Programming Assignment 1

Computer Networks

10 NOV 2024

Saifullah Mousaad

Ahmed Ashraf

1 Multi-threaded Web Server

1.1 Purpose of the Server

The server is a simple HTTP server designed to handle both `GET` and `POST` requests. It serves static files stored in a directory called root and logs information about each request, including details such as the client's IP address, port, method, and connection type.

1.2 Features and Capabilities

1. **HTTP/1.1 Compliant:** The server is designed to handle basic HTTP/1.1 requests and responses, supporting common methods such as `GET` and `POST`.
2. **Multi-Threaded**: It uses threading to handle multiple client connections concurrently, allowing multiple requests to be processed at the same time.
3. **Content-Type Identification**: The server can identify and set the correct MIME type based on file extensions, supporting a range of types, including `text`, `image`, and `application`.
4. **File Handling**:
    a. For `GET` requests, the server retrieves and serves requested files from a specified directory.
    b. For `POST` requests, the server saves the incoming data to a file with a suitable extension based on the `Content-Type` header.
5. **Error Handling**: The server handles various errors, including missing files (`404 Not Found`) and internal server errors (`500 Internal Server Error`), sending appropriate status codes to the client.
6. **Dynamic Timeout Adjustment**: It dynamically adjusts the timeout duration for each client connection based on the number of active connections, providing better resource management under load.

1.3 Components and Functions

1. **`get_content_type(extension)`**: This function determines the MIME type of a requested file based on its extension, returning the appropriate content type string (e.g., `text/html` for `.html` files). It classifies extensions as either text, image, or application types, mapping unsupported types to `application`.
2. **`generate_headers(status_code, content_type, content_length)`**: This function constructs HTTP response headers with details such as status code, date (in GMT), server details, content length, and content type. The headers are formatted as per HTTP protocol and returned as a string to be included in the response.
3. **`prepare_response(status_code, content_type, content, client_socket)`**: This function sends a full HTTP response, including headers and content, to the client. It generates headers, appends the content, and sends the data over the client socket. It handles both text and binary (e.g., image) content types.
4. **`parse_message(msg, client_socket)`**: This function parses an HTTP request by splitting it into header and body sections. It extracts the HTTP method, requested path, content type, and any body content, returning these values to be used by other functions. If the request body contains binary data, it prints a message.
5. **`process_message(msg, client_socket)`**: This function handles the HTTP request message and sends an appropriate response. For `GET` requests, it tries to read the requested file and respond with its content. For other requests (e.g., `POST`), it saves the request body to a file and responds with a success status. It also handles errors, sending a 404 or 500 status code when necessary.
6. **`receive_data(client_socket)`**: This function manages data reception from a connected client. It sets a timeout based on the number of active connections and continuously receives data, processing each message via

`process_message`. If a timeout or connection reset occurs, it adjusts the connection count and ends the session.

7. **`__main__`**: The main function initializes a TCP server socket, binds it to a specified port, and listens for client connections. Each incoming connection is handled in a separate thread to allow concurrent client processing. The server can be safely stopped with a keyboard interrupt.

2 HTTP Web Client

- The Client is developed to handle GET and POST requests with HTML, text and Images.

2.1 Flow of the program:

- Receive an IP address and port number as arguments to the command line
- Establish a connection with the server has the specified IP and port number entered
- Ask for a input file contains command to perform with the server
- Commands are executed back-to-back depending on the method (GET - POST) and the other parameters (File Type,...,etc

