计算机网络课程实验报告

姓名：孟启轩

学号：2212452

专业：计算机科学与技术

实验 1：利用 Socket，编写一个聊天程序

一、实验内容

（1）给出你聊天协议的完整说明。

（2）利用 C 或 C++语言，使用基本的 Socket 函数完成程序。不允许使用 CSocket 等封装后的类编写程序。

（3）使用流式 Socket 完成程序。

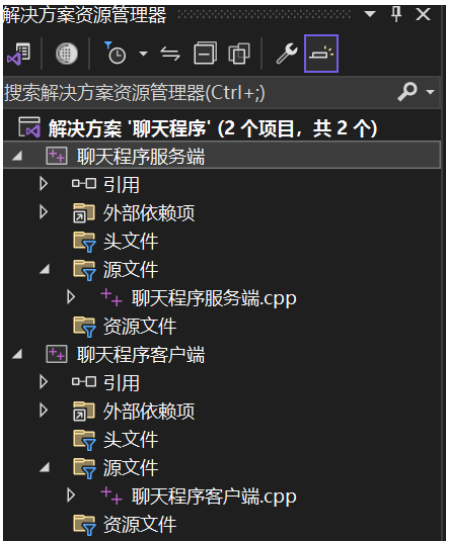
（4）程序应有基本的对话界面，但可以不是图形界面。程序应有正常的退出方式。

（5）完成的程序应能支持多人聊天，支持英文和中文聊天。

（6）编写的程序应该结构清晰，具有较好的可读性。

二、实验准备

使用 visual studio 2022 进行编程实现。首先，在同一解决方案中创建两个项目，一个服务端，一个客户端。



再把 cpp 文件的 SDL 检查设置为否。

点击目录最上方的“解决方案”设置属性，将两个项目设置为同时启动。

三、实验过程

（1）聊天协议

1. 用户连接：

用户启动客户端。

客户端通过 TCP 连接到服务器，使用 socket 函数创建连接。

连接成功后，客户端首先发送用户名给服务器。

服务器接收到用户名后，将该用户添加到在线用户列表。

服务器广播消息：“用户名 进入聊天室”，并更新在线人数。

2. 消息类型：

用户加入消息：”用户名 进入聊天室\n”

用户退出消息：”用户名 退出聊天室\n”

聊天消息：”用户名: 消息内容\n”

3. 消息语义

用户加入：连接后，客户端发送用户名，服务器广播用户加入消息，并更新在线人数。

用户退出：当用户输入“/exit”或“/退出”命令时，客户端断开连接。服务器接收到 断开信号后，从在线用户列表中移除该用户，广播用户退出消息，并更新在线人数。

聊天消息：用户输入的消息由客户端发送，服务器接收后广播给所有连接的用户。

4. 消息发送与接收时序

用户输入消息：用户在客户端输入一条消息。

客户端发送消息：客户端将输入的消息通过 send 函数发送到服务器。

服务器接收消息：服务器通过 recv 函数接收来自客户端的消息。

服务器格式化消息：服务器将接收到的消息格式化为：”用户名: 消息内容\n”。

服务器广播消息：服务器将消息格式化为：“用户名: 消息内容”，然后通过 send 函数广播给所有连接的用户。

客户端接收消息：所有在线的客户端通过 recv 函数接收服务器广播的消息。

客户端显示消息：客户端将接收到的消息输出到用户界面。

5. 其他要点

线程安全：由于多个用户可以同时连接，使用互斥锁（mutex）来保护共享资源，如用户列表和在线人数。

错误处理：在连接、接收和发送数据时，需检查错误并妥善处理，例如连接失败时给出提示。

UTF-8 支持：客户端和服务器应支持 UTF-8 编码，以便处理中文和英文的消息。

（2）服务器端：处理用户连接、消息接收和广播，同时维护在线用户列表。使用线程处理每个客户端的连接。

1. 包含头文件和宏定义

```
#include <iostream>                 // 引入输入输出流库
#include <thread>                   // 引入线程库
#include <vector>                   // 引入向量库
#include <string>                   // 引入字符串库
#include <mutex>                    // 引入互斥量库
#include <map>                      // 引入映射（字典）库
#include <algorithm>                // 引入算法库（用于算法功能）
#include <winsock2.h>               // 引入Winsock库，用于网络编程

#pragma comment(lib, "ws2_32.lib") // 链接Winsock库

#define MAX_BUFFER_SIZE 1024     // 定义最大缓冲区大小
#define SERVER_PORT 8888         // 定义服务器端口
```

