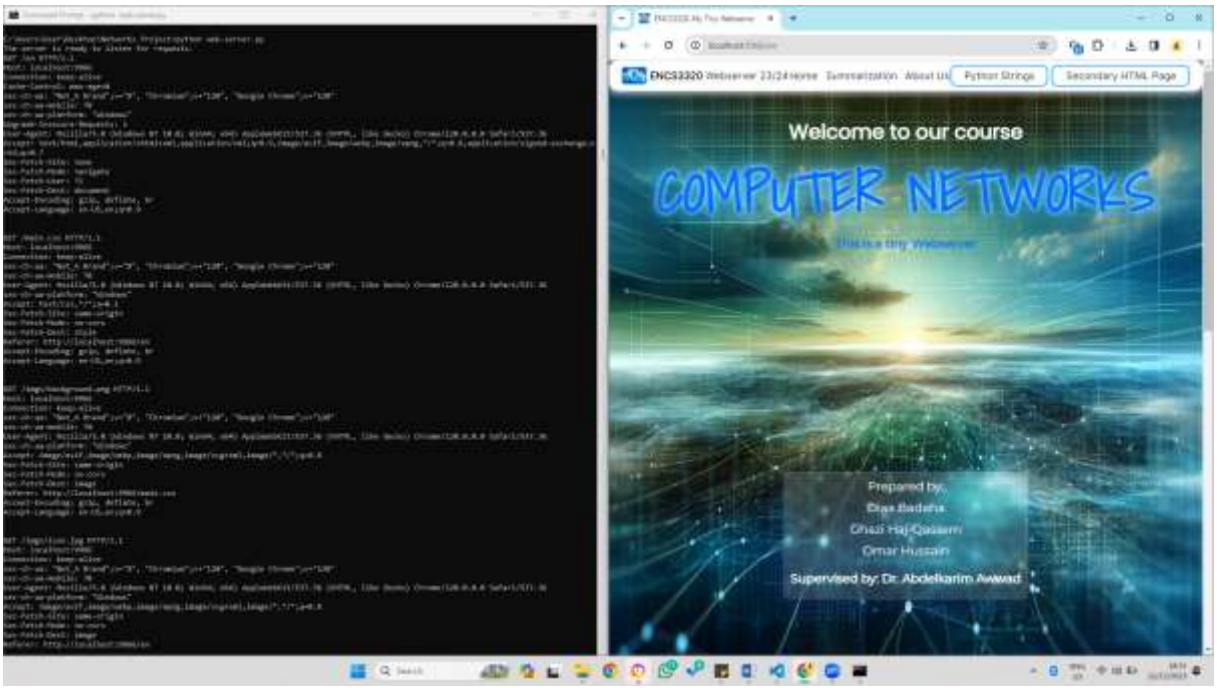


Figure 10: Requesting "/" through the browser and printing the HTTP response in the terminal window

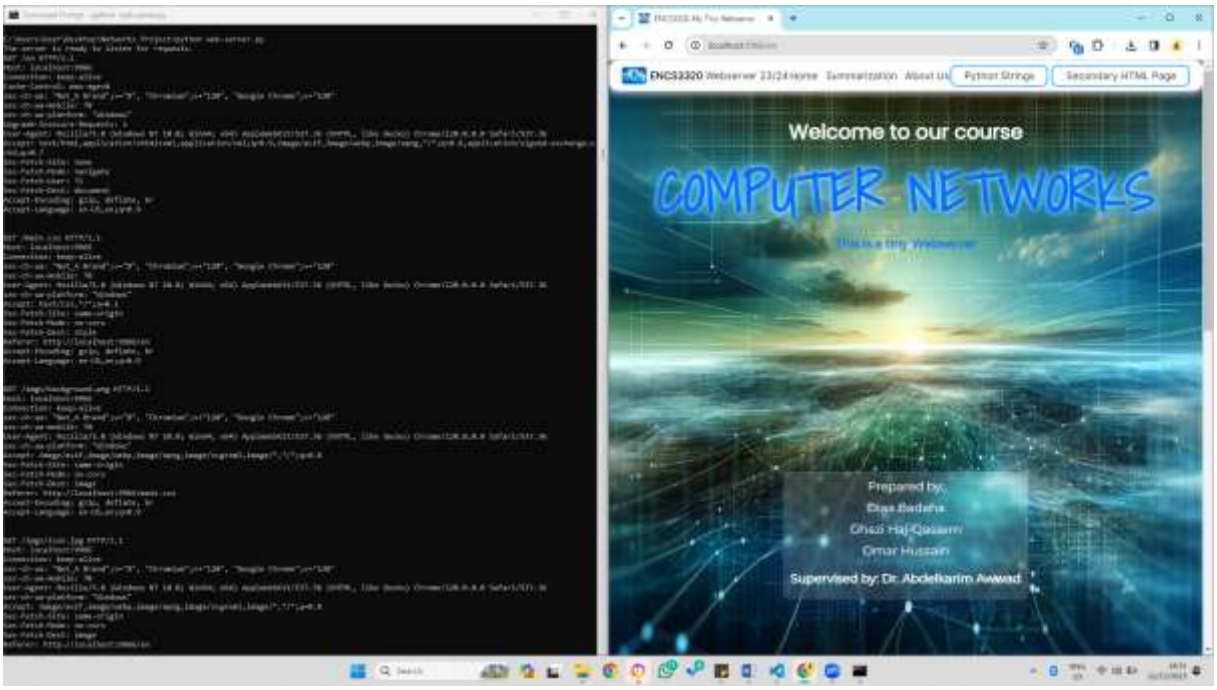


Figure 12: Requesting "/index.html" through the browser and printing the HTTP response in the terminal window

