Programming Assignment 1

Introduction to Socket Programming in Python

Hossam Osama Ahmed	21010451
Ali Hassan Ali	21010837

1. Introduction

The purpose of this assignment is to provide a practical understanding of HTTP communication and socket programming. HTTP, originally developed by Berners-Lee and his team, was designed to facilitate the transfer of web pages across the internet. In this assignment, we will develop a simplified version of the HTTP protocol to handle basic GET and POST requests, while utilizing the socket programming features in either Java or Python. This hands-on exercise focuses on using UNIX sockets to implement both a server and client for HTTP communication, simulating a real-world web service.

---

2. Part 1: Multi-threaded Web Server

2.1 Introduction and Background

The web server will be designed to accept client connections and process both GET and POST HTTP requests. For GET requests, it will send the requested file if it exists, or return a 404 error if the file is not found. For POST requests, the server will receive the file and store it on the server. Additionally, the server will support persistent connections, allowing multiple requests from the same client without needing to reopen a connection.

2.2 Specifications

The server must:

- Accept incoming client connections.
- Handle GET requests to serve files (HTML, TXT, images).
- Handle POST requests to receive files from the client.
- Respond with the appropriate HTTP status codes (200 OK for success, 404 Not Found for missing files).
- Keep the connection open for future requests or close it if the connection times out.

- Handle multiple clients using threading to improve performance.

2.3 Server-Side Pseudo Code

The server follows these steps:

python

Copy code

while true:

  listen for connections

  accept a new connection

  create a worker thread to handle the client

  parse HTTP request (GET or POST)

  check if the requested file exists (for GET)

  send file content or 404 error (for GET)

  save file content (for POST)

  keep connection open for more requests, or close if idle

end

2.4 Server Implementation

The provided server code is designed to:

- Use TCP sockets to listen for client connections.
- Create a new thread for each incoming client to handle requests concurrently.
- Process GET and POST requests as per the specifications, using appropriate HTTP responses.

Key Functions:

- handle_client(): Manages client communication, parses HTTP requests, and sends responses.
- start_server(): Initializes the server, listens for incoming connections, and delegates them to worker threads.

```
PS D:\projects year 3\semester 1\Networks\Programming Assignment 1>  & 'c:\Users\Hossam\anaconda3\python.exe' 'c:\Users\Hossam\.vscode\exte
nsions\ms-python.debugpy-2024.12.0-win32-x64\bundled\libs\debugpy\adapter/../..\debugpy\launcher' '58630' '--' 'd:\projects year 3\semester
 1\Networks\Programming Assignment 1\my_server.py'
Server running on 127.0.0.1:8080
Connected by ('127.0.0.1', 58679)
Connection with ('127.0.0.1', 58679) closed.
Connected by ('127.0.0.1', 58699)
Connection with ('127.0.0.1', 58699) timed out.
Connection with ('127.0.0.1', 58699) closed.
```

(This screenshot showing normal close and timeout close)

python

```
import socket

import threading

import os

HOST = '127.0.0.1'
PORT = 8080
BASE_DIR = './'

def handle_client(conn, addr):
    # Handles client requests (GET or POST)
    pass

def start_server():
    # Sets up the server to listen for client connections
    pass

if __name__ == "__main__":
    start_server()
```

---

3. Part 2: HTTP Web Client

3.1 Specifications

The client will:

- Connect to the server using a socket connection.
- Support both GET and POST commands based on input from the user.
- Send a GET request to retrieve a file and save it locally.
- Send a POST request to upload a file to the server.

3.2 Client-Side Pseudo Code

The client follows these steps:
python
connect to server
while more operations exist:
  send GET or POST request to the server
  receive response from the server
close connection

3.3 Client Implementation

The provided client code allows interaction with the server through a command-line interface, where the user can input commands like get <file-path> and post <file-path>.

Key Functions:

- client_connect(): Establishes a connection with the server.
- client_get(): Sends a GET request to the server and saves the received file.
- client_post(): Sends a POST request to upload a file to the server.

```
PS D:\projects year 3\semester 1\Networks\Programming Assignment 1> cd ..
PS D:\projects year 3\semester 1\Networks> python my_client.py 127.0.0.1 8080
Connected to server at 127.0.0.1:8080
Enter command (get <file_path> | post <file_path> | exit): get new.png
Server headers:
 HTTP/1.1 200 OK
Content-Length: 646480
Connection: keep-alive
Saved GET response to new.png
Enter command (get <file_path> | post <file_path> | exit): get example.html
Server headers:
 HTTP/1.1 200 OK
Content-Length: 486
Connection: keep-alive
Saved GET response to example.html
Enter command (get <file_path> | post <file_path> | exit): get upload.txt
Server headers:
 HTTP/1.1 200 OK
Content-Length: 7
Connection: keep-alive
Saved GET response to upload.txt
Enter command (get <file_path> | post <file_path> | exit): post h.png
Server response:
 HTTP/1.1 200 OK
Content-Length: 29181
Connection: keep-alive


Enter command (get <file_path> | post <file_path> | exit): exit
Closing connection.
Connection closed.
PS D:\projects year 3\semester 1\Networks>
```