引入必要的头文件，用于输入输出、线程处理、数据结构、网络编程等。

定义最大缓冲区大小和服务器监听端口。

2. 全局变量

```
// 全局变量
vector<SOCKET> clients;                 // 存储所有已连接客户端的Socket
map<SOCKET, string> client_names;       // 存储客户端Socket与用户名的映射
mutex clients_mutex;                    // 互斥量，用于保护客户端列表的访问
```

clients 用于存储所有连接的客户端。

client_names 用于映射客户端的 socket 和用户名。

clients_mutex 用于保护对 clients 和 client_names 的访问，以避免数据竞争。

3. 广播函数和在线人数广播函数

```
// 广播消息给所有连接的客户端
void broadcast(const string& message) {
    lock_guard<mutex> lock(clients_mutex);  // 自动加锁，保护对clients的访问
    for (SOCKET client : clients) {         // 遍历所有客户端Socket
        send(client, message.c_str(), message.size(), 0); // 发送消息
    }
}
```

用于向所有已连接的客户端发送消息。参数包括目标 Socket、消息内容和消息长度。

使用 lock_guard 进行自动加锁，保证在遍历客户端列表时不发生数据竞争。

```
void broadcast_user_count() {
    string count_message = "当前在线人数: " + to_string(clients.size()) + "\n";
    broadcast(count_message);
    cout << count_message;
}
```

计算并广播当前在线人数。

4. 客户端处理函数（主要函数）

```
void handle_client(SOCKET client_socket) {
    char buffer[MAX_BUFFER_SIZE];
    int bytes_received;

    // 获取用户名
    bytes_received = recv(client_socket, buffer, MAX_BUFFER_SIZE, 0);
    ...
    // 通知其他用户
    broadcast(join_message);
    broadcast_user_count();  // 显示在线人数
    ...
}
```

处理每个客户端的逻辑，包括接收用户名、处理消息和用户退出。

recv: 从 client_socket 接收数据。参数包括目标缓冲区、缓冲区大小和标志（通常为0）。

当用户连接时，添加到 clients 列表，并通知其他用户。

在用户发送消息时，广播给所有其他用户。

5. Socket 初始化、创建、绑定和监听

代码分析在注释中

```
int main() {
    WSADATA wsaData;
    // 初始化Winsock库，指定版本为2.2
    WSAStartup(MAKEWORD(2, 2), &wsaData);

    // 创建一个TCP类型的Socket
    SOCKET server_socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    sockaddr_in server_addr;  // 定义服务器地址结构
    server_addr.sin_family = AF_INET;  // 使用IPv4地址
    server_addr.sin_addr.s_addr = INADDR_ANY;  // 允许接受任意IP地址的连接
    server_addr.sin_port = htons(SERVER_PORT);  // 转换端口号为网络字节序

    // 将Socket与指定的地址和端口绑定
    bind(server_socket, (sockaddr*)&server_addr, sizeof(server_addr));
    // 设置Socket为监听状态，准备接受连接请求
    listen(server_socket, SOMAXCONN);

    cout << "服务器启动，等待客户端连接...\n";

    // 主循环，持续接受客户端连接
    while (true) {
        // 接受来自客户端的连接请求，返回一个新的Socket用于通信
        SOCKET client_socket = accept(server_socket, nullptr, nullptr);
        // 检查连接是否成功
        if (client_socket != INVALID_SOCKET) {
            // 为每个连接的客户端创建一个新线程处理
            thread(handle_client, client_socket).detach();
        }
    }

    // 关闭服务器Socket（这行代码在无限循环中实际上不会被执行）
    closesocket(server_socket);
    // 清理Winsock资源
    WSACleanup();
    return 0;
```

（3）客户端：负责与用户交互，发送用户输入的消息，并接收服务器的消息显示。

1. 包含头文件和宏定义

```cpp
#include <iostream>        // 引入输入输出流库
#include <thread>          // 引入线程库
#include <string>          // 引入字符串库
#include <winsock2.h>      // 引入Winsock库，用于网络编程

#pragma comment(lib, "ws2_32.lib")  // 链接Winsock库

#define MAX_BUFFER_SIZE 1024  // 定义最大缓冲区大小
#define SERVER_IP "127.0.0.1"  // 定义服务器IP地址
#define SERVER_PORT 8888       // 定义服务器端口号
```