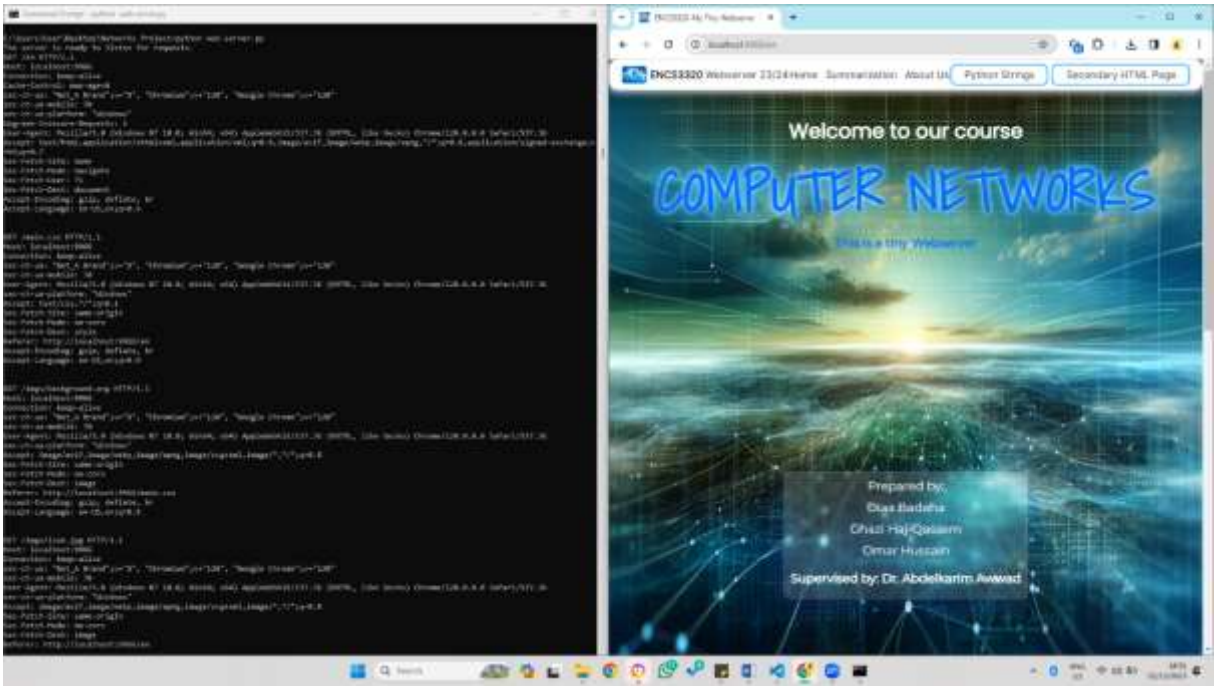


Figure 11: Requesting "/main_en" through the browser and printing the HTTP response in the terminal window

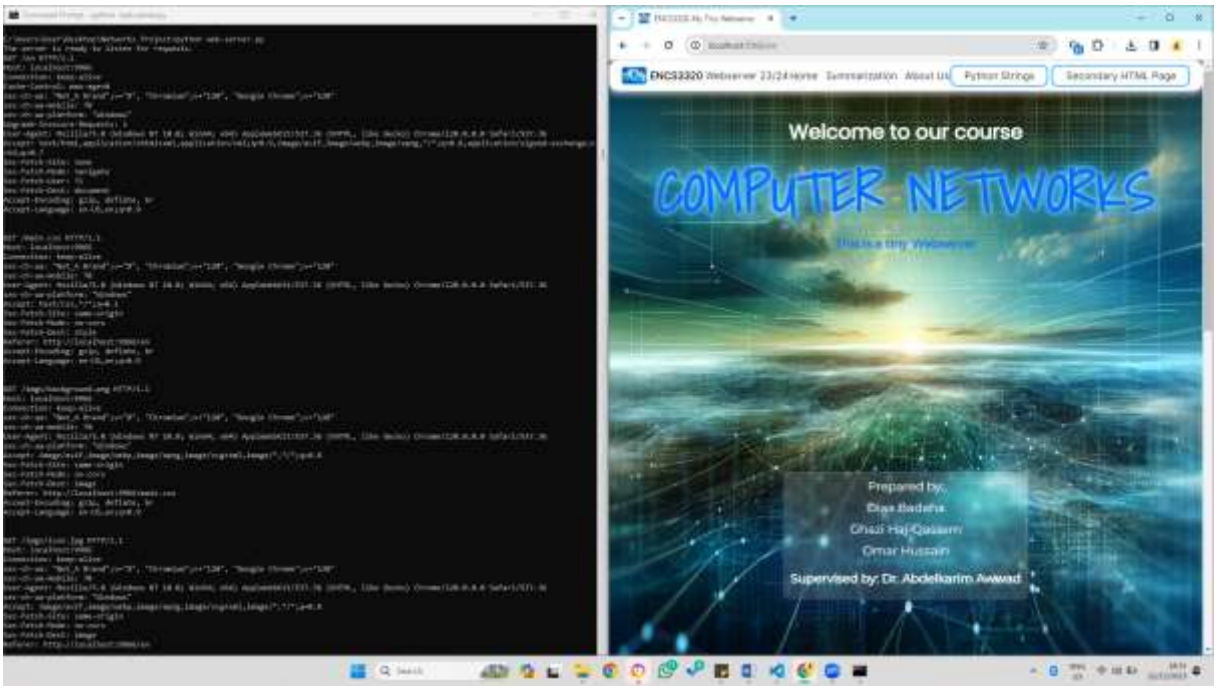


Figure 13: Requesting "/en" through the browser and printing the HTTP response in the terminal window

