TASK:04

TITLE: BUILDING A MULTI-THREADED WEB SERVER

AUTHOR: SIDRA YOUNAS

DATE:31-08-2024

COMPANY:

DIGITALEMPOWERMENTNETWORK

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:

o Setting up socket programming to handle HTTP requests and responses

o Implementing multi-threading using the C++ Standard Library thread support

o Serving static HTML files from a specified directory

o Handling concurrent client connections

<u>Code Explanation</u>

1. Including Necessary Headers

- **winsock2.h** and **ws2tcpip.h**: Required for Windows socket programming.
- **iostream**: Provides basic I/O functionality.
- **thread**: Used for creating and managing threads, allowing concurrent client handling.
- **vector**: Provides dynamic array functionality (not used here but included for potential future needs).
- **fstream**: Allows file I/O operations, used here to read HTML files.
- **sstream**: Used to work with strings, especially for request parsing

2. **Linking Winsock Library**

This directive tells the compiler to link against the Winsock library, which provides the necessary networking functions on Windows

3. **Initializing Winsock**

- **WSAStartup** initializes the Winsock DLL. It's required before calling any Winsock functions.
- **MAKEWORD(2, 2)** specifies the version of Winsock (2.2) to be used.

4. **Creating a Socket**

- **socket()** creates a new socket.
- **AF_INET** specifies the use of IPv4 addresses.

- **SOCK_STREAM** indicates the use of TCP (connection-oriented communication).
- **IPPROTO_TCP** explicitly specifies the use of the TCP protocol.

5. **Binding the Socket**

- **sockaddr_in** is a structure used to specify the address family, IP address, and port number.
- **INADDR_ANY** allows the server to accept connections on any available network interface.
- **htons(8080)** converts the port number 8080 from host byte order to network byte order.

6. Listening for Incoming Connections

- **listen()** puts the socket into listening mode, where it waits for clients to connect.
- **SOMAXCONN** is the maximum number of connections allowed in the queue.
- **cout** prints a message to the console indicating that the server is listening.

7. Accepting Client Connections

- **accept()** waits for a client to connect, returning a new socket that represents the connection to that client.
- **thread** creates a new thread to handle each client connection, allowing the server to manage multiple clients simultaneously.
- **detach()** allows the thread to run independently from the main thread.

8. Handling Client Requests

- The request is parsed to extract the method (usually GET) and the requested path (e.g., /index.html).
- If the requested path is /, it defaults to serving index.html.

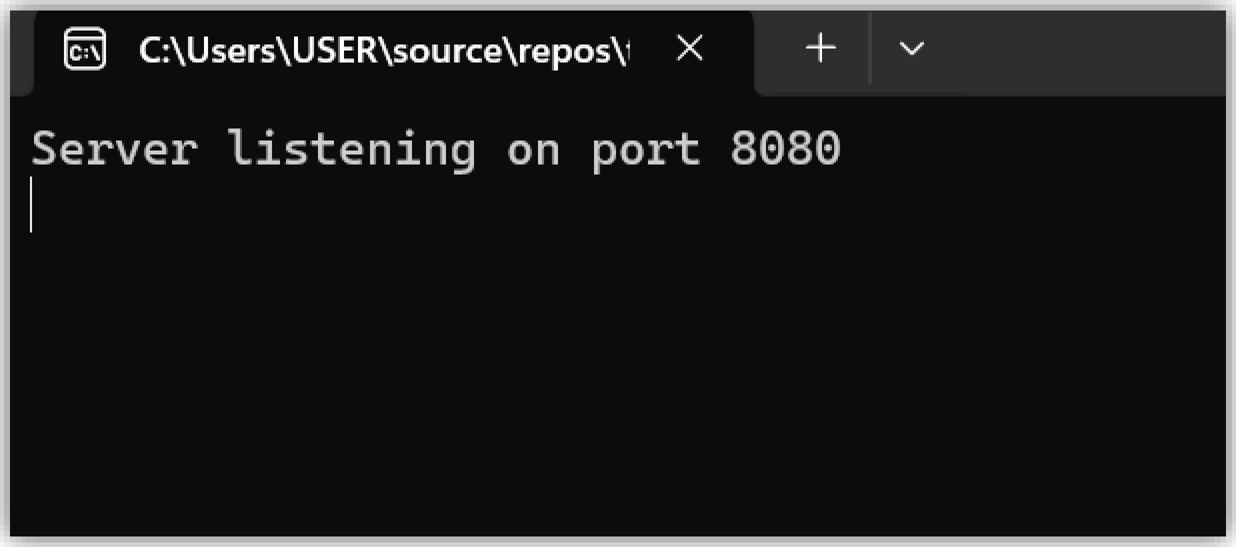
- The request is parsed to extract the method (usually GET) and the requested path (e.g., /index.html).
- If the requested path is /, it defaults to serving index.html.

<u>CODE:</u>

```
#include <winsock2.h>
#include <ws2tcpip.h>
#include <iostream>
#include <thread>
#include <vector>
#include <fstream>
#include <sstream>

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

void handleClient(SOCKET clientSocket);
```

```
int main() {
    WSADATA wsaData;
    int iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);
    if (iResult != 0) {
        std::cerr << "WSAStartup failed: " << iResult << std::endl;
        return 1;
    }

    SOCKET serverSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (serverSocket == INVALID_SOCKET) {
        std::cerr << "Socket creation failed: " << WSAGetLastError() << std::endl;
        WSACleanup();
        return 1;
    }

    sockaddr_in serverAddr;
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_addr.s_addr = INADDR_ANY;
    serverAddr.sin_port = htons(8080);

    if (bind(serverSocket, (sockaddr*)&serverAddr, sizeof(serverAddr)) == SOCKET_ERROR) {
        std::cerr << "Bind failed: " << WSAGetLastError() << std::endl;
        closesocket(serverSocket);
        WSACleanup();
        return 1;
    }

    if (listen(serverSocket, SOMAXCONN) == SOCKET_ERROR) {
        std::cerr << "Listen failed: " << WSAGetLastError() << std::endl;
        closesocket(serverSocket);
        WSACleanup();
        return 1;
    }

    std::cout << "Server listening on port 8080" << std::endl;

    while (true) {
        SOCKET clientSocket = accept(serverSocket, nullptr, nullptr);
        if (clientSocket == INVALID_SOCKET) {
            std::cerr << "Accept failed: " << WSAGetLastError() << std::endl;
            closesocket(serverSocket);
            WSACleanup();
            return 1;
        }
        std::thread clientThread(handleClient, clientSocket);
        clientThread.detach();
    }

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

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

    // Basic request parsing
    std::string request(buffer);
    std::istringstream requestStream(request);
    std::string method, path;
    requestStream >> method >> path;

    // Default to index.html if path is /
    if (path == "/") path = "/index.html";

    // Serve the requested file
    std::ifstream file("." + path);
    if (file.is_open()) {
        std::stringstream fileContent;
        fileContent << file.rdbuf();
        std::string html = fileContent.str();

        std::string response = "HTTP/1.1 200 OK\r\nContent-Type: text/html\r\nContent-Length: "
+ std::to_string(html.length()) + "\r\n\r\n" + html;
        send(clientSocket, response.c_str(), response.length(), 0);
    }
    else {
        std::string notFound = "HTTP/1.1 404 Not Found\r\nContent-Type:
text/html\r\n\r\n<h1>404 Not Found</h1>";
        send(clientSocket, notFound.c_str(), notFound.length(), 0);
    }

    closesocket(clientSocket);
}
```

<u>OUTPUT:</u>



```
Server listening on port 8080
```