包含必要的头文件，定义缓冲区大小和服务器地址/端口。

2. 消息接收函数（主要函数）

```cpp
void receive_messages(SOCKET client_socket) {
    char buffer[MAX_BUFFER_SIZE];  // 定义接收缓冲区
    int bytes_received;            // 定义接收到的字节数

    while (true) {
        // 从服务器接收消息
        bytes_received = recv(client_socket, buffer, MAX_BUFFER_SIZE, 0);
        if (bytes_received <= 0) {  // 如果接收失败或连接关闭
            break;  // 退出循环
        }
        buffer[bytes_received] = '\0';  // 将接收到的数据结尾设置为'\0'，形成字符串
        cout << buffer;  // 输出接收到的消息
    }
}
```

用于接收来自服务器的消息并打印到控制台。

通过死循环一直接收消息，直到连接关闭。

3. 主函数（与服务端大致相同）

代码分析在注释中

```cpp
int main() {
    WSADATA wsaData;  // 定义WSADATA结构体，用于存储Winsock的初始化信息
    // 初始化Winsock库
    WSAStartup(MAKEWORD(2, 2), &wsaData);

    // 创建客户端Socket
    SOCKET client_socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    sockaddr_in server_addr;  // 定义服务器地址结构
    server_addr.sin_family = AF_INET;  // 设置地址族为IPv4
    server_addr.sin_addr.s_addr = inet_addr(SERVER_IP);  // 将IP地址转换并赋值
    server_addr.sin_port = htons(SERVER_PORT);  // 将端口号转换为网络字节序

    // 尝试连接到服务器
    if (connect(client_socket, (sockaddr*)&server_addr, sizeof(server_addr)) == SOCKET_ERROR) {
        cerr << "连接到服务器失败\n";  // 如果连接失败，输出错误信息
        closesocket(client_socket);  // 关闭Socket
        WSACleanup();  // 清理Winsock资源
        return 1;  // 返回错误码
    }

    string username;  // 定义用户名字符串
    cout << "输入用户名: ";  // 提示用户输入用户名
    getline(cin, username);  // 获取用户输入的用户名

    // 发送用户名到服务器
    send(client_socket, username.c_str(), username.size(), 0);

    // 启动一个线程用于接收消息
    thread(receive_messages, client_socket).detach();  // 线程分离，使其独立运行

    // 主线程用于发送消息
    string message;  // 定义消息字符串
    while (true) {
        getline(cin, message);  // 获取用户输入的消息
        // 检查是否输入退出命令
        if (message == "/exit" || message == "/退出") {
            break;  // 退出循环
        }
        // 发送消息到服务器
        send(client_socket, message.c_str(), message.size(), 0);
    }

    closesocket(client_socket);  // 关闭Socket
    WSACleanup();  // 清理Winsock资源
    return 0;
}
```