Point 2

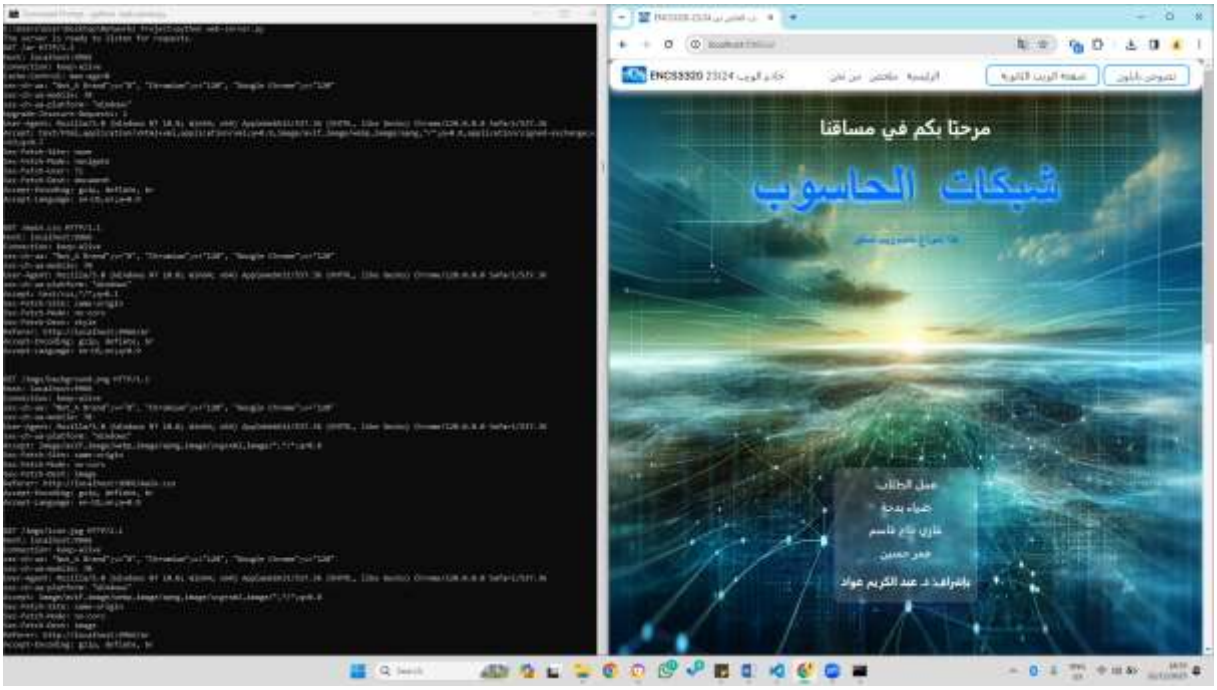


Figure 14: Requesting "/ar" through the browser and printing the HTTP response in the terminal window

Point 3

Figure 15: Requesting a file .html "secondary.html" through the browser and printing the HTTP response in the terminal window

Point 4

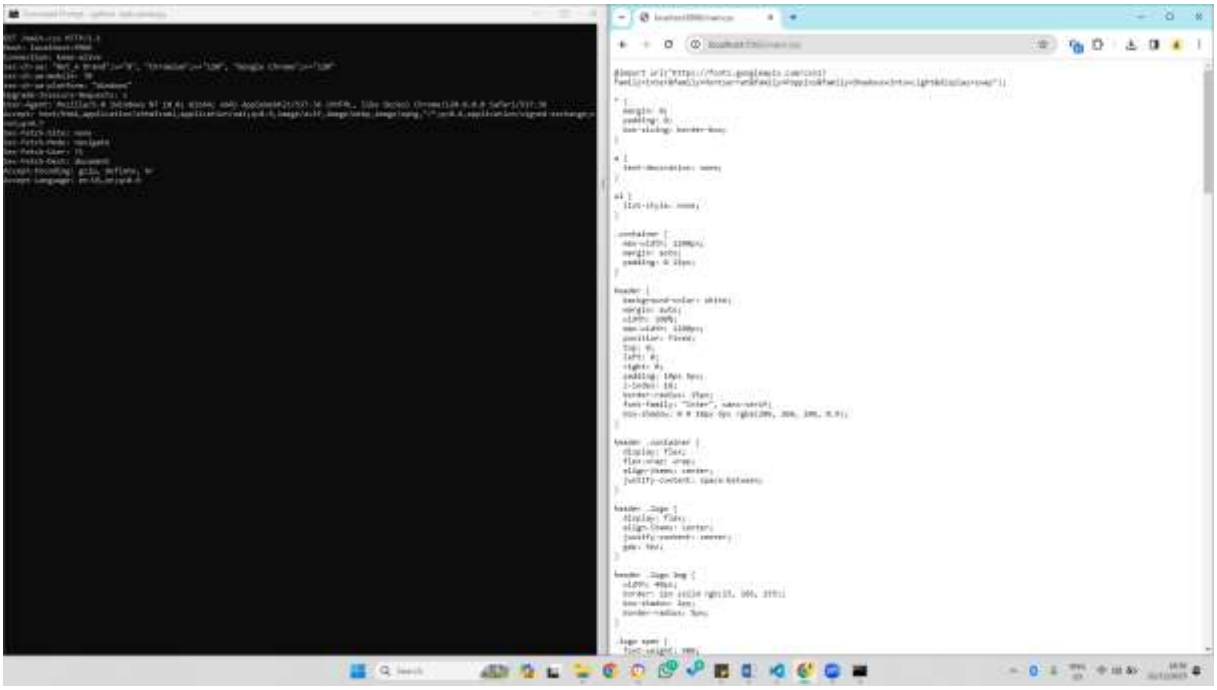


Figure 16: Requesting a file .css "main.css" through the browser and printing the HTTP response in the terminal window

