Programming Exercise 2

Mehraz Abedin Raz (2022293)
Nakul Garg (2022309)

Introduction

In this assignment, we made a TCP-based web application in Python. We used Socket programming and HTTP protocol. The system was divided into three parts:

- Simple web server that can accept and process one HTTP request at a time.

- A multi-threaded web server that handles multiple requests simultaneously.

- A tailored HTTP client that communicates with the server over TCP and retrieves files.

We have employed Python for both the side of client and server programming, and illustrated the basic concepts of socket creation, connection management, formatting HTTP messages, and multithreading.

Assumptions

- The server is assumed to run locally (localhost).
- Files to be served (like `helloworld.html`) are available in the server's directory.
- The server is listening on port `10403` (can be changed).
- The client can specify the server address, port, and filename via command-line arguments.
- If the requested file is missing, the server returns a 404 response.

Part A: Single-Request Web Server

We first developed a single-threaded web server that processes one HTTP request at a time.

key functions include:

- Listening for incoming connections on a specified port.
- Parsing the incoming HTTP request.
- Retrieving the requested file from the server's directory.
- Sending the file contents to the client, preceded by appropriate HTTP headers.

Core Concepts Covered:

- TCP connection establishment.
- HTTP request parsing.
- HTTP response construction with status codes (200 for success, 404 for file not found).

Code Explanation:

- We use Python's `socket` module to create the server socket.
- The server accepts connections using `accept()` and receives HTTP requests using `recv()`.
- Based on the request, the server opens and reads the requested file. If the file is not found, it redirects to a custom 404 error page.

Part B: Single-Request Web Server

The single-threaded server handles one request at a time. To handle multiple client requests simultaneously, we introduced threading. When a request is received, the server spawns a new thread to handle that particular connection. This approach allows multiple clients to request files concurrently.

Key Changes

- The server uses Python's `threading` module to create separate threads for each client request.
- Each thread runs the logic to process an individual request, ensuring that multiple clients are served at the same time.

Part C: Custom HTTP Client

To test the server without relying on a web browser, we created a custom HTTP client.

key functions include:

- Connects to the server using TCP.
- Sends an HTTP GET request for a specific file.
- Displays the server's response (either the html file contents or the 404 error html contents).

Command Line Format:
bash
python client.py <server_host> <server_port> <filename>

Example Usage:
bash
python client.py localhost 10403 helloworld.html

Testing and Results

To verify the implementation, we tested the following scenarios:

- Successful file retrieval (e.g., `helloworld.html`).
- File not found scenario (404 error).
- Multiple client connections handled simultaneously by the multi-threaded server.
- Variable Responses on custom client.