多人聊天室

姓名：李娅琦 学号：2213603 班级：计科

**1.实验要求及概述**

实现多人在线聊天室，有基本的对话界面并支持中英文信息。本次实验实现的聊天室由服务器端和客户端组成，其中服务器端负责连接客户端，传递消息；客户端则实现用户之间的发送和接收消息功能。

**2.聊天协议**

聊天协议基于TCP协议，使用流式Socket进行数据传输。服务器端和客户端都遵循以下聊天协议：

* 连接建立：连接服务器后首先发送客户端的name
* 消息格式：为纯文本（支持中英文），包括name+@+时间+内容
* 时间戳：广播消息前在每条消息前添加发送时间的时间戳，格式为“YYYY-MM-DD HH:MM:SS”
* 消息广播：服务器将消息只发送给其他客户端
* 退出机制：客户端发送“exit”时，服务器将其从客户端列表中移除，并广播该客户端已退出的消息

**3.服务器端（serve.cpp）**

在服务器端定义客户信息结构体并创建该结构体的容器存储连接服务器的**客户端的socket和name**，以及采用原子操作记录**客户端数量。**

|  |  |
| --- | --- |
| **变量类型** | **名称** |
| **vector<clientmessage>** | **clientlist** |
| **atomic<int>** | **clientnumber(0)** |

其中**clientmessage**定义如下：

**struct clientmessage {**

**SOCKET clientsocket;**

**string name;**

**};**

**主要有main函数,客户端处理函数(client)以及消息广播函数(broadcast)。**

**3.1 main函数：**主要功能为初始化WinSock，创建服务器的Socket，并绑定IP地址和端口号，之后监听客户端连接。其中绑定IP与端口代码中，自己设置了IP为”127.0.0.1”,端口号为8888。

循环监听是否有客户端连接时，在有客户端连接会创建新线程并调用handle函数以处理客户端请求，关键代码实现如下：

**SOCKET clientsocket = accept(sock, (SOCKADDR\*)&clientaddr, &addrlen);//创建socket**

**DWORD threadid; // 创建新的线程处理客户端请求**

**HANDLE hthread = CreateThread(NULL, 0, handle, (LPVOID)clientsocket, 0, &threadid);**

**3.2 Client函数：**主要功能为处理客户端的连接，接收客户端发送的消息，并将其广播给其他客户端。当客户端发送“exit”指令或连接断开时，关闭Socket并从客户端列表中移除该客户端。细节如下：

首先接收客户端的socket，之后消息的接收分为两个阶段。

第一，前一个阶段每一个新连接的客户端第一次输入都将为它的名字，并将新客户端存入clientlist中。

**clientlist.push\_back({ clientsocket, message });clientnumber++;**

**cout << "新客户端 " << message << " 连接成功（目前有聊天室人数为：" << clientnumber <<")"<< endl;**

1. 下面关于消息的接收以及广播的循环。服务器不断循环监听对收到的消息进行判断，当接收到的消息为“exit”时，即表示客户端想要退出与服务器的连接，会执行以下操作：
2. 从客户端列表移除该客户端
3. 向各个客户端广播“\*\*已退出”
4. 关闭该客户端的套接字clientsocket。

关于移除该客户端的代码细节如下：

**broadcast("已退出", clientsocket, sendername);**

**cout << "客户端 " << i->name << " 已从客户端列表中移除" << endl;**

**i = clientlist.erase(i);**

**clientnumber--;cout << "目前聊天室的人数为：" << clientnumber << endl;**

**closesocket(clientsocket);**

**3.3 Broadcast函数：**将消息广播给除发送者之外的所有客户端，并在消息前添加时间戳。其中在windows下的c++可以直接调用chrono中的函数to\_time\_t以此直接获得消息发送的时间。

但是要将时间与消息内容拼接为字符串进行发送需要将时间戳转化为字符串，因此使用localtime\_s和strftime 来转换时间戳，代码细节如下：

**// 获取当前时间的时间戳并转换为字符串**

**time\_t now = chrono::system\_clock::to\_time\_t(chrono::system\_clock::now());**

**tm localtime;**

**localtime\_s(&localtime, &now);**

**char timeBuffer[64];**

**strftime(timeBuffer, sizeof(timeBuffer), "%Y-%m-%d %H:%M:%S", &localtime);**

**// 完整的消息并发送**

**string fullMessage = senderName + " @ " + timeBuffer + ": " + message;**

**send(client.clientsocket, fullMessage.c\_str(), fullMessage.length(), 0);**

**cout << "客户端[" << senderName << "]在" << timeBuffer << " 发送消息: " << message;**

**3.客户端（client.cpp）**

在客户器端定义**客户端的socket**以及**接收消息线程，**以及**keep**用来决定是否循环接收和发送对话。

|  |  |
| --- | --- |
| **变量类型** | **名称** |
| **SOCKET** | **sock** |
| **pthread\_t** | **receivethread** |
| **bool** | **keep** |