Point 5

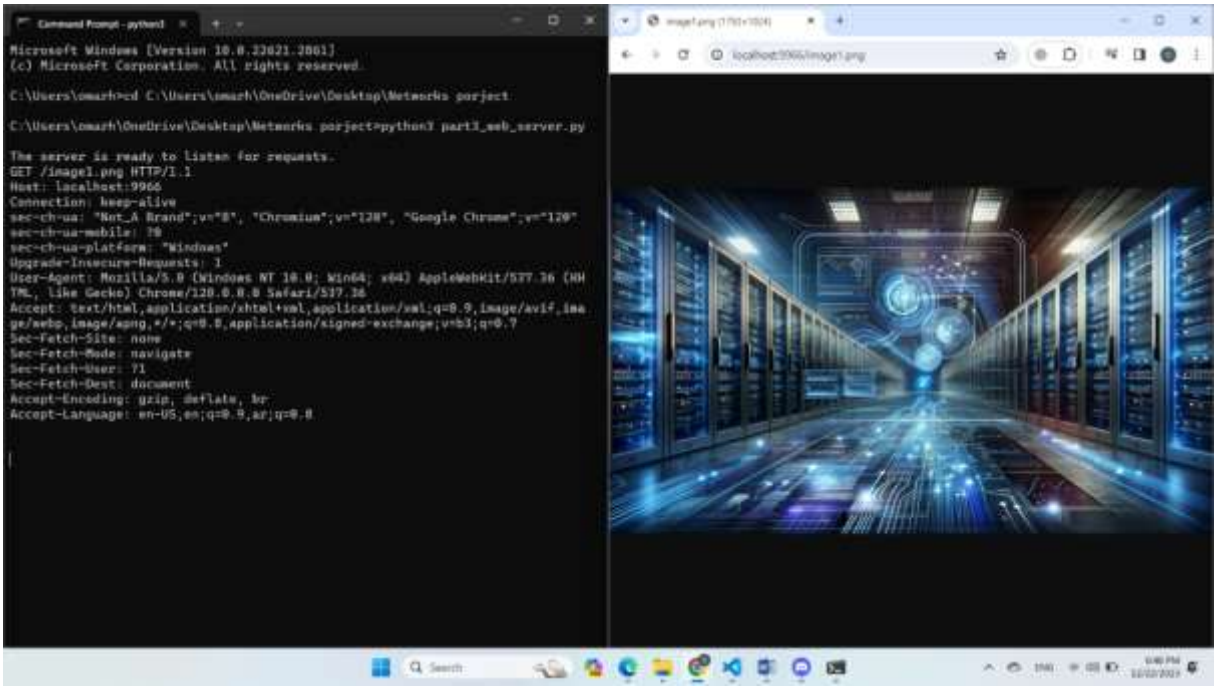


Figure 17: Requesting a png image through the browser and printing the HTTP response in the terminal window

Point 6

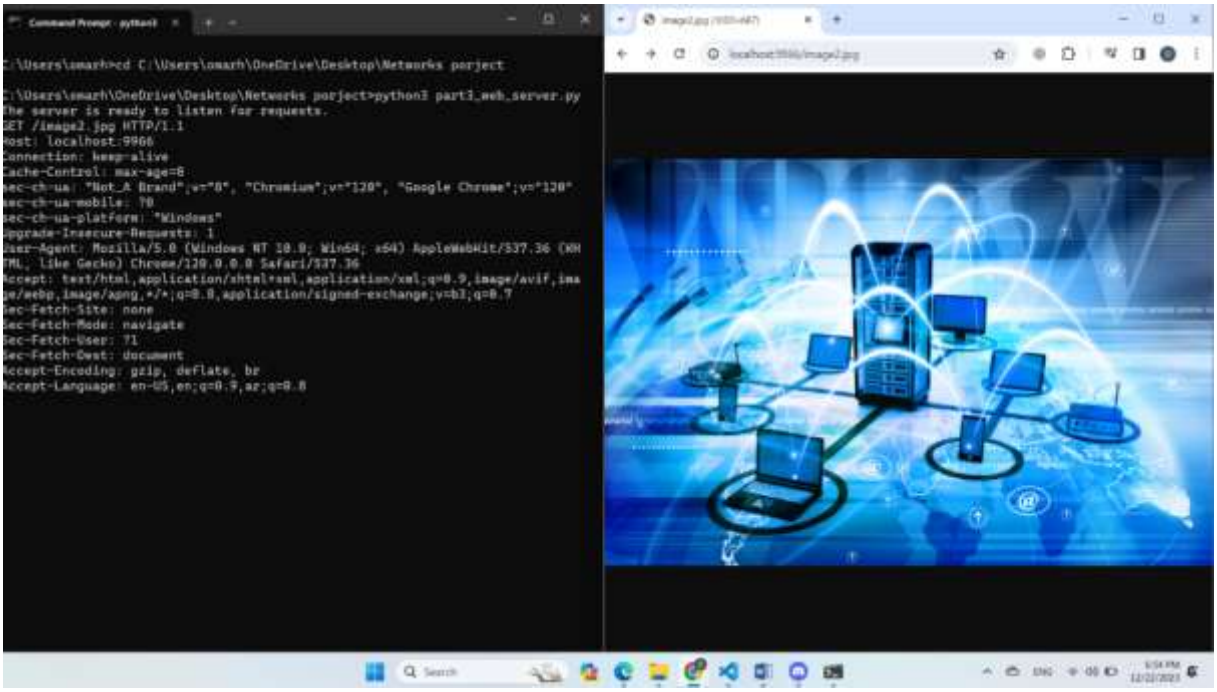


Figure 18: Requesting a jpg image through the browser and printing the HTTP response in the terminal window