先通过 connect 连接到服务器，读取输入的用户名并发送到服务器。

启动一个线程用于接收消息，同时在主线程中读取用户输入并发送到服务器。

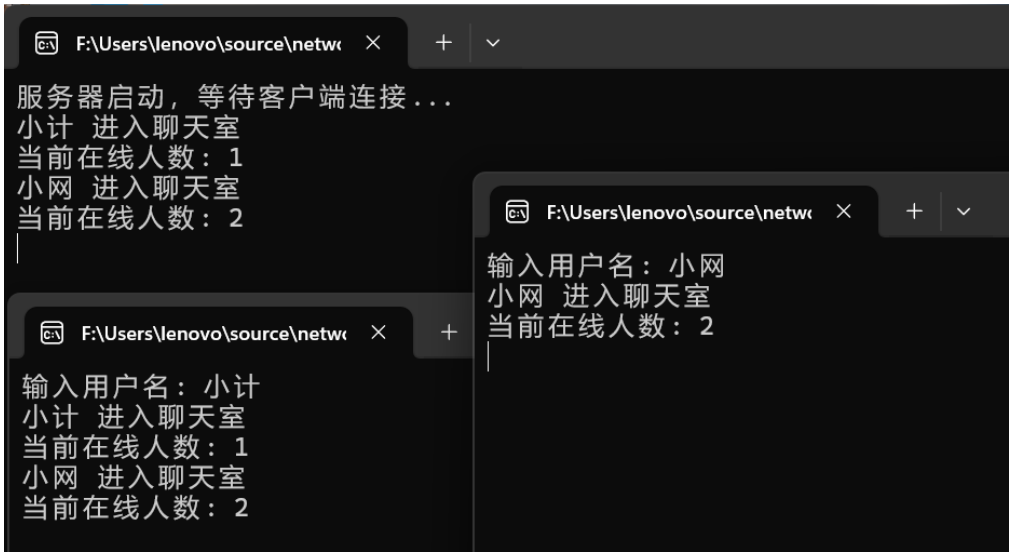
（4）点击“生成”->“生成解决方案”将程序打包为可执行文件。

名称	修改日期	类型
聊天程序服务端.exe	2024/10/14 22:23	应用程序
聊天程序服务端.pdb	2024/10/14 22:23	Program Debug Da...
聊天程序客户端.exe	2024/10/14 22:25	应用程序
聊天程序客户端.pdb	2024/10/14 22:25	Program Debug Da...