**主要有main函数以及消息接收函数(receiveMessages)。**

**3.1 main函数：**前面与服务器端相同（初始化WinSock以及创建Socket），不同的是使用connect()函数连接服务器对应IP以及端口号。

第一，使用getline()从终端获取客户端名称并发送至服务器端。

第二，创建线程用于接收消息

第三，在循环中获取输入并发送

具体代码细节如下：

**// 创建接收消息的线程**

**pthread\_create(&receivethread, nullptr, receiveMessages, nullptr);**

**string message;**

**while (keep) {**

**cout << "请输入要发送的消息（输入exit结束）: "<<endl;**

**getline(cin, message);**

**send(sock, message.c\_str(), message.length(), 0);**

**if (message == "exit") {// 如果用户输入 "exit"，则退出循环**

**break;**

**}**

**}**

**3.2receiveMessages函数：**

该函数为线程函数，它在循环中接收来自服务器的消息。

当接收到消息后，它会打印这个消息，并提示用户输入要发送的消息。

否则的话，表示服务器关闭了连接，此时令keep=false可在下一次输入回车时退出消息的接收和发送的循环，同时待线程结束关闭套接字，这样可以确保在服务器端异常关闭而客户端还在运行时，客户端能够及时的退出。

**if (recvResult > 0) {**

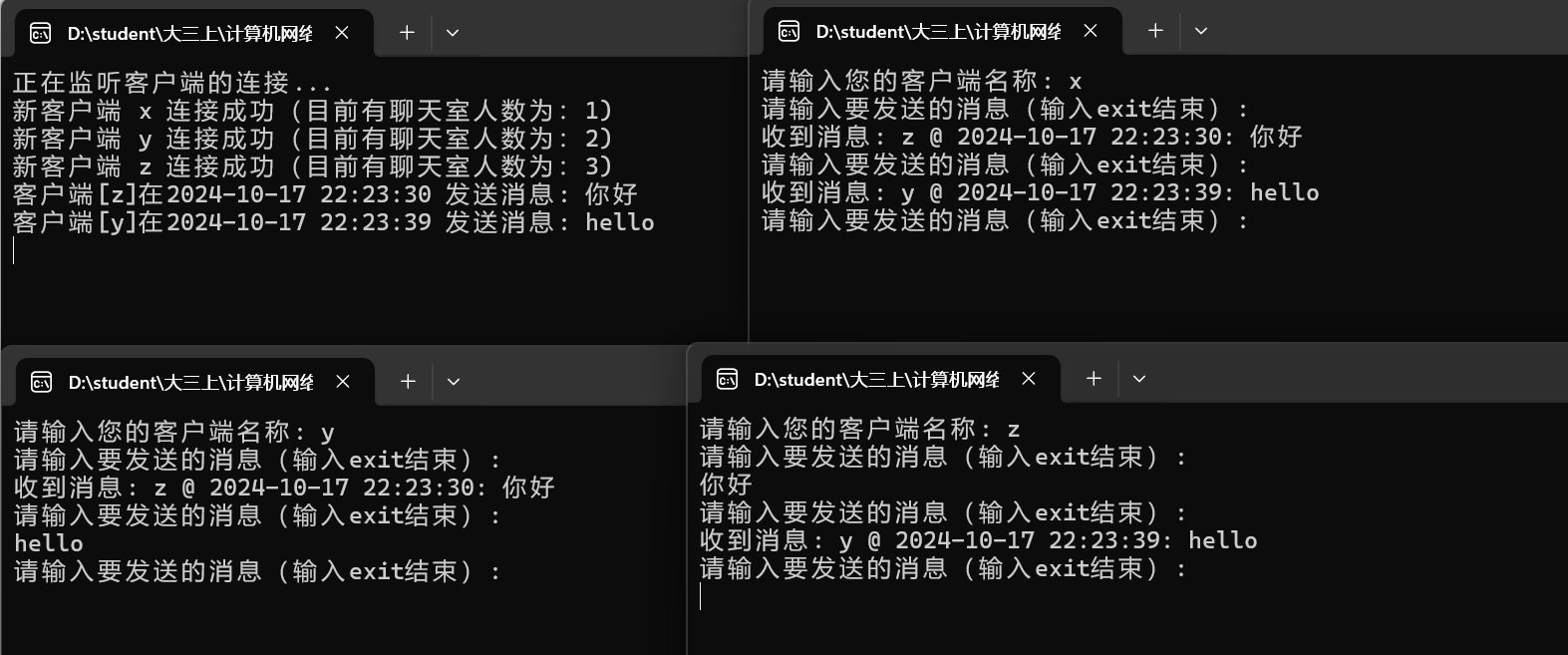
**cout << "收到消息: " << recvBuf << endl;**

