CSE3027 Project 1 Report

Student Name : 이민성

Student ID : 2022006971

Course : CSE3027: Computer Networks

Project Title : Concurrent Web Server using BSD Sockets

Due Date : April 27, 2025

1. High-Level Description of Server Design

The goal of this project was to implement a concurrent web server in C using BSD sockets. The server accepts HTTP requests from web browsers and serves static files (HTML, images, audio, PDF) in response. The server was designed in two parts

- Part A : Display incoming HTTP request messages on the server console.

- Part B : Parse HTTP GET requests, identify the requested resource, and respond with the correct HTTP headers and file content.

The server creates a TCP socket, binds it to the specified port, and listens for incoming connections. For each connection, the server forks a new process to handle the request concurrently, allowing multiple clients to connect simultaneously.

The server reads the request message, extracts the requested file name, and attempts to locate and read the file. If the file exists, the server sends a 200 OK response with the appropriate Content-Type header and the file contents. If the file does not exist, the server responds with a 404 Not Found message.

Supported content types include HTML, JPEG, GIF, MP3, and PDF.

2. Difficulties Faced and Solutions

- Understanding HTTP Request Format : Initially, understanding how HTTP GET requests are structured was challenging. This was resolved by analyzing real browser requests using server-side logging and RFC 1945 documentation.

- File Handling for Binary Types : Serving binary files like images and PDFs required

careful file I/O handling to ensure no corruption. Using `open()`, `read()`, and `write()` with byte-level control solved this.

- Content-Type Identification : Mapping file extensions to proper MIME types required a custom function. A simple if-else structure was used to match known extensions to their correct Content-Type.

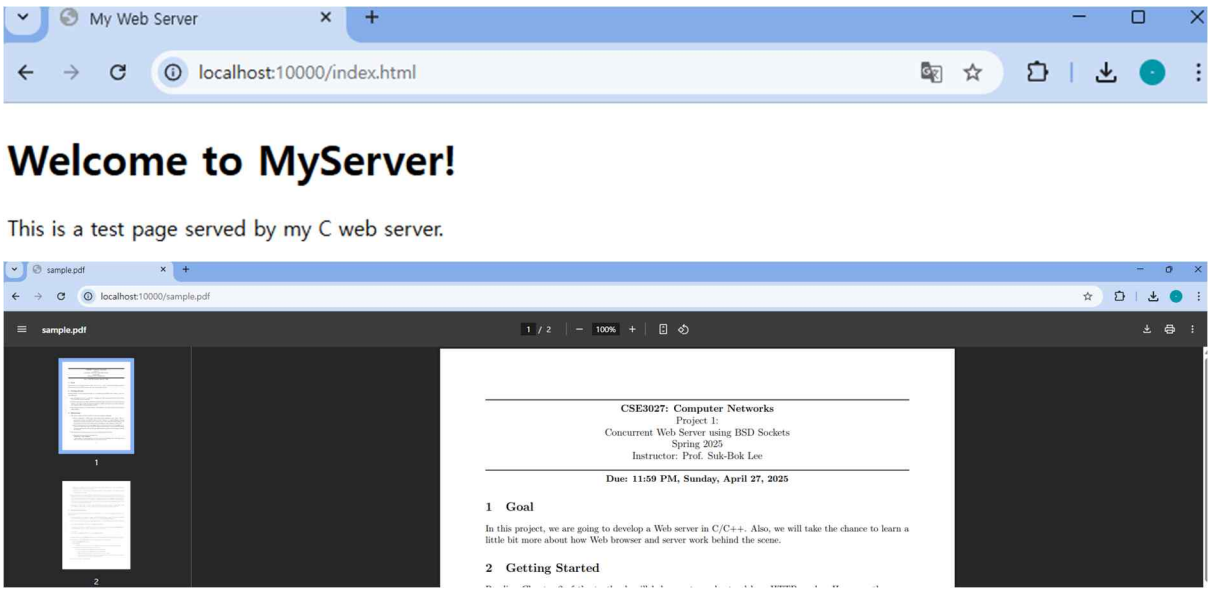
- Concurrent Client Handling : Ensuring the server could handle multiple clients without crashing was handled using `fork()` to spawn child processes. Proper `close()` handling ensured no resource leakage.

