COMP2322 WebServer Project Report

LIU Siyuan*
Department of Computing
The Hong Kong Polytechnic University
23101026d@connect.polyu.hk

1 Introduction

This project delivers a robust multi-threaded Web server capable of handling concurrent HTTP client requests efficiently. The server is well designed and debugged to ensure reliable performance, supporting core HTTP functionalities such as file retrieval and error handling. A key innovation lies in the development of **client.py**, a custom testing tool that simplifies server interaction, enabling users to send requests and validate responses with ease. This tool streamlines testing workflows and enhances usability during development. The server's codebase emphasizes clean architecture and scalability, adhering to foundational networking principles. All source code is open-sourced on GitHub, fostering transparency and collaboration for future improvements. This project demonstrates a practical implementation of multi-threaded socket programming while prioritizing user-friendly testing and accessibility.

2 Design and Implementation

2.1 Overall Design

The server adopts a modular and multi-threaded architecture to ensure clear structure and robust performance. Request parsing, response generation, connection management, and logging are implemented as independent functions to improve readability and maintainability. Python's built-in `threading` module is used to spawn a dedicated thread for each incoming client connection, allowing the server to handle multiple concurrent requests without blocking.

The server supports both GET and HEAD methods for HTTP/1.0 and HTTP/1.1 clients. It handles persistent connections according to HTTP/1.1 standards, where connections remain open unless the client explicitly includes `Connection: close`. For HTTP/1.0 clients, the connection is closed by default unless `Connection: keep-alive` is specified. This behavior ensures compliance with the HTTP specification and improves efficiency when handling multiple requests from the same client.

To ensure security and sandboxing, the server strictly confines all file access to the `www/` directory. All incoming paths are normalized and verified to prevent directory traversal attacks. If a requested path attempts to access files outside `www/`, the server responds with a 403 Forbidden error. Supported files are served based on MIME type mappings, and files with unknown or unsupported extensions are rejected with a 415 Unsupported Media Type response.

A custom `client.py` script is provided to facilitate functional and concurrency testing. In addition to basic requests, it supports batch testing to verify different HTTP statuses, and a concurrent testing mode that simulates multiple clients accessing the server simultaneously. This verifies the server's multi-threaded behavior and helps confirm thread-safety in both file serving and logging.

In terms of web interface, the homepage `index.html` is located in the `www/` folder and contains a navigation link labeled *Chart 1*. Clicking this link navigates to `ink_vis_r123_bt123.html`. This

*Project Repository: `https://github.com/siyuan0000/COMP2322_WebServerProject/tree/main`

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

structure demonstrates the server's ability to handle HTML navigation, static file requests, and image content delivery.

All requests are recorded in `server.log`, which includes the timestamp, client IP and port, the exact HTTP request line, and the response status. To ensure correctness under concurrency, a thread-safe logging mechanism is implemented using mutual exclusion via Python's `threading.Lock`. This design supports both functional correctness and ease of demonstration during testing.

2.2 Implementation Details

The server is written in Python with a clean separation of concerns. Core tasks—request parsing, response construction, logging, and per-client handling—are independent functions. 'parse_request' validates the method (GET / HEAD), HTTP version, headers, and prevents directory-traversal by normalising paths inside the `www/` sandbox. 'build_response' maps the request to a file, checks MIME type, sets `Last-Modified`, honours `If-Modified-Since`, and generates all required status codes (200, 304, 400, 403, 404, 415).

To support concurrency, the main loop only accepts sockets and spawns a new `threading.Thread` per connection. The dedicated `handle_client` function manages persistent (`keep-alive`) and non-persistent connections, re-using the same socket for multiple requests until either side closes. All network I/O is blocking but isolated within each thread, so simultaneous clients do not interfere with one another.

Logging is centralised in `log_request`. A global `threading.Lock` guards writes to `server.log`, ensuring entries remain atomic and ordered even under heavy load. Each line records the timestamp, client address, request line, and response status, providing a reliable trace for debugging and assessment.

3 Demonstration

Table 1: Testing plan for multi-threaded HTTP server

Test Case	**Request Method and Path**	**Expected Response**	**Test Command**
Normal file retrieval	GET /index.html	200 OK (HTML file returned)	`python client.py GET /index.html`
HEAD request retrieval	HEAD /index.html	200 OK (headers only, no body)	`python client.py HEAD /index.html`
Conditional GET (cache hit)	GET /index.html with If-Modified-Since	304 Not Modified (no body)	`python client.py batch` (automatic)
File not found error	GET /nofile.html	404 Not Found	`python client.py GET /nofile.html`
Forbidden directory traversal	GET /../server.log	403 Forbidden	`python client.py GET /../server.log`
Unsupported media type	GET /unsupported.xyz	415 Unsupported Media Type	`python client.py GET /unsupported.xyz`
Malformed method request	POST /index.html	400 Bad Request	`python client.py POST /index.html`
Concurrent clients test	Multiple GET /index.html from different clients	All clients receive 200 OK (multi-threading validated)	`python client.py concurrent 10`