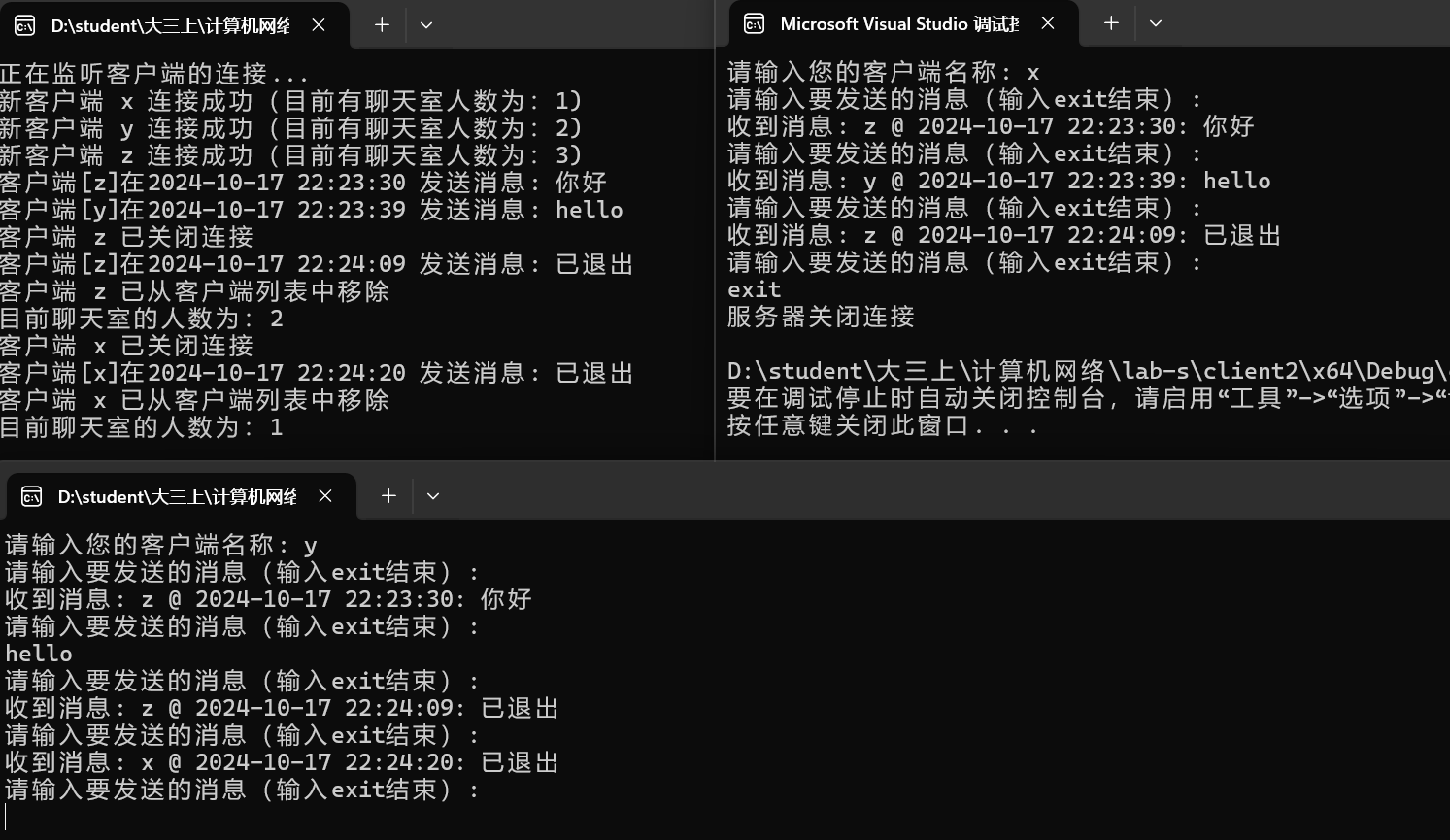
**cout << "请输入要发送的消息（输入exit结束）: " << endl;**