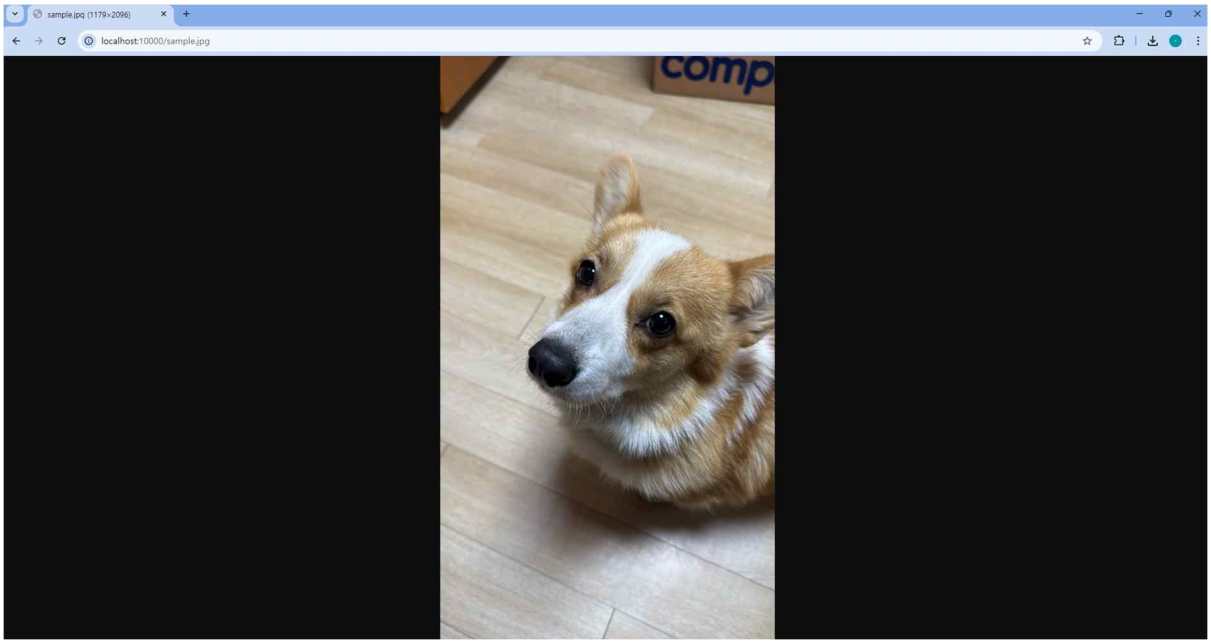
3. Sample Outputs

(1) Console Output in Part A :

```
minsung@Minsung:~/ComputerNetwork/Project1$ ./myserver 10000
HTTP Server running on port 10000...
===== Received HTTP Request =====
GET /index.html HTTP/1.1
Host: localhost:10000
Connection: keep-alive
```

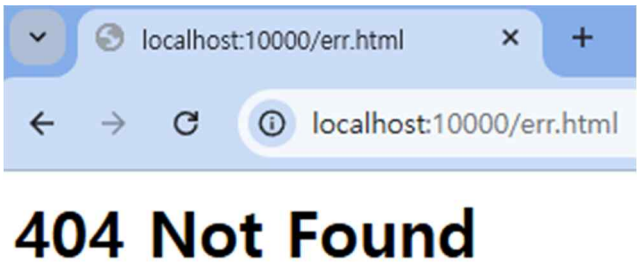
(2) Browser Output in Part B :





(3) 404 Not Found :

When accessing a non-existent file:



404 Not Found

4. Conclusion

This project provided hands-on experience in implementing a basic web server, understanding HTTP communication, and handling concurrent client requests. It also offered practical challenges in C network programming, file I/O, and process management, reinforcing core concepts of computer networks and systems programming.