



## **Internship Program (Batch 2)**

### **Task.#.4**

**Name: Mian Muhammad Awais**

**Section: C++ (Programming)**

# Building a Multi-Threaded Web Server:

## Objective:

- Develop a basic web server that can handle multiple client requests simultaneously.

## Description:

- Create a C++ program that listens for HTTP requests and serves static HTML files. Use multi-threading to handle multiple clients concurrently.

## Key Steps:

- Setting up socket programming to handle HTTP requests and responses
  - Implementing multi-threading using the C++ Standard Library's thread support
  - Serving static HTML files from a specified directory
  - Handling concurrent client connections
-

## † Code with explanation:

### 1. Includes:

```
1  #include <iostream>
2  #include <string>
3  #include <thread>
4  #include <mutex>
5  #include <vector>
6  #include <fstream>
7  #include <sstream>
8  #include <sys/socket.h>
9  #include <netinet/in.h>
10 #include <unistd.h>
11
12 #define PORT 8080
13
14 using namespace std;
```

- **Headers:**

- `<iostream>`: For input/output operations (e.g., `cout`).
- `<string>`: For using `std::string`.
- `<thread>`: For multi-threading support.
- `<mutex>`: For synchronization (though not used explicitly here).
- `<vector>`: Not used in this code but included for completeness.
- `<fstream>`: For file input/output.
- `<sstream>`: For string stream operations.
- `<sys/socket.h>`, `<netinet/in.h>`, `<unistd.h>`: For socket programming (e.g., creating, binding, and listening on sockets).
- **#define PORT 8080**: Defines the port number that the server will listen on (8080 in this case).
- **using namespace std;**: Allows direct access to standard library classes and functions without prefixing them with `std::`.

## 2. Global Mutex:

```
15  
16 mutex mtx;  
17
```

**mutex mtx;** A mutex object to synchronize access to shared resources (though it's not actively used in this simplified code).

## 3. Function to Read a File:

```
18 string read_file(const string &path) {  
19     ifstream file(path);  
20     if (!file.is_open()) {  
21         return "<html><body><h1>404 Not Found</h1></body></html>";  
22     }  
23     stringstream buffer;  
24     buffer << file.rdbuf();  
25     return buffer.str();  
26 }  
27
```

**read\_file:**

- Opens a file specified by path.
- If the file does not open (e.g., it does not exist), returns a "404 Not Found" HTML response.
- If the file opens, reads its content into a stringstream and returns it as a string.

#### 4. Function to Parse HTTP Requests::

```
28 string parse_request(const string &request) {
29     istringstream stream(request);
30     string method, path, version;
31     stream >> method >> path >> version;
32
33     // Remove leading "/" from the path
34     if (path == "/") {
35         path = "/index.html";
36     }
37     return path.substr(1);
38 }
39
```

**parse\_request:**

- Parses the HTTP request to extract the HTTP method, requested path, and version.
- Converts the path to serve index.html if the root path / is requested.
- Returns the path without the leading /.

#### 5. Function to Handle Client Requests:

```
40 void handle_client(int client_socket) {
41     char buffer[1024] = {0};
42     read(client_socket, buffer, 1024);
43
44     string requested_file = parse_request(buffer);
45     cout << "Serving file: " << requested_file << endl;
46
47     string file_content = read_file(requested_file);
48
49     string response = "HTTP/1.1 200 OK\nContent-Type: text/html\n\n" + file_content;
50     send(client_socket, response.c_str(), response.size(), 0);
51
52     close(client_socket);
53     cout << "Connection closed for file: " << requested_file << endl;
54 }
55
```

**handle\_client:**

- Reads data from the client into buffer.
- Parses the request to get the file to serve.
- Prints the file being served to the console.
- Reads the content of the requested file and constructs an HTTP response.

- Sends the response to the client.
- Closes the client socket and prints a message indicating the connection is closed.

## 6. Main Function:

```

56 int main() {
57     int server_fd, new_socket;
58     sockaddr_in address;
59     int addrlen = sizeof(address);
60
61     // Creating socket file descriptor
62     if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
63         perror("Socket failed");
64         exit(EXIT_FAILURE);
65     }
66
67     // Bind the socket to the network address and port
68     address.sin_family = AF_INET;
69     address.sin_addr.s_addr = INADDR_ANY;
70     address.sin_port = htons(PORT);
71
72     if (bind(server_fd, (sockaddr *)&address, sizeof(address)) < 0) {
73         perror("Bind failed");
74         exit(EXIT_FAILURE);
75     }
76
77     // Start listening for connections
78     if (listen(server_fd, 3) < 0) {
79         perror("Listen failed");
80         exit(EXIT_FAILURE);
81     }
82
83     cout << "Server is listening on port " << PORT << endl;
84
85     while (true) {
86         // Accept incoming connections
87         if ((new_socket = accept(server_fd, (sockaddr *)&address, (socklen_t *)&addrlen)) < 0) {
88             perror("Accept failed");
89             exit(EXIT_FAILURE);
90         }
91
92         // Create a thread to handle the client
93         thread client_thread(handle_client, new_socket);
94         client_thread.detach(); // Allow the thread to run independently
95     }
96
97     return 0;
98 }

```

main:

- **Creates the Server Socket:**
  - Calls socket to create a socket file descriptor for IPv4 and TCP.
- **Binds the Socket:**
  - Sets up the server address (IP and port) and binds the socket to this address.
- **Listens for Connections:**
  - Begins listening for incoming connections with a backlog of 3.

- **Accepts Connections:**
    - Enters an infinite loop to accept incoming client connections.
  - **Handles Clients:**
    - For each accepted connection, creates a new thread (client\_thread) to handle the client using the handle\_client function.
    - Detaches the thread so it runs independently.
- 

## Output:

```
Server is listening on port 8080  
Serving file: index.html  
Connection closed for file: index.html
```

## Explanation:

1. **Server is listening on port 8080:**
    - Indicates that the server has successfully started and is actively listening for incoming connections on port 8080.
  2. **Serving file: index.html:**
    - Shows that the server received a request for index.html and is processing it to send back to the client.
  3. **Connection closed for file: index.html:**
    - Signifies that the server has completed sending the requested index.html file to the client and has closed the connection.
-