Name: Noor Akhtar

C++ Programming Internship

Task 4

Web Server in C++

Table of Contents

Multi-Threaded Web Server

```cpp
#include <iostream>
#include <thread>
#include <vector>
#include <string>
#include <cstring>
#include <winsock2.h>
#include <ws2tcpip.h>
#include <fstream>
using namespace std;

#pragma comment(lib, "Ws2_32.lib")

#define PORT 8080
#define BACKLOG 10

void handle(SOCKET clientSocket);

int main() {
    WSADATA wsaData;
    WSAStartup(MAKEWORD(2, 2), &wsaData);

    SOCKET serverSocket, clientSocket;
    struct sockaddr_in serverAddr, clientAddr;
    int addr_size;


    serverSocket = socket(AF_INET, SOCK_STREAM, 0);
    if (serverSocket == INVALID_SOCKET) {
        std::cerr << "Error creating socket" << std::endl;
        WSACleanup();
        return -1;
    }


    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(PORT);
    serverAddr.sin_addr.s_addr = INADDR_ANY;
    memset(serverAddr.sin_zero, '\0', sizeof(serverAddr.sin_zero));

    if (bind(serverSocket, (struct sockaddr*)&serverAddr, sizeof(serverAddr)) ==
SOCKET_ERROR) {
        std::cerr << "Error binding socket" << std::endl;
        closesocket(serverSocket);
        WSACleanup();
        return -1;
    }


    if (listen(serverSocket, BACKLOG) == SOCKET_ERROR) {
        std::cerr << "Error listening on socket" << std::endl;
        closesocket(serverSocket);
        WSACleanup();
        return -1;
    }

    std::cout << "Server is listening on port " << PORT << std::endl;

    while (true) {
        addr_size = sizeof(clientAddr);
        clientSocket = accept(serverSocket, (struct sockaddr*)&clientAddr,
&addr_size);
        if (clientSocket == INVALID_SOCKET) {
            std::cerr << "Error accepting connection" << std::endl;
            continue;
        }

        std::thread(handleClient, clientSocket).detach();
    }

    closesocket(serverSocket);
    WSACleanup();
    return 0;
}

void handleClient(SOCKET clientSocket) {
    char buffer[1024];
    recv(clientSocket, buffer, sizeof(buffer), 0);

    std::string request(buffer);
    std::cout << "Received request:\n" << request << std::endl;


    std::istringstream requestStream(request);
    std::string method, path, version;
    requestStream >> method >> path >> version;

    if (path == "/") {
        path = "/index.html";
    }

    std::ifstream file("www" + path);
    if (file.is_open()) {
        std::string response = "HTTP/1.1 200 OK\r\nContent-Type:
text/html\r\n\r\n";
        std::string line;
        while (getline(file, line)) {
            response += line + "\n";
        }
        file.close();
        send(clientSocket, response.c_str(), response.size(), 0);
    }
    else {
        std::string notFoundResponse = "HTTP/1.1 404 NOT FOUND\r\nContent-Type:
text/html\r\n\r\n<html><body><h1>404 Not Found</h1></body></html>";
        send(clientSocket, notFoundResponse.c_str(), notFoundResponse.size(), 0);
    }

    closesocket(clientSocket);
}
```

Documentation

1. Introduction

The Multi-Threaded Web Server is designed to handle multiple client requests simultaneously. It listens for HTTP requests and serves static HTML files. The server utilizes multi-threading to manage concurrent client connections effectively.

2. Setup Instructions

2.1 File Setup

1. **Static HTML Files**:
   - Ensure you have HTML files that you want to serve in a directory named www.
   - Place an index.html file in the www directory for the root path.

2.2 Environment Setup

Ensure your development environment is set up for C++ development. Follow these general steps:

1. **Install a C++ Compiler**:
   - **Linux**: Use the package manager (e.g., sudo apt-get install g++).
   - **macOS**: Use Homebrew (e.g., brew install gcc).
   - **Windows**: Download and install MinGW or use an IDE like Visual Studio.
2. **Set Up an Integrated Development Environment (IDE)**:
   - **Visual Studio Code**: A lightweight IDE with extensions for C++ development.
   - **CLion**: A powerful IDE for C++ development by JetBrains.
   - **Visual Studio**: A full-featured IDE for Windows.

3. Running the Application

3.1 Compilation and Execution (Linux/macOS)

1. **Compile the Application**:
   - Use a C++ compiler to compile the source code file server.cpp.

     ```
     bash
     Copy code
     g++ -std=c++11 -pthread server.cpp -o server
     ```

2. **Run the Application**:
   - Execute the compiled executable.

     ```
     bash
     Copy code
     ./server
     ```

3.2 Compilation and Execution (Windows using Visual Studio)

1. **Set Up the Project**:
   - Create a new Win32 Console Application project in Visual Studio.
   - Add the server.cpp file to the project.
2. **Configure Project Properties**:
   - Add #include <winsock2.h> and #include <ws2tcpip.h> at the beginning of your file.
   - Add #pragma comment(lib, "Ws2_32.lib") to link the Winsock library.
   - Initialize and clean up Winsock in the main() function.
3. **Build the Project**:
   - Go to Build > Build Solution (or press Ctrl+Shift+B).
4. **Run the Application**:
   - Go to Debug > Start Without Debugging (or press Ctrl+F5).

4. Functionality

4.1 Handling Client Connections

The server listens for incoming client connections and handles each connection in a separate thread.

4.2 Serving Static HTML Files

The server serves static HTML files from the www directory. If the requested file is not found, it returns a 404 Not Found response.

5. Error Handling

The application includes mechanisms to handle errors that may occur during socket operations and file access:

5.1 Socket Operation Errors

- **Socket Creation Errors**:
  - The server checks if the socket is created successfully. If not, an error message is displayed.
- **Binding Errors**:
  - The server checks if the socket is bound to the address successfully. If not, an error message is displayed.
- **Listening Errors**:
  - The server checks if the socket is listening for connections successfully. If not, an error message is displayed.

5.2 File Operation Errors

- **File Opening Errors**:
  - The server checks if the requested file can be opened for reading. If not, it returns a 404 Not Found response.

6. How to Run the Program

1. **Set Up the HTML Files**:
   - Create a directory named www in the same directory as the executable.

    - Place index.html and any other HTML files you want to serve in the www directory.
2. **Compile and Run the Server**:
    - Follow the compilation and execution steps based on your operating system and development environment.
3. **Access the Server**:
    - Open a web browser and navigate to http://localhost:8080.

7. Conclusion

The Multi-Threaded Web Server is a basic implementation that demonstrates key concepts in socket programming, multi-threading, and file handling in C++. It can be further extended and customized to support additional features such as handling different MIME types, logging, and serving other types of files based on specific requirements.