2.2 Methods and Functionalities:

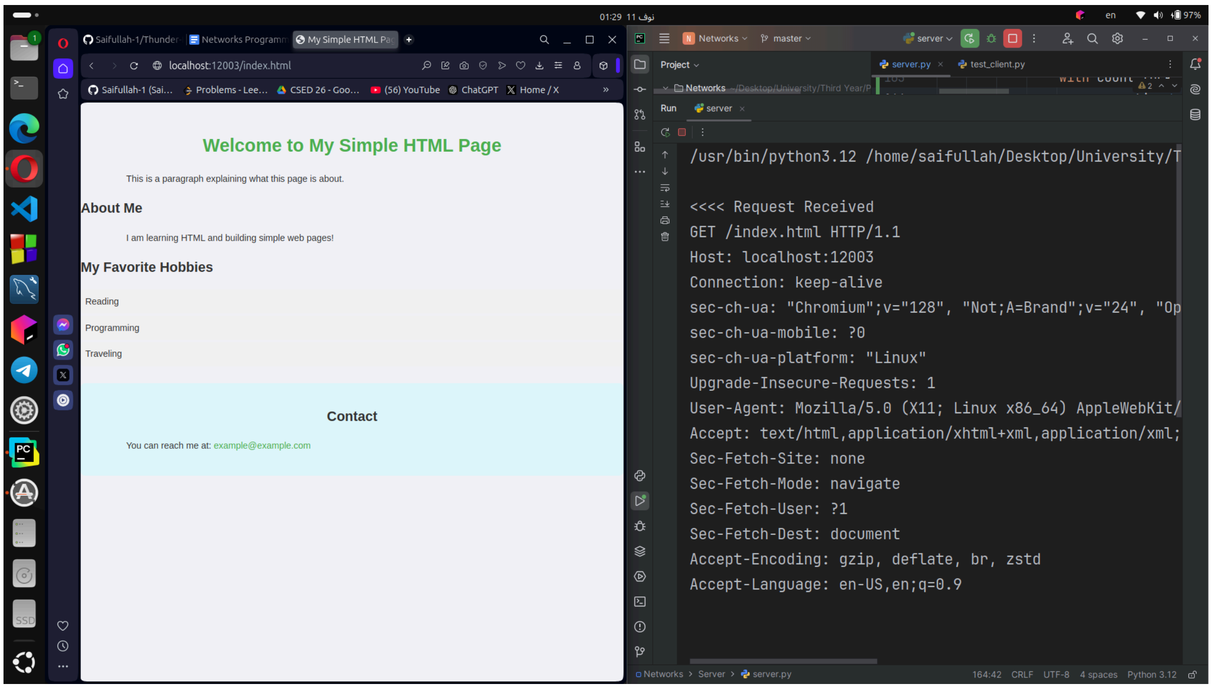
1. **`send_request`**
    a. The executor of each command, take the request as an input
    b. Does the main steps represented in calling the following functions below in an ordered manner
    c. Handles receiving large data and it's open for the user to POST a file with any size he wants, the buffering is done dynamically based on the size of the file.
    d. After receiving responses, based upon the method (GET, POST) the function either only shows the response message in case of POST or displays the response message including the body then creates a file for saving the data received and stores it in depending on the type (Image-text).
2. **`parse_command`**
    a. Handles the splitting of the command line from the input file to retrieve the required information setting the port to 80 by default if the command ignores the port
3. **`file_type`**
    a. This function generates Content_Type header in its standard form to send with the request.
4. **`read_posted_file`**
    a. Takes the path of the file requested to be posted on the server and returns the data within the file and handles the error resulting from the absence of the file mentioned.
5. **`form_request`**
    a. responsible for forming requests based on the method (GET, POST) while putting any necessary header lines such as Content_Type and Content_Length. The requests must follow the standard format of HTTP request.

3 HTTP 1.1

1. The server by default runs persistent connections between clients and servers.
2. As Required in Part 3 of the assignment, Pipelining is considered as each client gets one thread to serve him alone, hence the server can serve many clients at the same time.
3. A timeout consideration is applied also, i.e each client has a limited amount of time for inactivity before the connection will be lost (aborted by the server). The Heuristic function is a constant number, in our case 20, divided by the number of current clients using the server at this time. Of course, the number of concurrent clients changes with each new client so clients get their allowed timeout dynamically.
4. Heuristic Function = $\frac{20}{Number\ of\ Currently\ Active\ Clients}$
5. The timeout duration of the server itself is put to None using "**server_socket.settimeout(None)**" to ensure that the server is always on.
6. For the clients, *settimeout* method is put before each client gets his service with the existence of except "**TimeoutError**" to catch the client when its connection is closed by the server to handle what happens after, which is terminating his program.

4 Bonus

3.1 Test With Browser



Welcome to My Simple HTML Page

This is a paragraph explaining what this page is about.

About Me

I am learning HTML and building simple web pages!