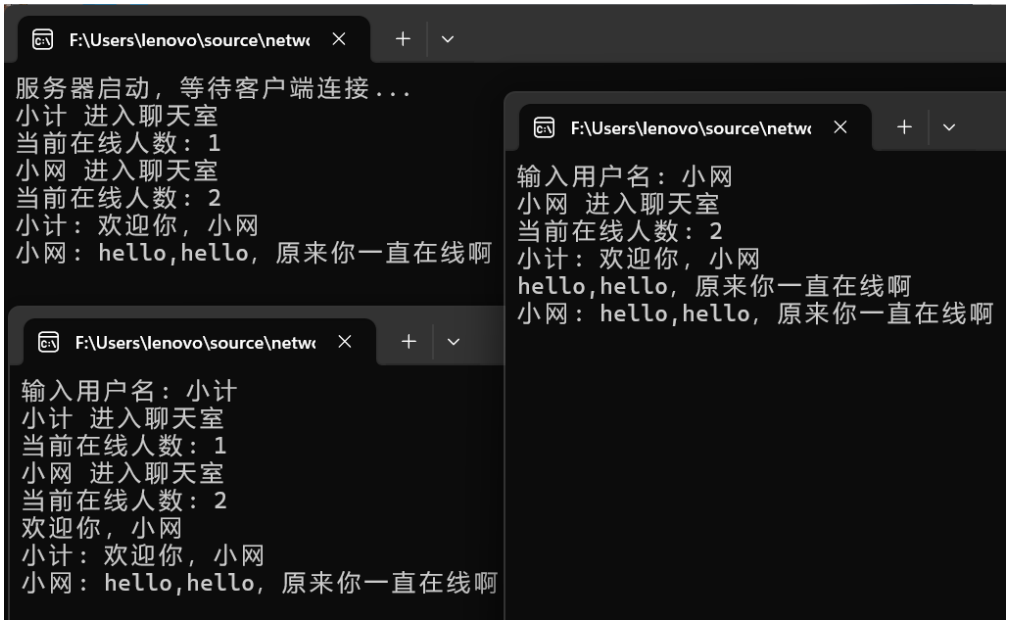
四、程序演示

①首先双击.exe 文件启动服务端与客户端，客户端可以启动多个。

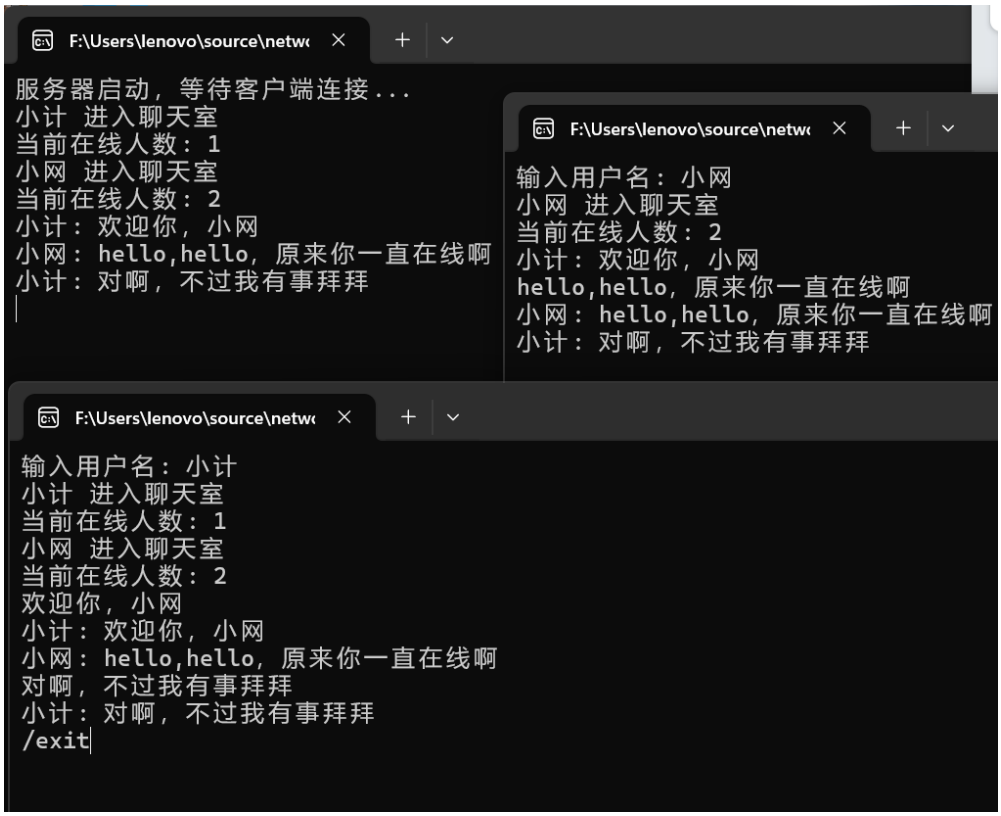
②分别在客户端输入用户名，进入聊天室，服务端会广播进入聊天室提示信息，并更新在线人数。

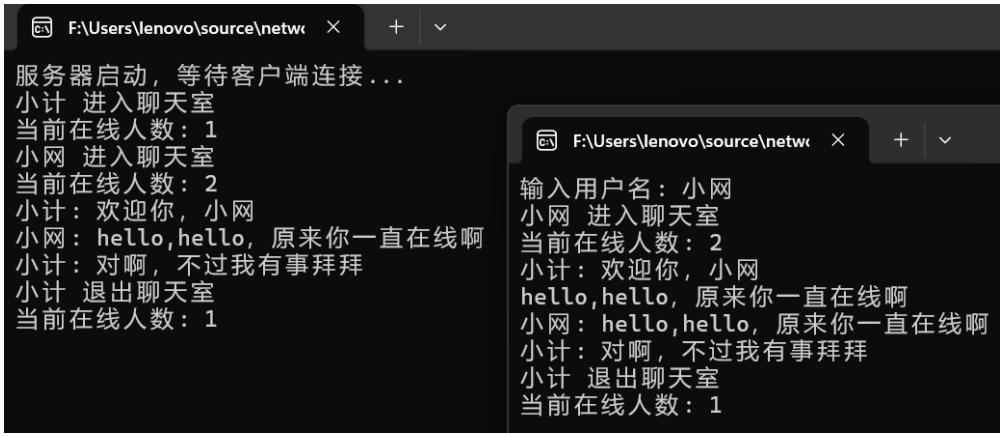


③然后进行对话。



④退出聊天室，输入“/exit”或“/退出”，当然，也可以直接关闭终端。服务端会广播退出提示信息并更新在线人数。





五、实验遇到的问题以及感悟

本次实验遇到了一些问题，比如对 Socket API 的使用不熟悉，在创建、绑定、监听等过程中遇到错误；如何用线程处理多个客户端的连接，确保线程安全；在客户端发送消息与服务端显示消息时的格式问题，换行符的处理等。当然，问题都一一解决了。

通过本次实验，我深入掌握了 Socket 编程的基础，熟悉了 TCP/IP 协议，能够独立实现客户端与服务端的通信。此外，我学习了如何使用线程处理并发连接，提高了对多线程编程的理解。遇到问题时，我通过调试和查阅文档增强了问题解决能力，同时学会了模块化组织代码，使其结构清晰且易于维护。整体而言，这次实验不仅提升了我的编程技能，还加深了对网络通信原理的理解，为未来的学习和工作打下了坚实的基础。