<u>Digital Empowerment Pakistan</u>

Name: Muhammad Ahsan Ali

C++ Programming Internship

Task 4

Building a Multi-Threaded Web Server

Code:

```
#include <iostream>

#include <thread>

#include <string>

#include <fstream>

#include <sys/socket.h>

#include <netinet/in.h>

#include <unistd.h>

#include <sstream>

using namespace std;

class HttpServer {

public:

    explicit HttpServer(int portNumber) {

        serverSocket = socket(AF_INET, SOCK_STREAM, 0);

        if (serverSocket < 0) {

            cerr << "Failed to create socket" << endl;

            exit(EXIT_FAILURE);

        }

        serverAddr.sin_family = AF_INET;

        serverAddr.sin_port = htons(portNumber);

        serverAddr.sin_addr.s_addr = INADDR_ANY;

        if (bind(serverSocket, (struct
sockaddr*)&serverAddr, sizeof(serverAddr)) < 0) {

            cerr << "Binding socket failed" << endl;

            exit(EXIT_FAILURE);

        }

    }
    void begin() {
        if (listen(serverSocket, 5) < 0) {
            cerr << "Listening failed" << endl;
            exit(EXIT_FAILURE);
        }
        cout << "Server started on port 8080..." << endl;
        while (true) {
            sockaddr_in clientAddr;
            socklen_t clientAddrLen = sizeof(clientAddr);
            int clientSock = accept(serverSocket, (struct
sockaddr*)&clientAddr, &clientAddrLen);
            if (clientSock < 0) {
                cerr << "Connection acceptance failed" <<
endl;
                continue;
            }
            thread clientThread(&HttpServer::handleClient,
this, clientSock);
            clientThread.detach();
        }
    }
private:
    void handleClient(int clientSock) {
        string request = receiveRequest(clientSock);
        string resource = extractResource(request);
        sendResponse(clientSock, resource);
        close(clientSock);
    }
    string receiveRequest(int clientSock) {
        char buffer[1024];
        stringstream requestStream;
        while (true) {
            int bytes = read(clientSock, buffer,
sizeof(buffer));
            if (bytes < 0) {
                cerr << "Failed to read from socket" <<
endl;
                return "";
            }
            requestStream.write(buffer, bytes);
            if (requestStream.str().find("\r\n\r\n") !=
string::npos) {
                break;
            }
        }
        return requestStream.str();
    }
    string extractResource(const string& request) {
        size_t methodEnd = request.find(' ');
        if (methodEnd == string::npos || request.substr(0,
methodEnd) != "GET") {
            return "/404.html";
        }

        size_t pathStart = methodEnd + 1;
        size_t pathEnd = request.find(' ', pathStart);
        if (pathEnd == string::npos) {
            return "/404.html";
        }
        string resourcePath = request.substr(pathStart,
pathEnd - pathStart);
        return resourcePath == "/" ? "/index.html" :
resourcePath;
    }
    void sendResponse(int clientSock, const string&
resource) {
        string fullPath = "." + resource;
        ifstream resourceFile(fullPath);
        if (!resourceFile) {
            fullPath = "./404.html";
            resourceFile.open(fullPath);
        }
        stringstream responseBody;
        responseBody << resourceFile.rdbuf();
        string response = "HTTP/1.1 200 OK\r\n"
                          "Content-Type: text/html\r\n"
                          "Content-Length: " +
to_string(responseBody.str().length()) + "\r\n"
                          "\r\n" + responseBody.str();
        write(clientSock, response.c_str(),
response.length());
    }
    int serverSocket;
    sockaddr_in serverAddr;
```

```
};
int main() {
    HttpServer server(8080);
    server.begin();
    return 0;
}
```

Documentation for Building a Multi-Threaded Web Server

Overview

This C++ program implements a basic multi-threaded HTTP server. It listens for HTTP requests on a specified port, handles multiple clients concurrently using threads, and serves static HTML files. The server can respond with requested HTML files or a default 404 error page if the requested file is not found.

Class: `HttpServer`

Constructor: `HttpServer(int portNumber)`

- **Parameters:**
  - `int portNumber`: The port number on which the server listens for incoming connections.
- **Description:** Initializes the server by creating a socket, binding it to the specified port, and preparing the server to accept incoming connections.

Method: `void begin()`

- **Description:** Starts the server to listen for incoming connections. It runs an infinite loop where it accepts client connections, creates a new thread to handle each client, and continues listening for new connections.

Private Method: `void handleClient(int clientSock)`

- **Parameters:**
  - `int clientSock`: The socket descriptor for the client connection.
- **Description:** Handles an individual client connection. It reads the client's request, determines the requested resource, and sends the appropriate HTTP response.

Private Method: `string receiveRequest(int clientSock)`

- **Parameters:**

    - `int clientSock`: The socket descriptor for the client connection.
- **Returns:** A `string` containing the full HTTP request received from the client.
- **Description:** Reads the HTTP request from the client socket and returns it as a string.

Private Method: `string extractResource(const string& request)`

- **Parameters:**
    - `const string& request`: The HTTP request string received from the client.
- **Returns:** A `string` containing the path to the requested resource. Defaults to `/404.html` if the request is invalid or the resource is not specified.
- **Description:** Parses the HTTP request to extract the requested resource path. It handles basic validation and defaults to serving the index page (`/index.html`) if no specific file is requested.

Private Method: `void sendResponse(int clientSock, const string& resource)`

- **Parameters:**
    - `int clientSock`: The socket descriptor for the client connection.
    - `const string& resource`: The path to the requested resource.
- **Description:** Sends an HTTP response to the client. If the requested resource is found, the server responds with a 200 OK status and the file content. If not found, it responds with a 404 Not Found status and the content of the 404 error page.

Main Function: `int main()`

- **Description:** The entry point of the program. It creates an instance of the `HttpServer` class with port 8080 and starts the server.

Usage

To use this server, compile the code with a C++ compiler that supports C++11 or later, and then run the compiled executable. The server will listen on port 8080 for incoming HTTP requests.

```
bash
Copy code
g++ -std=c++11 -o http_server server.cpp -lpthread
./http_server
```

After running the server, you can access it using a web browser or any HTTP client by navigating to `http://localhost:8080`.

Notes

- The server expects to find HTML files in the same directory as the executable. The default file served is `index.html`, and `404.html` is used for error handling.
- The server handles simple GET requests and responds with static HTML content. It does not support other HTTP methods (e.g., POST, PUT) or dynamic content generation.

- Error handling and security features are minimal and intended for educational purposes. For a production environment, additional considerations are necessary.