Point 7

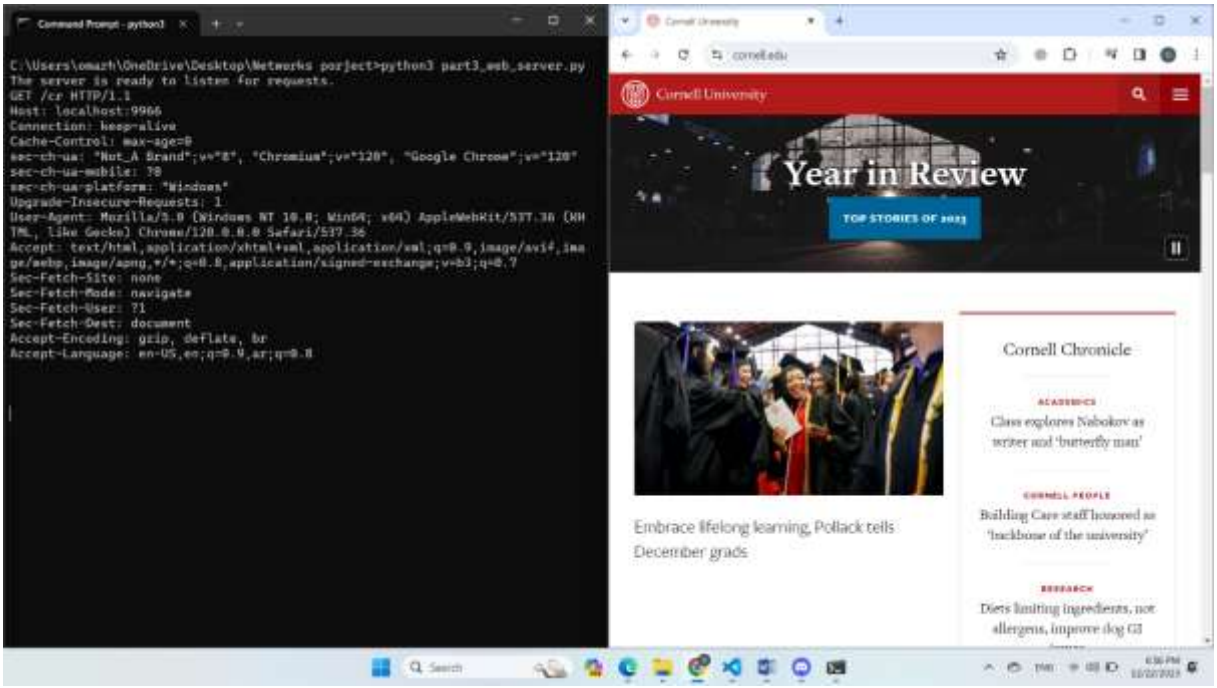


Figure 19: Requesting redirection to cornell.edu through /cr and printing the HTTP response in the terminal window

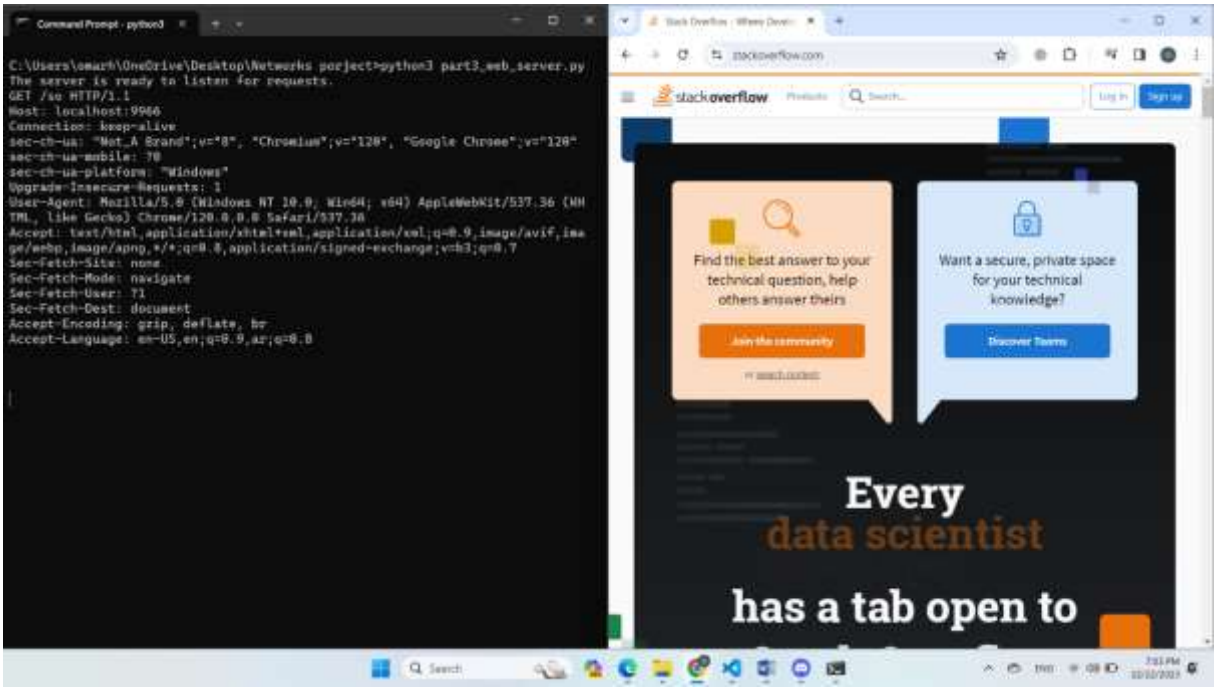


Figure 20: Requesting redirection to stackoverflow.com through /so and printing the HTTP response in the terminal window

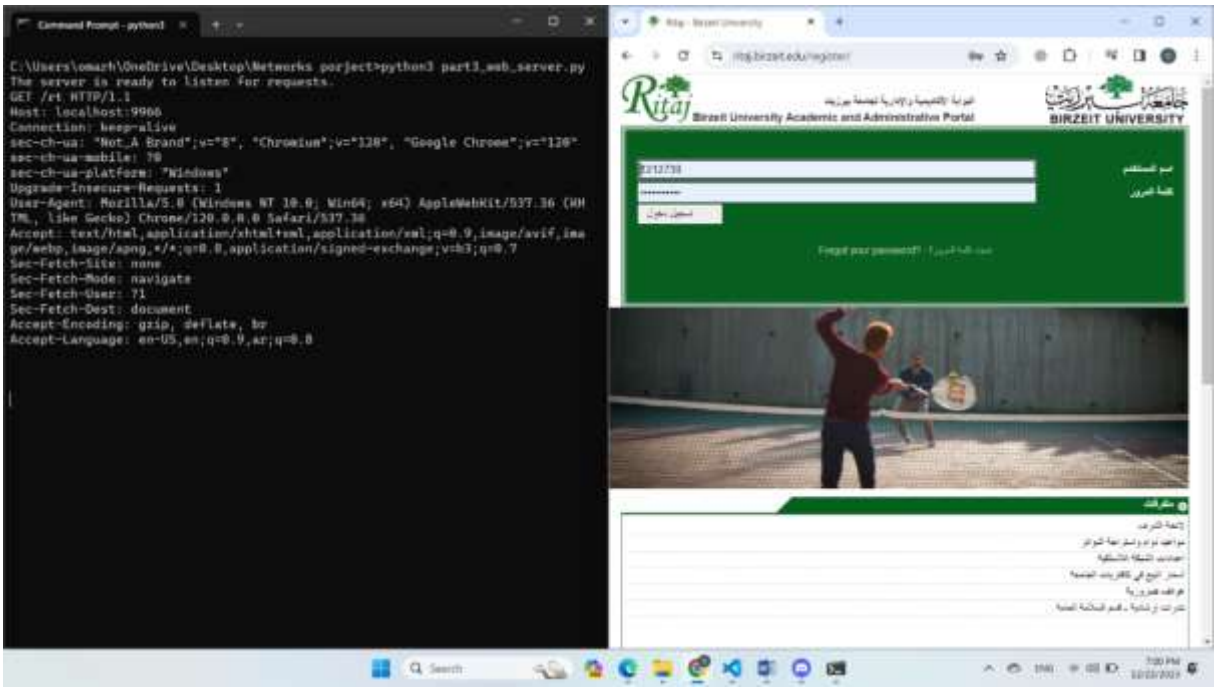


Figure 21: Requesting redirection to ritaj.birzeit.edu through /rt and printing the HTTP response in the terminal window

