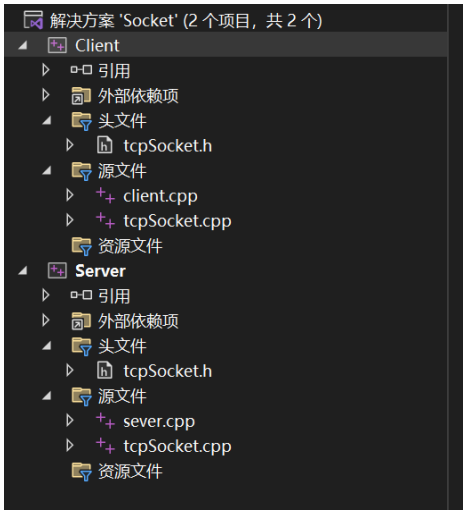
计算机学院 计算机网络 课程实验报告

<table>
<tr><td colspan="2">实验题目： Socket 套接字综合应用实验</td><td>学号：202200400053</td></tr>
<tr><td>日期：2024/6/13</td><td>班级：22.2</td><td>姓名：王宇涵</td></tr>
<tr><td colspan="3">实验小组成员：张祎乾、王宇涵、陈宇飞</td></tr>
<tr><td colspan="3">Email：1941497679@qq.com</td></tr>
<tr><td colspan="3">实验要求：
1、实验内容：编写类似微信群功能，允许多人聊天
(1) 必须实现功能（自由选择 TCP 或 UDP，编程语言）：
①服务器端：允许用户联网接入，接收用户键入内容，把内容传输给所有在群里的用户（客户端）。
②客户端：加入服务器，向服务器发送用户键入的内容，接收并显示服务器发送 的内容。
(2) 扩展内容：自由构思</td></tr>
<tr><td colspan="3">实验过程：
一 ： 建立项目
在进行本次实验时，我们首先在 Visual Studio Community 2022 中创建了一个解决方案 Socket。由于该环境默认每个项目只能运行一个程序（即一个 exe 文件），因此我们在同一解决方案下创建了两个独立的项目：Client 和 Server，分别用于客户端和服务端的实现。
 

我们在两个项目中都需要使用同一个头文件 tcpSocket.h。为了避免重复代码的低效和冗余，我们在 Socket 解决方案下创建了一个名为 tcpSocket 的文件夹，将头文件及其对应的 cpp 文件放入其中。接着，通过“添加现有项”的方式，将该头文件分别添加到 Client 和 Server 项目中，从而实现代码共享。
二 ： 编写头文件
以下是一些必要的初始化设置：
引入头文件 WinSock2.h
使用#pragma comment(lib, "Ws2_32.lib")链接 Ws2_32.lib 库
定义端口号为 8888：#define PORT 8888
定义宏#define err，方便错误处理
头文件中声明了以下函数：</td></tr>
</table>

bool init_Socket(); // 初始化网络库
bool close_Socket(); // 关闭网络库
SOCKET createServerSocket(); // 创建服务器 socket
SOCKET createClientSocket(const char* ip); // 创建客户端 socket

```h
#ifndef _TCPSOCKET_H_
#define _TCPSOCKET_H_
#include<WinSock2.h>        // windows平台的网络库头文件
#pragma comment(lib,"ws2_32.lib")
#include<stdbool.h>
#include<stdio.h>

#define PORT 8888
#define err(errMsg) printf("[line:%d]%s failed code %d",__LINE__,errMsg,WSAGetLastError());
// 打开网络库
bool init_Socket();

// 关闭网络库
bool close_Socket();

// 创建服务器socket
SOCKET createServerSocket();

// 创建客户端socket
SOCKET createClientSocket(const char* ip);

#endif // !_TCPSOCKET_H_
```

各函数实现的主要步骤如下：

初始化网络库

```cpp
bool init_Socket()
{
    // wsa  Windows socket async    windows异步套接字
    // parm1：请求的socket版本 parm2：传出参数
    WSADATA wsadata;
    if (WSAStartup(MAKEWORD(2, 2), &wsadata) != 0)
    {// 初始化网络库并判断是否成功
        err("WSAStartup");
        return false;
    }
    return true;
}
```

清理初始化设置

```cpp
bool close_Socket()
{
    if (WSACleanup() != 0)
    {// 清楚初始化设置并判断是否成功
        err("WSACleanup");
        return false;
    }
    return true;
}
```