My Favorite Hobbies

Reading

Programming

Traveling

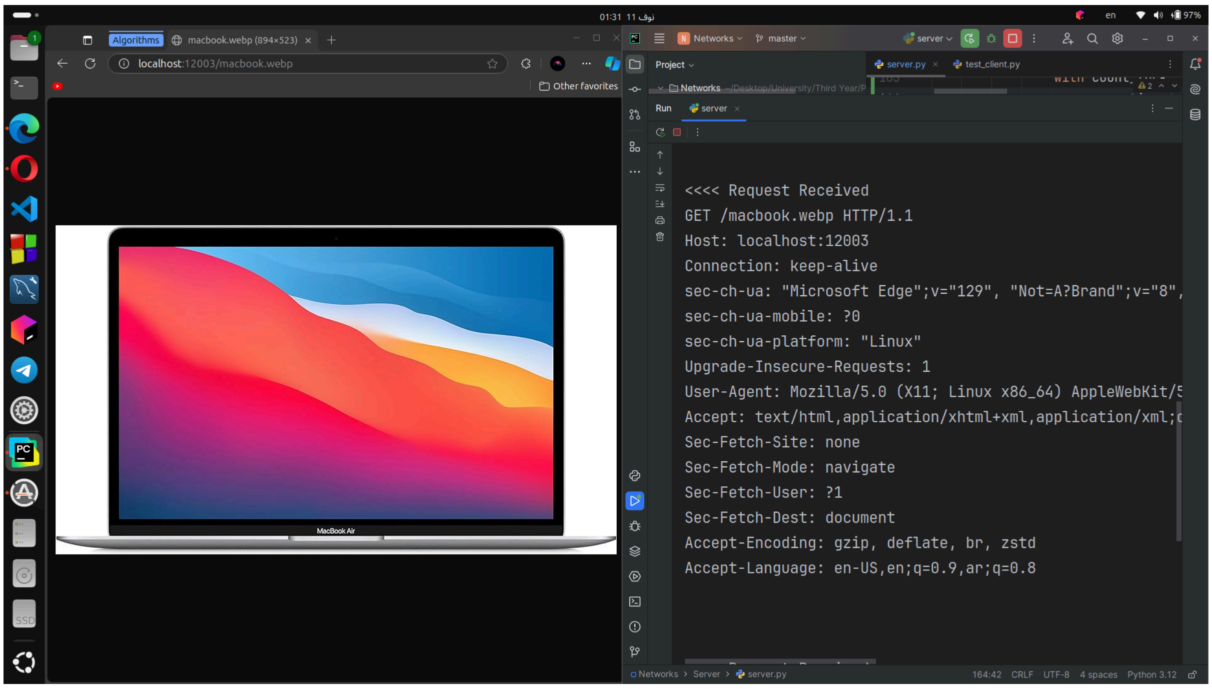
Contact

You can reach me at: example@example.com

```
/usr/bin/python3.12 /home/saifullah/Desktop/University/T

<<<< Request Received
GET /index.html HTTP/1.1
Host: localhost:12003
Connection: keep-alive
sec-ch-ua: "Chromium";v="128", "Not;A=Brand";v="24", "Op
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Linux"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/
Accept: text/html,application/xhtml+xml,application/xml;
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9
```

Figure 1: Test with Opera Browser



```
<<<< Request Received
GET /macbook.webp HTTP/1.1
Host: localhost:12003
Connection: keep-alive
sec-ch-ua: "Microsoft Edge";v="129", "Not=A?Brand";v="8",
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Linux"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/5
Accept: text/html,application/xhtml+xml,application/xml;
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9,ar;q=0.8
```

Figure 2: Test with Microsoft Edge Browser

3.2 Performance Evaluation

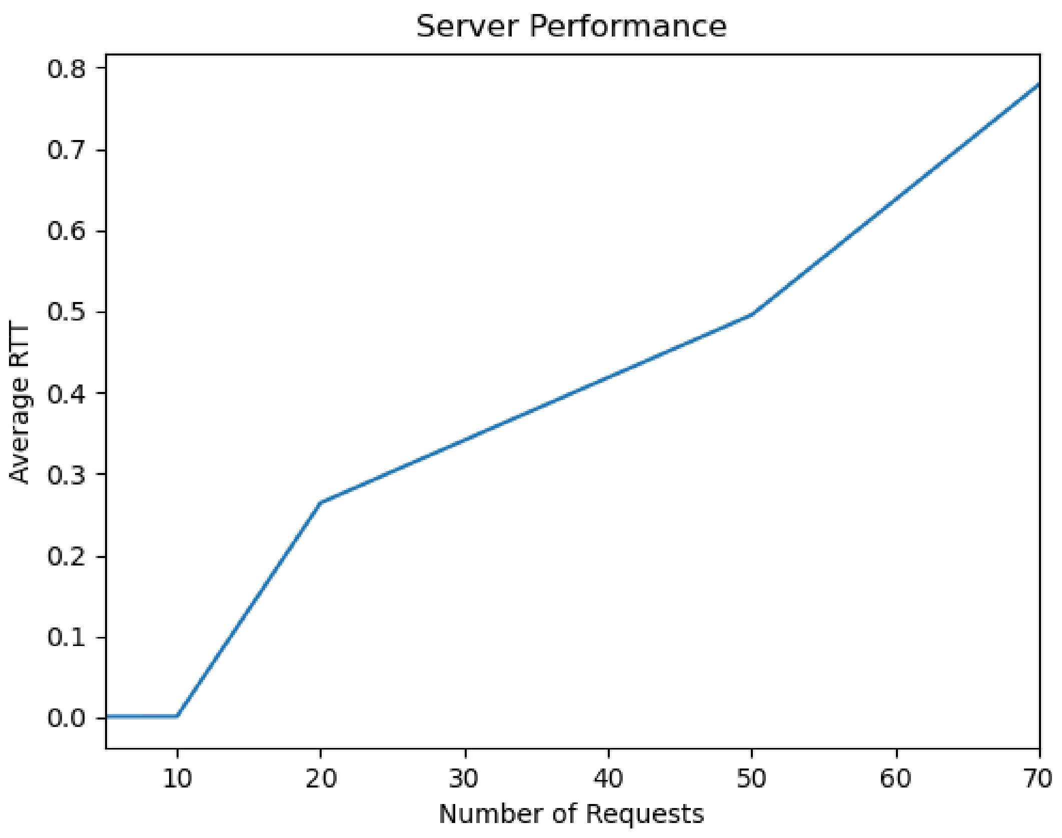


Figure 1: Performance Evaluation

5 Source Code

https://github.com/Saifullah-1/HTTP-Client-Ser