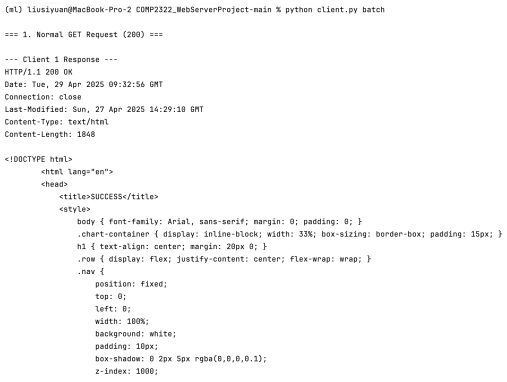

Figure 1: batch command 1

Figure 2: batch command 2&3

Figure 3: batch command 4&5

Figure 4: batch command 6&7

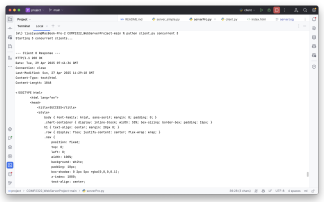

Figure 5: concurrent command

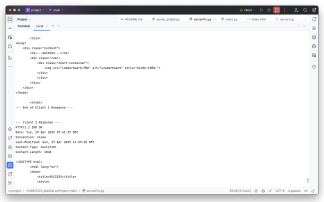

Figure 6: concurrent command

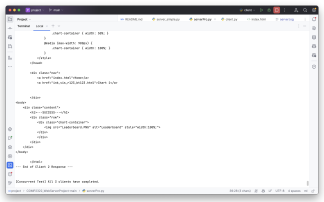

Figure 7: concurrent command

3.1 Log File

The server is designed to record all incoming HTTP requests into a log file named `server.log`. Each log entry includes the timestamp, the client's IP address and port number, the original HTTP request line, and the server's response status code along with its reason phrase. A dedicated function `log_request` handles the logging operation by appending each entry to the file.To ensure correct behavior under multi-threaded connections, a simple locking mechanism is used when writing to the log. This design helps maintain a clear and reliable request history, which is useful for server monitoring, debugging, and result demonstration.

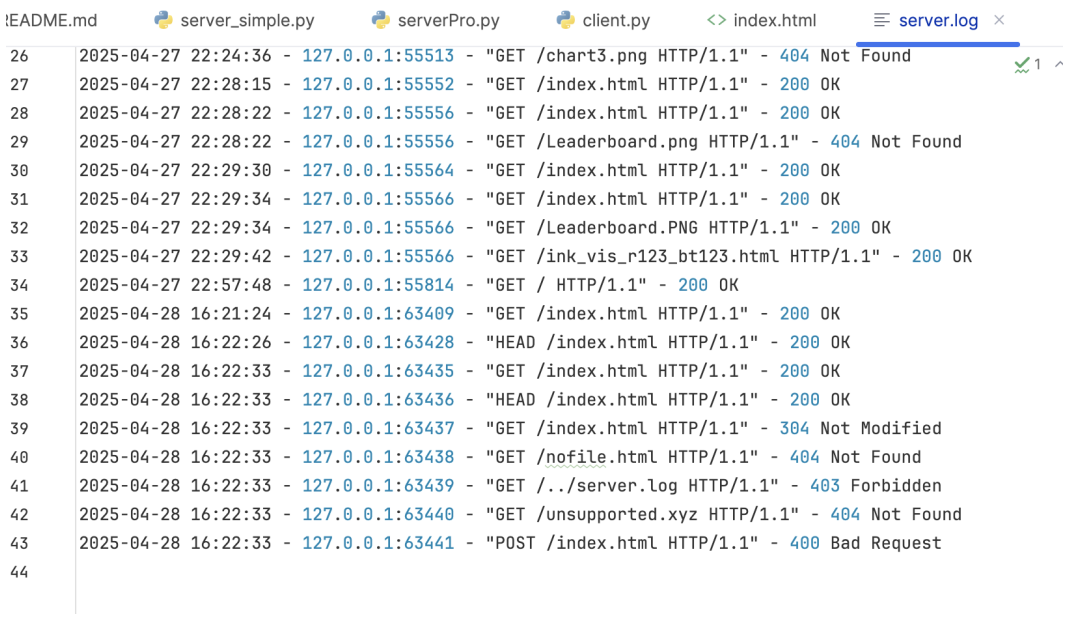

Figure 8: log file