建立服务端

```
SOCKET createServerSocket()
{
    // 1，创建空的socket     类似于买了一个新手机
    // parm1:af 地址协议ipv4,ipv6
    // parm2:type 传输协议类型 流式套接字 数据报
    // parm3:protocol 使用具体的某个传输协议
    SOCKET fd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);// ipv4,流式套接字,tcp协议
    if (fd == INVALID_SOCKET)
    {// socket创建失败
        err("socket");
        return INVALID_SOCKET;
    }

    // 2，给socket绑定ip地址和端口号     类似于给手机买了一个电话卡
    // parm1:socket需要绑定信息的socket
    // parm2:ip地址
    // parm3:地址的长度
    struct sockaddr_in addr;
    addr.sin_family = AF_INET;              // 和创建socket时的必须一样
    addr.sin_port = htons(PORT);            // [0,65535) 0~1024是保留端口
    addr.sin_addr.S_un.S_addr = inet_addr("127.0.0.1");// 绑定ip
    if (bind(fd, (sockaddr*)&addr, sizeof(addr)) == SOCKET_ERROR)
    {
        err("bind");
        return false;
    }

    // 3，监听电话       类似于等别人打电话
    listen(fd, 10);
    return fd;
}
```

根据参数中的服务端 IP 地址创建客户端并建立连接

```
SOCKET createClientSocket(const char* ip)
{
    // 1，创建空的socket     类似于买了一个新手机
    // parm1:af 地址协议ipv4,ipv6
    // parm2:type 传输协议类型 流式套接字 数据报
    // parm3:protocol 使用具体的某个传输协议
    SOCKET fd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);// ipv4,流式套接字,tcp协议
    if (fd == INVALID_SOCKET)
    {// socket创建失败
        err("socket");
        return INVALID_SOCKET;
    }

    // 2，与服务器建立连接
    // parm1:socket需要绑定信息的socket
    // parm2:ip地址
    // parm3:地址的长度
    struct sockaddr_in addr;
    addr.sin_family = AF_INET;              // 和创建socket时的必须一样
    addr.sin_port = htons(PORT);            // [0,65535) 0~1024是保留端口
    addr.sin_addr.S_un.S_addr = inet_addr(ip);// 绑定ip

    if (connect(fd, (sockaddr*)&addr, sizeof(addr)) == INVALID_SOCKET)
    {
        err("connect");
        return false;
    }

    return fd;
}
```

三 ： 服务器实现

服务端需要实现以下三个主要功能：

接收连接

监听消息

发送消息

由于在服务器等待连接或消息时无法执行其他任务，因此我们采用了多线程编程。以下是具体实现步骤：

创建服务端并执行任务

```
init_Socket();// 初始化网络库

serfd = createServerSocket();// 建立服务端
cli_num = 0;
cout << "服务器建立成功，以下是聊天室的聊天内容：" << endl;
ExeServer();
cout << "服务器已关闭" << endl;
```

执行服务端任务，主要使用了两个函数：ExeServer 和 ExeClient

ExeServer 函数：不断监听客户端连接请求，每当有新客户端连接时，为其创建一个线程，用于监听该客户端的信息

ExeClient 函数：每个线程对应一个客户端，负责监听该客户端的所有消息，并将其转发给聊天室内的其他客户端

```
void ExeServer()
{
    // 这个函数的作用就是不断监听客户端的连接，每多一个客户端的连接，就建立一个线程给该客户端，并接收其信息
    while (true)
    {
        SOCKET* new_clifd = new SOCKET(accept(serfd, NULL, NULL));
        if (INVALID_SOCKET == *new_clifd)// 判断客户端是否正常
        {
            err("accept");
        }
        clifds.push_back(new_clifd);
        thread* new_thread = new thread(ExeClient, cli_num);
        //ths.push_back(new_thread);
        cli_num++;
    }
}
```

```
void ExeClient(int index)
{
    char recvbuf[BUFSIZ] = { 0 };
    while (true)
    {
        // 服务端从客户端接收信息
        // recv从指定的socket接受消息
        memset(recvbuf, 0, BUFSIZ);
        if (recv(*clifds[index], recvbuf, BUFSIZ, 0) > 0)
        {
            printf("%s\n", recvbuf);
        }
        // 服务端向客户端发送消息
        for (int i = 0; i < cli_num; i++)
        {
            if (i != index)
            {
                if (send(*clifds[i], recvbuf, strlen(recvbuf), 0) == SOCKET_ERROR)
                {
                    err("send");
                }
            }
        }
    }
}
```