(the screenshot above showed how to run client and try get and post for different types)

```
python
import socket
import sys

DEFAULT_PORT = 80

def client_connect(host, port=DEFAULT_PORT):
    # Connects to the server
    pass

def client_get(client, file_path, host):
    # Sends a GET request and saves the file
```

```
    pass

def client_post(client, file_path, host):
    # Sends a POST request to upload a file
    pass

if __name__ == "__main__":
    main()
```

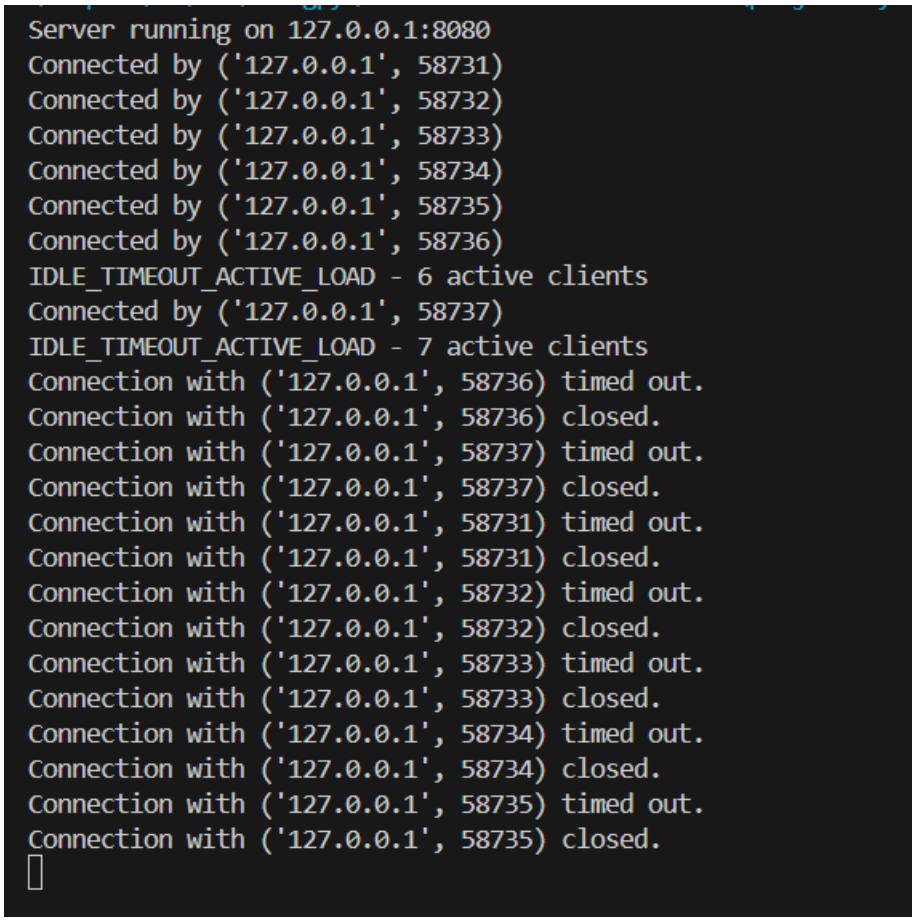
---

4. Part 3: HTTP/1.1 Support

- Support persistent connections, allowing multiple requests to be handled on the same connection.
- Implement a connection timeout strategy that adapts based on the number of active clients.

4.1 Persistent Connections

- The server will leave the connection open for a set time after a request, allowing for subsequent requests to use the same connection (pipelining).
- The timeout period will vary based on server load. When there are many active clients, the server will shorten the timeout to conserve resources.



