[Single threaded Server]

* C++ Server for Windows

```cpp
#include <winsock2.h>
#include <ws2tcpip.h>
#include <iostream>
#include <string>

// Link with ws2_32.lib
#pragma comment(lib, "ws2_32.lib")

const int PORT = 8080;

int main() {
    WSADATA wsaData;
    SOCKET server_fd = INVALID_SOCKET, new_socket = INVALID_SOCKET;
    struct sockaddr_in address;
    int addrlen = sizeof(address);
    char buffer[1024] = { 0 };
    const std::string response =
        "HTTP/2 200 OK\r\n"
        "Content-Type: text/plain\r\n"
        "Content-Length: 13\r\n"
        "\r\n"
        "Hello, World!";

    // Initialize Winsock
    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
        std::cerr << "WSAStartup failed: " << WSAGetLastError() << "\n";
        return -1;
    }

    // Create socket
    if ((server_fd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) == INVALID_SOCKET) {
        std::cerr << "Socket creation failed: " << WSAGetLastError() << "\n";
        WSACleanup();
        return -1;
    }

    // Bind socket to port
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(PORT);
    if (bind(server_fd, (struct sockaddr*)&address, sizeof(address)) == SOCKET_ERROR) {
        std::cerr << "Bind failed: " << WSAGetLastError() << "\n";
        closesocket(server_fd);
        WSACleanup();
        return -1;
    }
```

port

OS system call

Bind() internally do system call

```
    std::cout << "Server is listening on port " << PORT << "...\n";

    while (true) {
        std::cout << "Single Threaded with 'Hello, World!'\n";
        // Accept incoming connection
        if ((new_socket = accept(server_fd, (struct sockaddr*)&address, &addrlen)) == INVALID_SOCKET) {
            std::cerr << "Accept failed: " << WSAGetLastError() << "\n";
            closesocket(server_fd);
            WSACleanup();
            return -1;
        }

        // Read request (for demonstration, we ignore its contents)
        recv(new_socket, buffer, 1024, 0);

        // Send response
        send(new_socket, response.c_str(), response.size(), 0);
        std::cout << "Responded with 'Hello, World!'\n";

        // Close the connection
        closesocket(new_socket);
    }

    // Cleanup
    closesocket(server_fd);
    WSACleanup();
    return 0;
}
```

↳ accept()
↓
Blocking call
↓
Internally OS System call

↳ Can't run until a client connects.

* We compiles this file and executed the executable

Steps:- ① Creat Socket (OS code)
② Bind Socket to port (OS code)
③ While loop
↳ calls accept()

↳ No this C++ program/process is put to wait state by OS

* Until this process gets client connection, code underneath the accept() call is not run.

* Once client connects, OS gives pointer to accept queue present in memory.

* Now this process has pointer

(4) Reads Request data

(5) Send response

(6) Again next iteration of

while loop starts

Problems

① What will happen if the code after accept() takes 10 seconds

↳ Consider in 10 seconds, this sever recieves 10K connection

↳ This will Bloat the memory

↳ CPU execution Increases

② Only 1 request is served at a time, Multicore CPU or Multiple CPU are not being utilised properly

```
Server is listening on port 8080...
Single Threaded with 'Hello, World!'
Responded with 'Hello, World!'
Single Threaded with 'Hello, World!'
```

iteration 1

iteration 2

waiting after accept() call

* Also this C++ server is infinitely running because of (while (true))

↳ Its what web servers are. Always running server waiting for connection.

* Single threaded blocking servers are (never) implemented