关闭所有应关闭的变量，结束程序

```
    // 关闭客户端和服务端，解除初始化设置
    closesocket(clifd);
    closesocket(serfd);

    close_Socket();

    printf("server-----end-----\n");
    system("pause");
    return 0;
}
```

四 ： 客户端实现

客户端的实现步骤如下：

初始化网络库并创建客户端，客户端创建时会根据传入的 IP 地址直接连接到服务端。由于聊天室内有多个用户，因此需要一个用户名来区分不同用户。

```
init_Socket();

SOCKET clifd = createClientSocket("127.0.0.1");
string name;
cout << "请输入您的用户名：" << endl;
cin >> name;
cout << "现在，你可以发送信息进行聊天了!" << endl;
```

创建两个线程：

```
// 可以和服务端进行通信了
thread send_th(sendMessage, clifd, name);
thread recv_th(recvMessage, clifd);
```

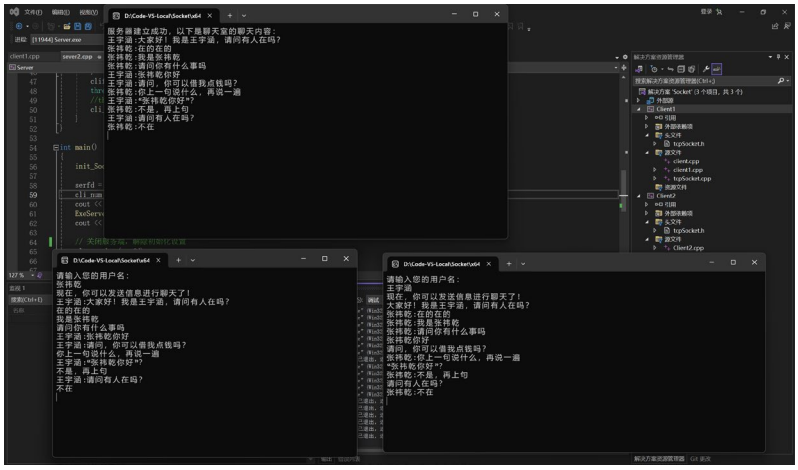
线程 1 负责等待客户消息，一旦消息编写完成，立即发送给服务端

线程 2 负责等待服务器的消息（即其他客户端发送的消息）

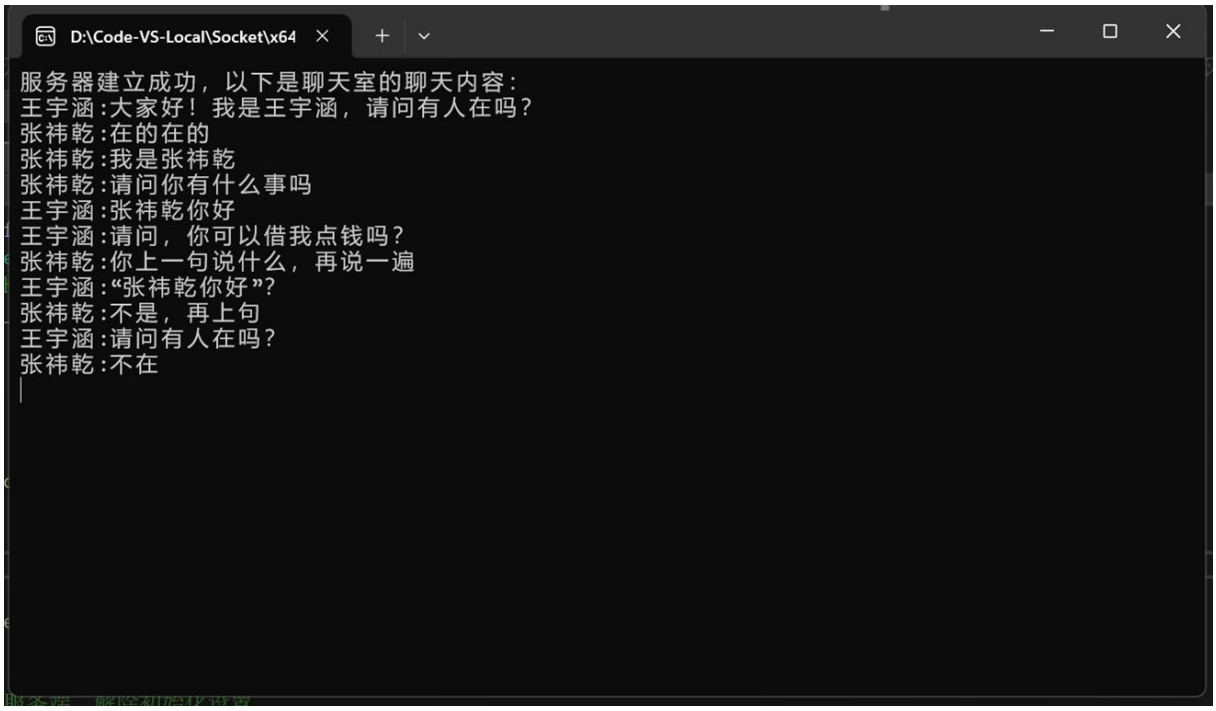
```
void sendMessage(SOCKET clifd, string name)
{
    string message, sendbuf;
    while (true)
    {
        // 发送消息
        cin >> message;
        sendbuf = name + ":" + message;
        if (send(clifd, sendbuf.c_str(), sendbuf.size(), 0) == SOCKET_ERROR)
        {
            err("send");
        }
    }
}
```

```cpp
void recvMessage(SOCKET clifd)
{
    char recvbuf[BUFSIZ] = { 0 };
    while (true)
    {
        // recv从指定的socket接受消息
        memset(recvbuf, 0, BUFSIZ);
        if (recv(clifd, recvbuf, BUFSIZ, 0) > 0)
        {
            printf("%s\n", recvbuf);
        }
    }
}
```