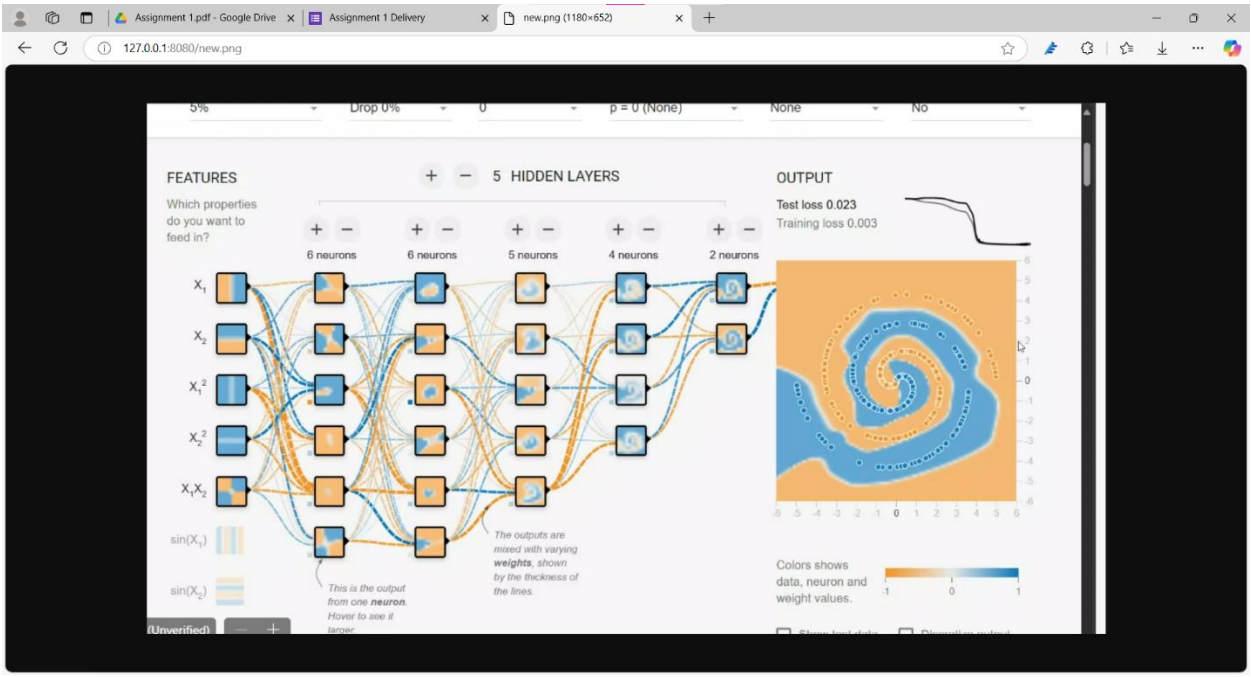
(this screenshot shows change of timeout when many clients connect with the server)

---

5. Bonus: Performance Evaluation and Browser Testing

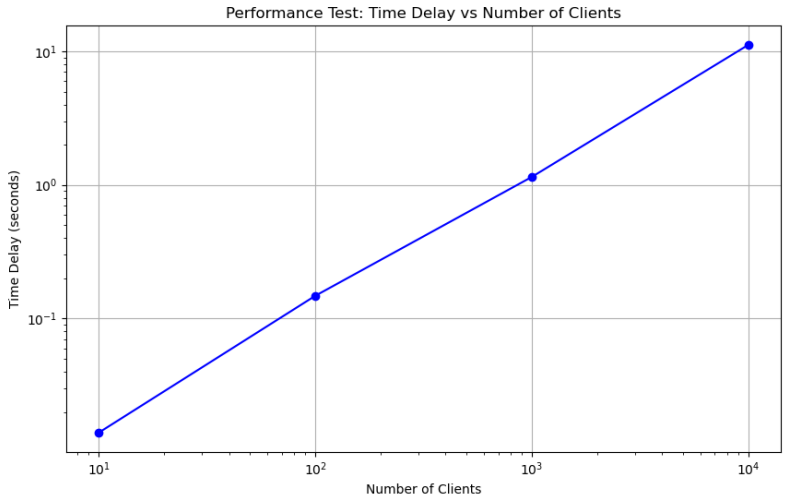
5.1 Test with a Real Web Browser

Test the server with an actual web browser (e.g., Chrome, Firefox) by accessing the server using 127.0.0.1/8080/(what you want but near server file).



5.2 Performance Evaluation

- Explore the server's performance under different loads (more clients and requests).




```
Starting performance test with 10 clients...
Test completed for 10 clients. Total time: 0.0140 seconds.
Starting performance test with 100 clients...
Test completed for 100 clients. Total time: 0.1480 seconds.
Starting performance test with 1000 clients...
Test completed for 1000 clients. Total time: 1.1476 seconds.
Starting performance test with 10000 clients...
Test completed for 10000 clients. Total time: 11.1699 seconds.
PS D:\projects year 3\semester 1\Networks\Programming Assignment 1>
```