**}else {cout << "服务器关闭连接" << endl;keep = false;pthread\_join(receivethread, nullptr);closesocket(sock);break;}**

**4.运行结果**

本次实验创建了三个客户端与服务器进行连接，同时运行四个.cpp如下：

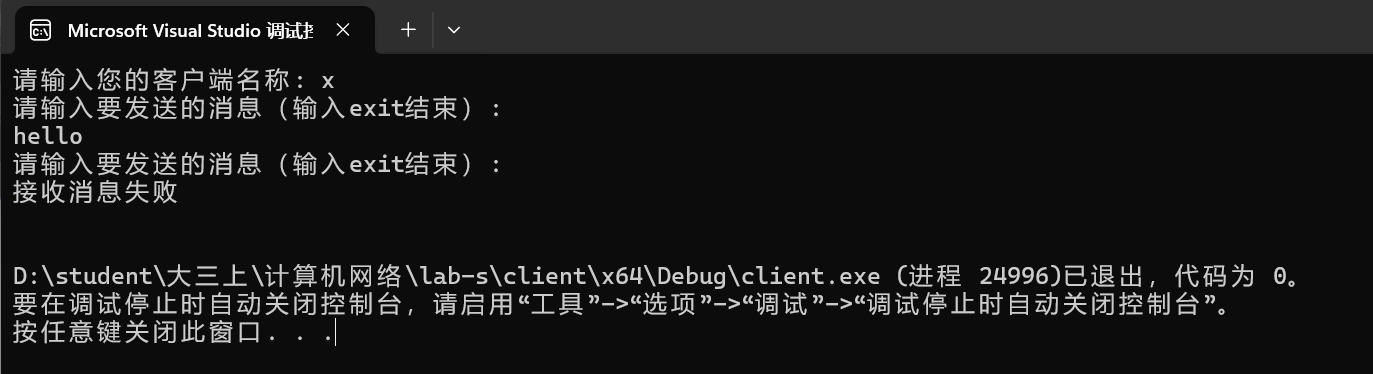
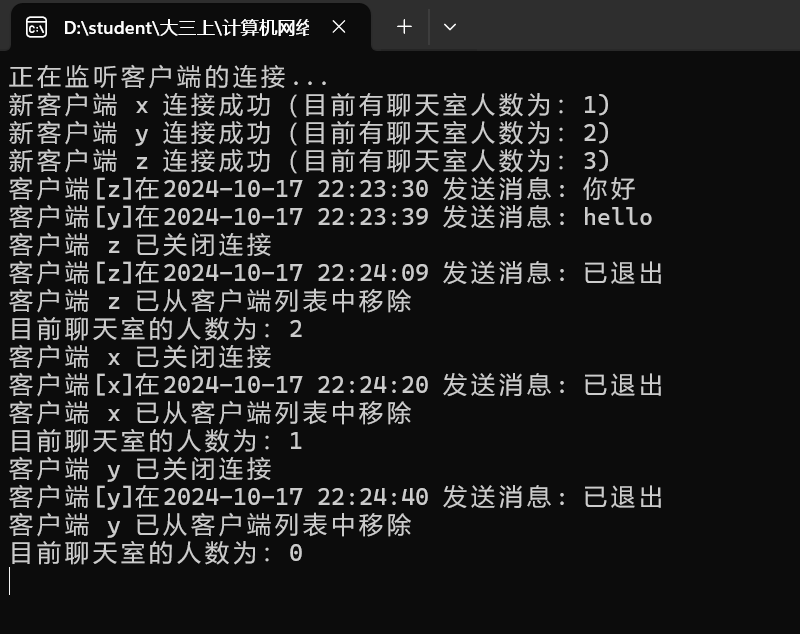




可以看到：

1.服务器监听客户端连接并通报实时聊天室人数；

1. 三个客户端成功与服务器进行连接；
2. 每个客户端可收到其他客户端发送的消息（包括客户端名称，时间以及内容）；
3. 输入exit，客户端正常退出；
4. 可在服务器端看到客户端退出以及移除，并显示聊天室剩下人数；
5. 可在其他客户端看到已退出的客户端发送的退出消息；
6. 服务端退出直接关闭控制台窗口；
7. 如果服务器端意外关闭，客户端会报错并在按enter后直接关闭而非卡死；



**5.总结与思考**

本次实验实现的聊天室较为简单，在实验中也遇到了许多问题，比如加入互斥锁导致死锁的问题等，个人认为可能还能从以下几点进行改进：

1. 互斥锁mutex：多线程可能导致数据竞争，使用互斥锁可保护对共享资源的访问；

2. 资源泄露：在线程执行过程中发生异常或错误，可能会导致资源（如套接字和线程句柄）未被正确释放；

3. 用户界面：是命令行界面，不够友好且功能可能有限，可使用GUI等进一步改进；

4. 消息存储和历史记录：目前不支持消息的历史记录和持久化存储，可以添加数据库来存储。