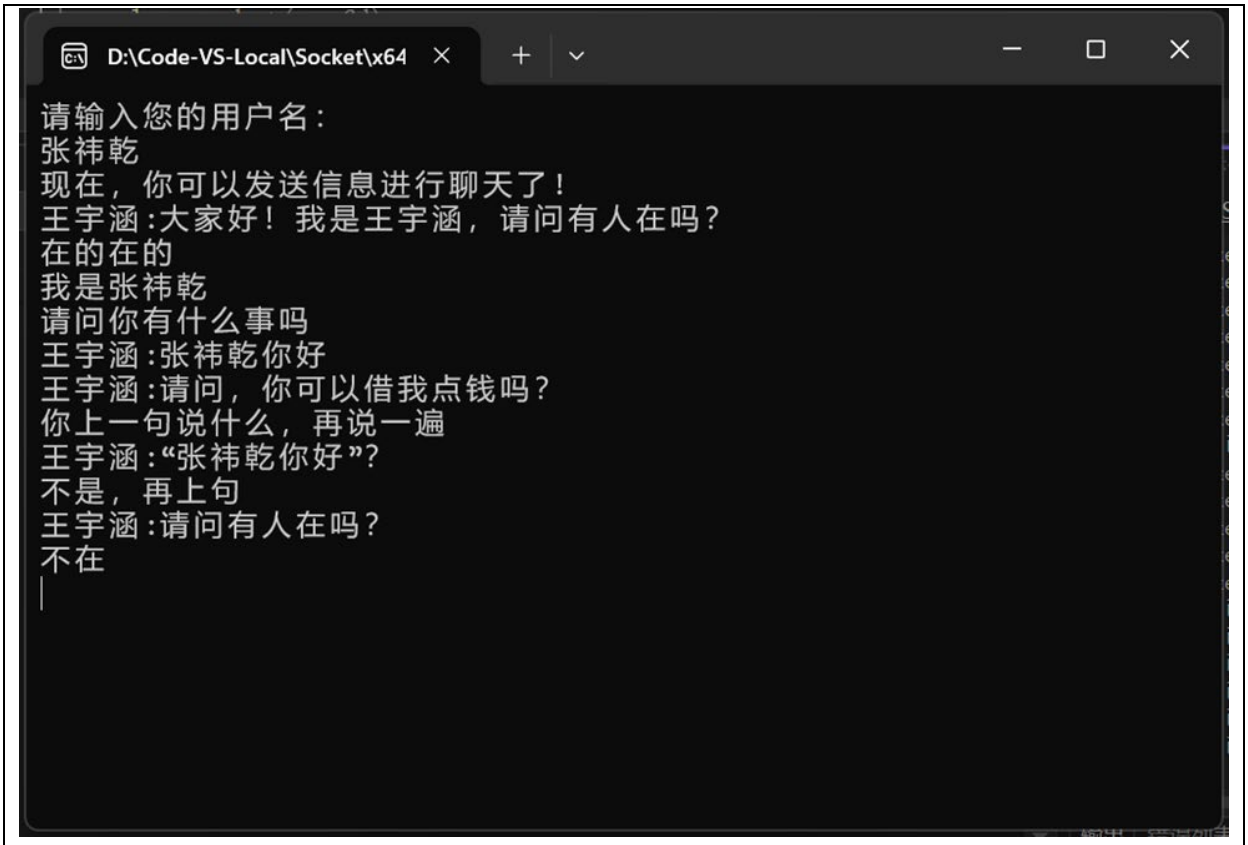
五 ： 结果展示



服务器端输出：



客户端 1： 张祎乾



客户端 2： 王宇涵

六 ： 补充功能(群聊功能)

在上述实验的基础上，我们增加了客户端群聊号的区分功能。每个客户端连接服务端后，可以选择加入特定的群聊，只有同一群聊内的用户才能相互接收到消息。

服务端修改：

在接收到连接请求后，进一步接收客户端选择的群聊信息，并将该客户端归入相应群聊中。向其他客户端发送信息时，需考虑群聊归属，只发送给同一群聊中的客户端。

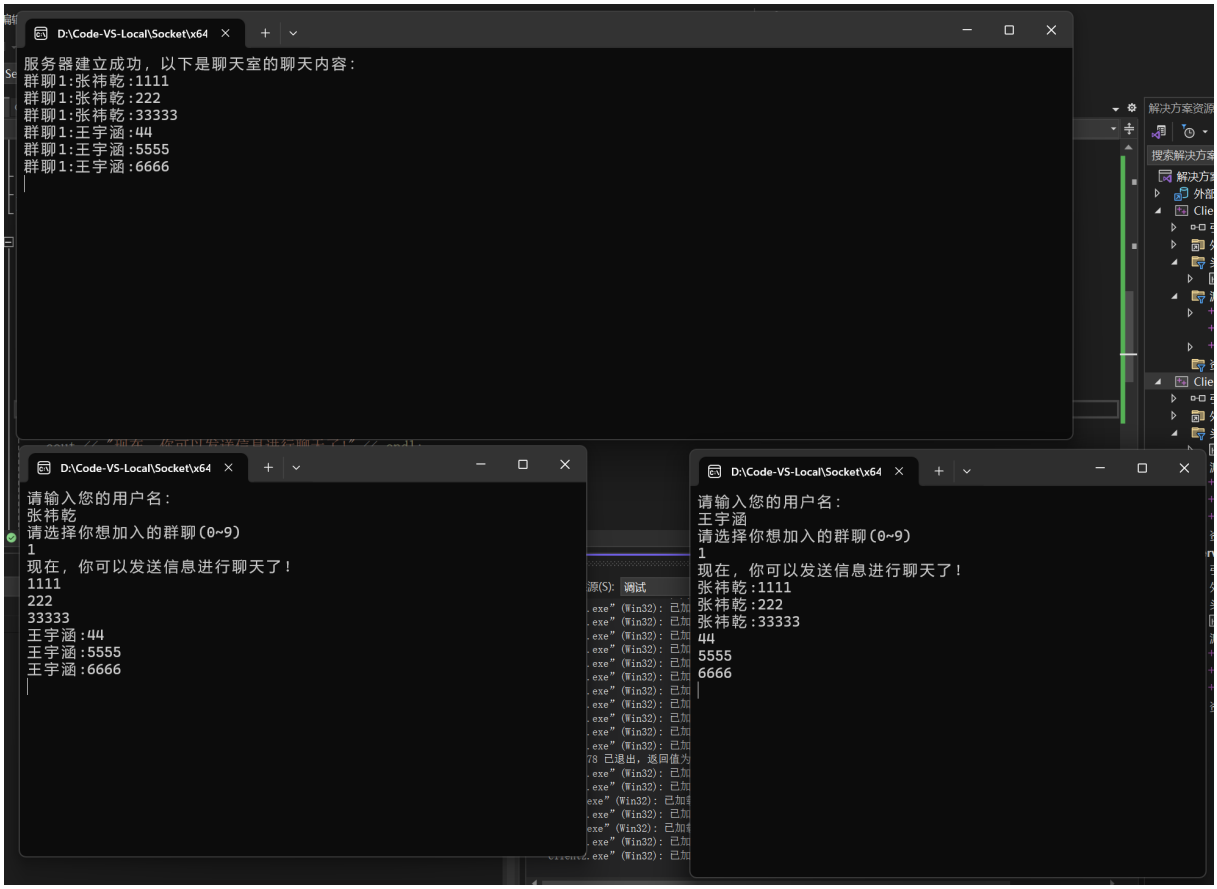
```
void ExeServer()
{
    // 这个函数的作用就是不断监听客户端的连接，每多一个客户端的连接，就建立一个线程给该客户端，并接收其信息
    while (true)
    {
        SOCKET* new_clifd = new SOCKET(accept(serfd, NULL, NULL));
        if (INVALID_SOCKET == *new_clifd)// 判断客户端是否正常
        {
            err("accept");
        }
        // 接收群聊信息
        char recvbuf[BUFSIZ] = { 0 };
        recv(*new_clifd, recvbuf, BUFSIZ, 0);
        int group_num = recvbuf[0] - '0';
        group_clifds[group_num].push_back(new_clifd);
        // 建立线程
        thread* new_thread = new thread(ExeClient, group_num, group_clifds[group_num].size() - 1);
        //ths.push_back(new_thread);
    }
}
```

