#include <iostream>

#include <fstream>

#include <sstream>

#include <thread>

#include <vector>

#include <cstring>

#include <netinet/in.h>

#include <unistd.h>

#define PORT 8080

#define BUFFER\_SIZE 1024

Void handleClient(int clientSocket);

Std::string getFileContent(const std::string& path);

Int main() {

Int serverSocket, newSocket;

Struct sockaddr\_in address;

Int addrlen = sizeof(address);

// Creating socket file descriptor

If ((serverSocket = socket(AF\_INET, SOCK\_STREAM, 0)) == 0) {

Std::cerr << “Socket failed” << std::endl;

Exit(EXIT\_FAILURE);

}

// Configure server address

Address.sin\_family = AF\_INET;

Address.sin\_addr.s\_addr = INADDR\_ANY;

Address.sin\_port = htons(PORT);

// Bind the socket to the network address and port

If (bind(serverSocket, (struct sockaddr \*)&address, sizeof(address)) < 0) {

Std::cerr << “Bind failed” << std::endl;

Exit(EXIT\_FAILURE);

}

// Start listening for incoming connections

If (listen(serverSocket, 10) < 0) {

Std::cerr << “Listen failed” << std::endl;

Exit(EXIT\_FAILURE);

}

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

// Vector to hold threads

Std::vector<std::thread> threads;

While (true) {

// Accept new incoming connection

If ((newSocket = accept(serverSocket, (struct sockaddr \*)&address, (socklen\_t\*)&addrlen)) < 0) {

Std::cerr << “Accept failed” << std::endl;

Continue;

}

// Create a new thread for each client connection

Threads.emplace\_back(std::thread(handleClient, newSocket));

}

// Close the server socket

Close(serverSocket);

Return 0;

}

Void handleClient(int clientSocket) {

Char buffer[BUFFER\_SIZE] = {0};

Int bytesRead = read(clientSocket, buffer, BUFFER\_SIZE);

If (bytesRead <= 0) {

Close(clientSocket);

Return;

}

Std::string request(buffer);

Std::cout << “Received request:\n” << request << std::endl;

Std::istringstream requestStream(request);

Std::string method, path;

requestStream >> method >> path;

if (method == “GET”) {

if (path == “/”) {

path = “/index.html”; // Default file

}

Std::string fullPath = “.” + path; // Assuming files are in the current directory

Std::string content = getFileContent(fullPath);

If (content.empty()) {

Std::string notFoundResponse = “HTTP/1.1 404 Not Found\r\nContent-Length: 13\r\n\r\n404 Not Found”;

Send(clientSocket, notFoundResponse.c\_str(), notFoundResponse.size(), 0);

} else {

Std::string okResponse = “HTTP/1.1 200 OK\r\nContent-Type: text/html\r\nContent-Length: “ + std::to\_string(content.size()) + “\r\n\r\n” + content;

Send(clientSocket, okResponse.c\_str(), okResponse.size(), 0);

}

}

// Close the connection socket

Close(clientSocket);

}

Std::string getFileContent(const std::string& path) {

Std::ifstream file(path);

If (!file.is\_open()) {

Return “”;

}

Std::stringstream buffer;

Buffer << file.rdbuf();

Return buffer.str();

}