Point 8

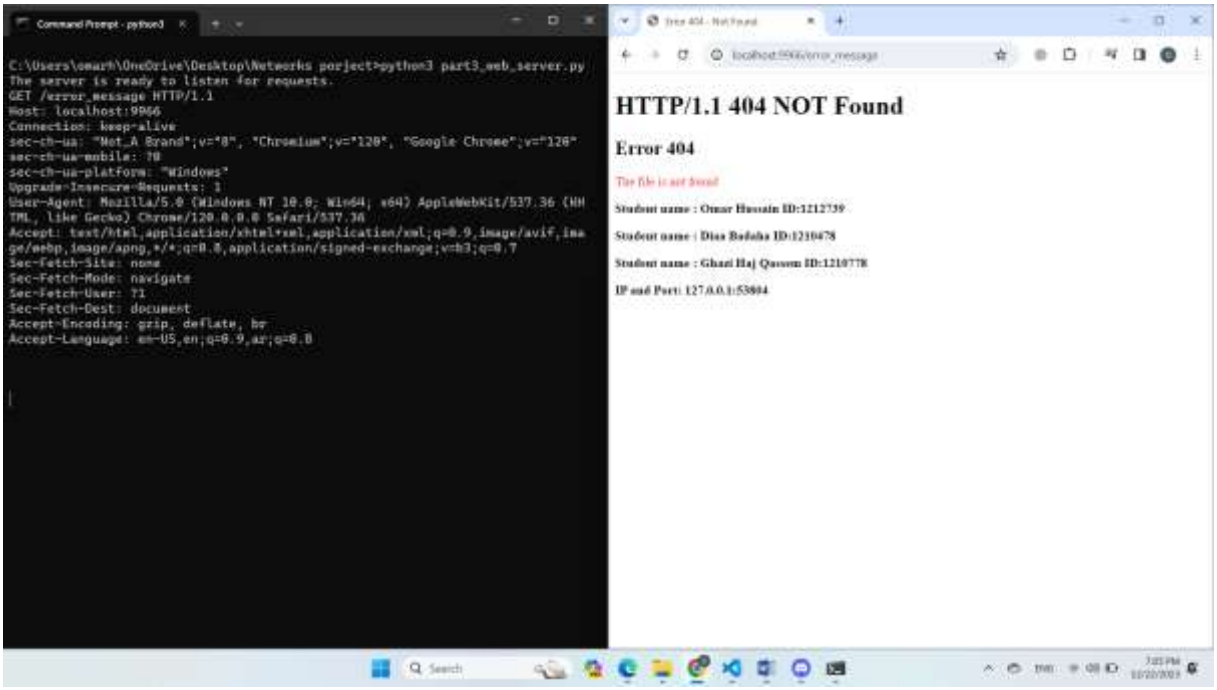


Figure 22: Requesting an invalid file and printing the HTTP response in the terminal window

Point 9

Done in the previous points…

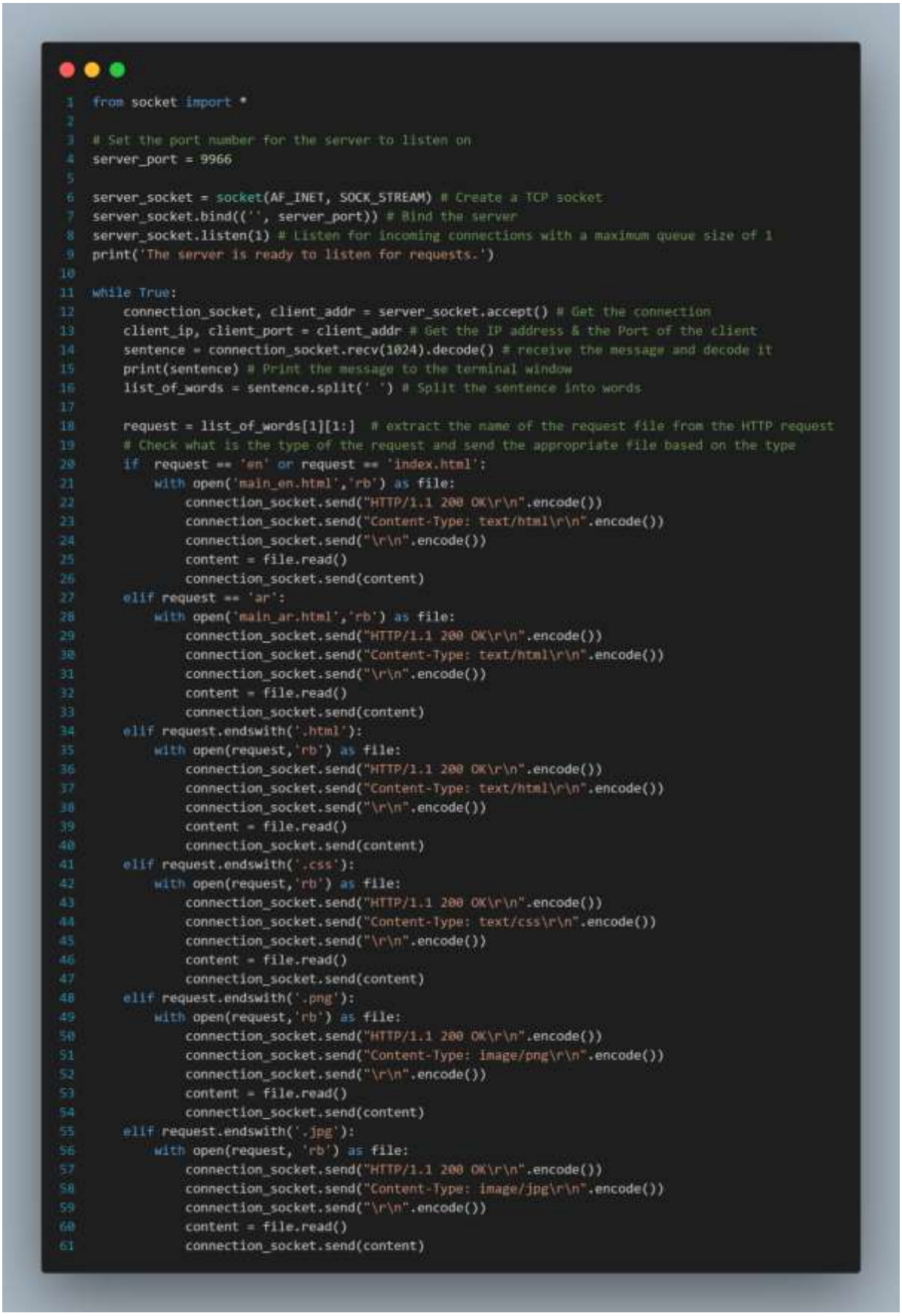
Appendix

```python
from socket import *

server_name = 'localhost' # The host is the same device running both server and client programs
server_port = 9955

client_socket = socket(AF_INET, SOCK_STREAM) # Create a client socket: AF_INET is for IPv4, and SOCK_STREAM for TCP
client_socket.connect((server_name, server_port)) # Request a TCP connection to the server process specified with the given IP address and port number

message = input('Enter your ID number:') # Prompt the user to enter his/her ID number
client_socket.send(message.encode()) # Send the message to the server

new_message = client_socket.recv(2048) # Receive a message from the server

print('Server\'s message:', new_message.decode()) # Decode the message received from the server and print it

client_socket.close()
```

