CN REPORT ASSIGNMENT - 2

Ketan Mohan Garg (2022248)
Keshav Bindlish (2022246)

Part - 1

INTRODUCTION

In this project, we created a rudimentary web server capable of handling one HTTP request at a time. The server accepts and parses HTTP requests, obtains the requested files from its file system, and returns the desired material to the client via HTTP response. If the file cannot be located, the server will return the "404 Not Found" message. The server listens on a certain port and may be checked by requesting an HTML file via a browser, which will be shown if successfully served.

ASSUMPTIONS

Single Request Handling: The server is designed to handle one HTTP request at a time, meaning it is not multi-threaded or concurrent.
File Types: The server is assumed to serve static files such as HTML, CSS, and images. Dynamic content handling, such as server-side scripting, is not included.
Valid HTTP Requests: The client sends valid HTTP requests formatted according to HTTP/1.1 standards. The server assumes the request contains the necessary method (e.g., GET) and path.

PROPER CODE WITH THE SNIPPETS EXPLAINED

```
#http://localhost:5500/HelloWorld.html
from socket import *
serverSocket = socket(AF_INET, SOCK_STREAM)
serverSocket.bind(('127.0.0.1', 5500))
serverSocket.listen(1)
while True:
    print("Server is running")
    connectionSocket, addr = serverSocket.accept()
    try:
        message = connectionSocket.recv(1024)
        filename = message.split()[1].decode('utf-8').strip("/")
        print(filename)
        f = open(filename)
        outputdata = f.read()
        f.close()
        connectionSocket.send('HTTP/1.0 200 OK\r\n\r\n'.encode())
        for i in range(0, len(outputdata)):
            connectionSocket.send(outputdata[i].encode())
        connectionSocket.close()
    except IOError:
        connectionSocket.send('404 Not Found'.encode())
        connectionSocket.close()
serverSocket.close()
sys.close()
```

SNIPPET OF CODE WHICH WE HAD TO FILL ARE EXPLAINED BELOW-

serverSocket.listen(1): Configures the server to listen for incoming connections, enabling a single client to be queued.

connectionSocket, addr = serverSocket.accept(): Accepts an incoming client connection and generates a new socket (connectionSocket) using the client's address (addr).

message = connectionSocket.recv(1024): Receives the client's HTTP request, which can include up to 1024 bytes of data.

outputdata = f.read(): Reads the file contents into the output data variable. f.close() closes the file once it has been read***.***

connectionSocket.send('HTTP/1.0 200 OK\r\n\r\n'.encode()): Returns an HTTP 200 OK response header to the client, indicating that the file was found and will be provided.

connectionSocket.send('404 Not Found'.encode()): If the file does not exist, the client will get a "404 Not Found" response.

connectionSocket.close(): In the event of an error, this method terminates the client connection.

PART - 2

INTRODUCTION

In this section, we changed the server from Question 1 so that it could process many HTTP requests simultaneously. The server uses multithreading to generate a main thread that listens for client connections on a defined port. When it receives a TCP connection request, it creates a second thread to handle the client's request and answer via a separate TCP connection. Each request-response pair is processed in its own thread, allowing the server to execute several requests simultaneously.

ASSUMPTIONS

Port Number: 6786

CODE AND ITS EXPLANATION

```
def action_on_client(connectionSocket):
    try:
        message = connectionSocket.recv(1024)
        filename = message.split()[1].decode('utf-8').strip("/")
        print(f"file to be opened: {filename}")
        f = open(filename)
        outputdata = f.read()
        f.close()
        connectionSocket.send('HTTP/1.0 200 OK\r\n\r\n'.encode())
        connectionSocket.send(outputdata.encode())
    except IOError:
```

```
        connectionSocket.send('HTTP/1.0 404 Not Found\r\n\r\n'.encode())
    connectionSocket.close()
```

def action_on_client: This method receives a message from the client or browser and checks to see if the requested file exists in the server's directory. If the file is located, it is shown to the client. However, if the file does not exist, the function provides an error message stating "404 Not Found" to indicate that the requested resource could not be found.

```
def begin_servo(port):
    serverSocket = socket(AF_INET, SOCK_STREAM)
    serverSocket.bind(('localhost', port))
    serverSocket.listen(5)
    print("Ready to serve..,")
    while True:
        connectionSocket, addr = serverSocket.accept()
        print(f"{addr} has connected with the server")
        client_thread = threading.Thread(target=action_on_client,
args=(connectionSocket,))
        client_thread.start()
```

The ***begin_servo*** function creates a TCP connection and listens for incoming requests from clients or browsers. A while loop is used to continue accepting each incoming request. Each request generates a distinct thread, allowing the server to process several requests at once. Each thread performs a distinct function that processes and replies to the client's request separately.

PART - 3

INTRODUCTION

In this section, we created an HTTP client that connects to the server over TCP protocol. The client performs a GET request to the server and shows its response. The client receives command line inputs that include the server's IP address or hostname, the port on which the server listens, and the path to the requested file on the server. This client functions as a web server testing tool, supporting both single-request processing and more complex implementations..

COMMAND TO RUN CLIENT:

client.py server_host server_port filename

-> python3 client.py localhost 6786 HelloWorld.html

PROPER CODE OF CLIENT WITH EXPLANATION

```
#python3 client.py localhost 6786 HelloWorld.html
import socket
server_name = 'localhost'
server_port = 6786
socket_of_client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
socket_of_client.connect((server_name, server_port))
messages = "GET /HelloWorld.html HTTP/1.1\r\nHost: localhost\r\n\r\n"
messages= messages.encode()
socket_of_client.send(messages)
repo= b""
while(True):
    output = socket_of_client.recv(4096)
    if  not output  :
        break
    repo+=output
print(repo.decode())
socket_of_client.close()
```

socket_of_client = Socket.socket(socket.AF_INET, socket.SOCK_STREAM) establishes a TCP connection between the client and server.
socket_of_client.connect((server_name, server_port): Connects the client to the server at the specified IP address (server_name) and port (server_port).
Messages = "GET /HelloWorld.html HTTP/1.1\r\nHost: localhost\r\n\r\n": Constructs the HTTP GET request, asking the server to send the HelloWorld.html file.
socket_of_client.send(request.encode()): Sends the encoded HTTP request to the server.
output= socket_of_client.recv(1024): Receives data from the server (up to 1024 bytes at a time).
print(repo.decode()): Decodes the received data and prints it, displaying the server's response.
socket_of_client.close(): Closes the socket, terminating the connection after receiving the server's response.

THE IMPLEMENTED CLIENT WORKS WITH SERVER IMPLEMENTED IN BOTH PART 1 AND 2.
ALSO while loop is used for the client to continuously receive data from the server until no more data is sent.