Figure 23: Client code

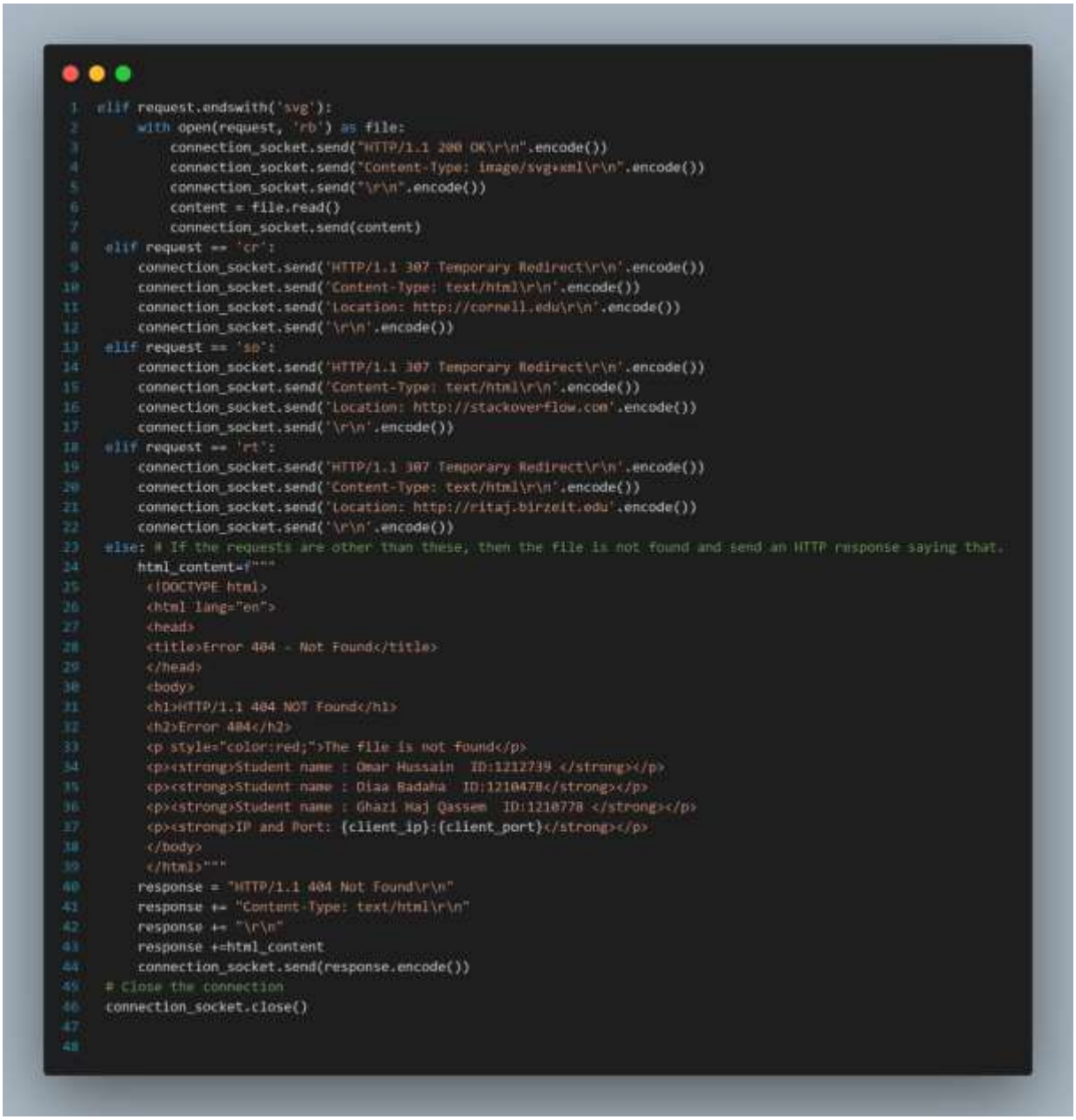
```python
from socket import *
import ctypes
import time

lockWorkStation = ctypes.windll.user32.LockWorkStation # The method that locks the OS imported from ctypes library

server_port = 9955

server_socket = socket(AF_INET, SOCK_STREAM) # The server socket that receives the TCP connection requests from the client: AF_INET is for IPv4, and SOCK_STREAM for TCP
server_socket.bind(('', server_port)) # Bind the server socket with this port
server_socket.listen(1) # Listen for at most one client request
print('The server is ready to receive TCP connection requests.')

while True:
    connection_socket, client_addr = server_socket.accept() # Accept the TCP connection request issued by the client, and create a new connection socket
    client_message = connection_socket.recv(1024).decode() # Receive the message from the client and decode it

    # Check for ID validity
    if client_message == '1210778' or client_message == '1210478' or client_message == '1212778':
        server_message = 'The OS will lock down after 10 seconds'
    else:
        server_message = 'Invalid ID'
    # Print the message on the server side and send it to the client
    print(server_message)
    connection_socket.send(server_message.encode())
    # Close the connection socket
    connection_socket.close()
    # Lock the OS after 10 sec
    if client_message == '1210778' or client_message == '1210478' or client_message == '1212778':
        time.sleep(10) # Sleep for 10 seconds
        lockWorkStation() # Lock the OS
```

Figure 24: Server code



```python
from socket import *

# Set the port number for the server to listen on
server_port = 9966

server_socket = socket(AF_INET, SOCK_STREAM) # Create a TCP socket
server_socket.bind(('', server_port)) # Bind the server
server_socket.listen(1) # Listen for incoming connections with a maximum queue size of 1
print('The server is ready to listen for requests.')

while True:
    connection_socket, client_addr = server_socket.accept() # Get the connection
    client_ip, client_port = client_addr # Get the IP address & the Port of the client
    sentence = connection_socket.recv(1024).decode() # receive the message and decode it
    print(sentence) # Print the message to the terminal window
    list_of_words = sentence.split(' ') # Split the sentence into words

    request = list_of_words[1][1:]  # extract the name of the request file from the HTTP request
    # Check what is the type of the request and send the appropriate file based on the type
    if  request == 'en' or request == 'index.html':
        with open('main_en.html','rb') as file:
            connection_socket.send("HTTP/1.1 200 OK\r\n".encode())
            connection_socket.send("Content-Type: text/html\r\n".encode())
            connection_socket.send("\r\n".encode())
            content = file.read()
            connection_socket.send(content)
    elif request == 'ar':
        with open('main_ar.html','rb') as file:
            connection_socket.send("HTTP/1.1 200 OK\r\n".encode())
            connection_socket.send("Content-Type: text/html\r\n".encode())
            connection_socket.send("\r\n".encode())
            content = file.read()
            connection_socket.send(content)
    elif request.endswith('.html'):
        with open(request,'rb') as file:
            connection_socket.send("HTTP/1.1 200 OK\r\n".encode())
            connection_socket.send("Content-Type: text/html\r\n".encode())
            connection_socket.send("\r\n".encode())
            content = file.read()
            connection_socket.send(content)
    elif request.endswith('.css'):
        with open(request,'rb') as file:
            connection_socket.send("HTTP/1.1 200 OK\r\n".encode())
            connection_socket.send("Content-Type: text/css\r\n".encode())
            connection_socket.send("\r\n".encode())
            content = file.read()
            connection_socket.send(content)
    elif request.endswith('.png'):
        with open(request,'rb') as file:
            connection_socket.send("HTTP/1.1 200 OK\r\n".encode())
            connection_socket.send("Content-Type: image/png\r\n".encode())
            connection_socket.send("\r\n".encode())
            content = file.read()
            connection_socket.send(content)
    elif request.endswith('.jpg'):
        with open(request, 'rb') as file:
            connection_socket.send("HTTP/1.1 200 OK\r\n".encode())
            connection_socket.send("Content-Type: image/jpg\r\n".encode())
            connection_socket.send("\r\n".encode())
            content = file.read()
            connection_socket.send(content)
```

Figure 25: Webserver code 1



```
elif request.endswith('svg'):
    with open(request, 'rb') as file:
        connection_socket.send("HTTP/1.1 200 OK\r\n".encode())
        connection_socket.send("Content-Type: image/svg+xml\r\n".encode())
        connection_socket.send("\r\n".encode())
        content = file.read()
        connection_socket.send(content)
elif request == 'cr':
    connection_socket.send('HTTP/1.1 307 Temporary Redirect\r\n'.encode())
    connection_socket.send('Content-Type: text/html\r\n'.encode())
    connection_socket.send('Location: http://cornell.edu\r\n'.encode())
    connection_socket.send('\r\n'.encode())
elif request == 'sp':
    connection_socket.send('HTTP/1.1 307 Temporary Redirect\r\n'.encode())
    connection_socket.send('Content-Type: text/html\r\n'.encode())
    connection_socket.send('Location: http://stackoverflow.com'.encode())
    connection_socket.send('\r\n'.encode())
elif request == 'rt':
    connection_socket.send('HTTP/1.1 307 Temporary Redirect\r\n'.encode())
    connection_socket.send('Content-Type: text/html\r\n'.encode())
    connection_socket.send('Location: http://ritaj.birzeit.edu'.encode())
    connection_socket.send('\r\n'.encode())
else: # If the requests are other than these, then the file is not found and send an HTTP response saying that.
    html_content=f"""
     <!DOCTYPE html>
     <html lang="en">
     <head>
     <title>Error 404 - Not Found</title>
     </head>
     <body>
     <h1>HTTP/1.1 404 NOT Found</h1>
     <h2>Error 404</h2>
     <p style="color:red;">The file is not found</p>
     <p><strong>Student name : Omar Hussain  ID:1212739 </strong></p>
     <p><strong>Student name : Diaa Badaha  ID:1210478</strong></p>
     <p><strong>Student name : Ghazi Haj Qassem  ID:1210778 </strong></p>
     <p><strong>IP and Port: {client_ip}:{client_port}</strong></p>
     </body>
     </html>"""
    response = "HTTP/1.1 404 Not Found\r\n"
    response += "Content-Type: text/html\r\n"
    response += "\r\n"
    response +=html_content
    connection_socket.send(response.encode())
# Close the connection
connection_socket.close()
```

Figure 26: Webserver code 2