客户端修改：

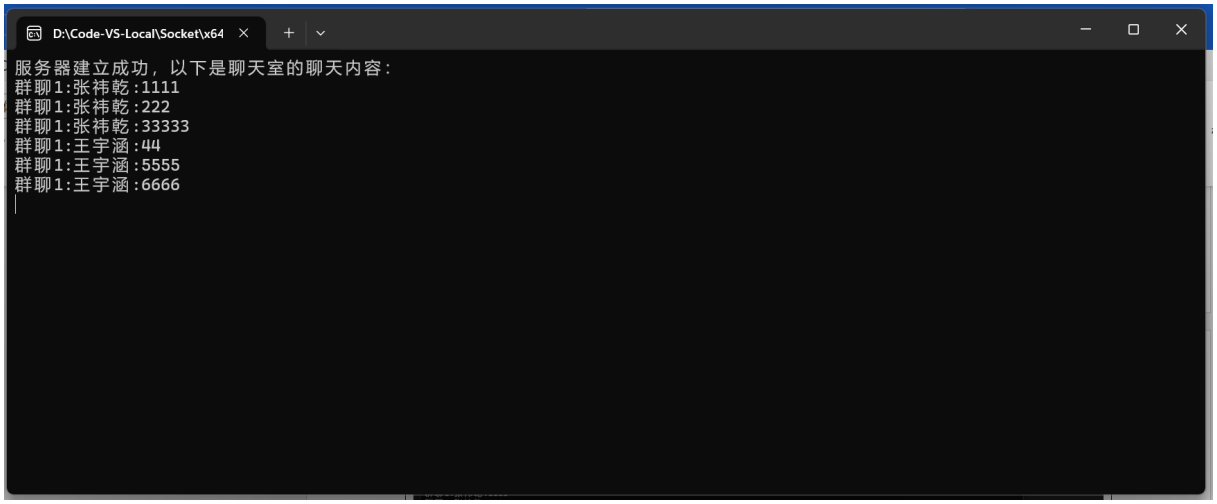
大部分代码保持不变，只是在连接服务端后，需要发送一个字符的数字，表示选择加入的群聊。

```
void ExeClient(int group_num, int index)
{
    char recvbuf[BUFSIZ] = { 0 };
    while (true)
    {
        // 服务端从客户端接收信息
        // recv从指定的socket接受消息
        memset(recvbuf, 0, BUFSIZ);
        if (recv(*group_clifds[group_num][index], recvbuf, BUFSIZ, 0) > 0)
        {
            printf("群聊%d:%s\n", group_num, recvbuf);
        }
        // 服务端向客户端发送消息
        for (int i = 0; i < group_clifds[group_num].size(); i++)
        {
            if (i != index)
            {
                if (send(*group_clifds[group_num][i], recvbuf, strlen(recvbuf), 0) == SOCKET_ERROR)
                {
                    err("send");
                }
            }
        }
    }
}
```

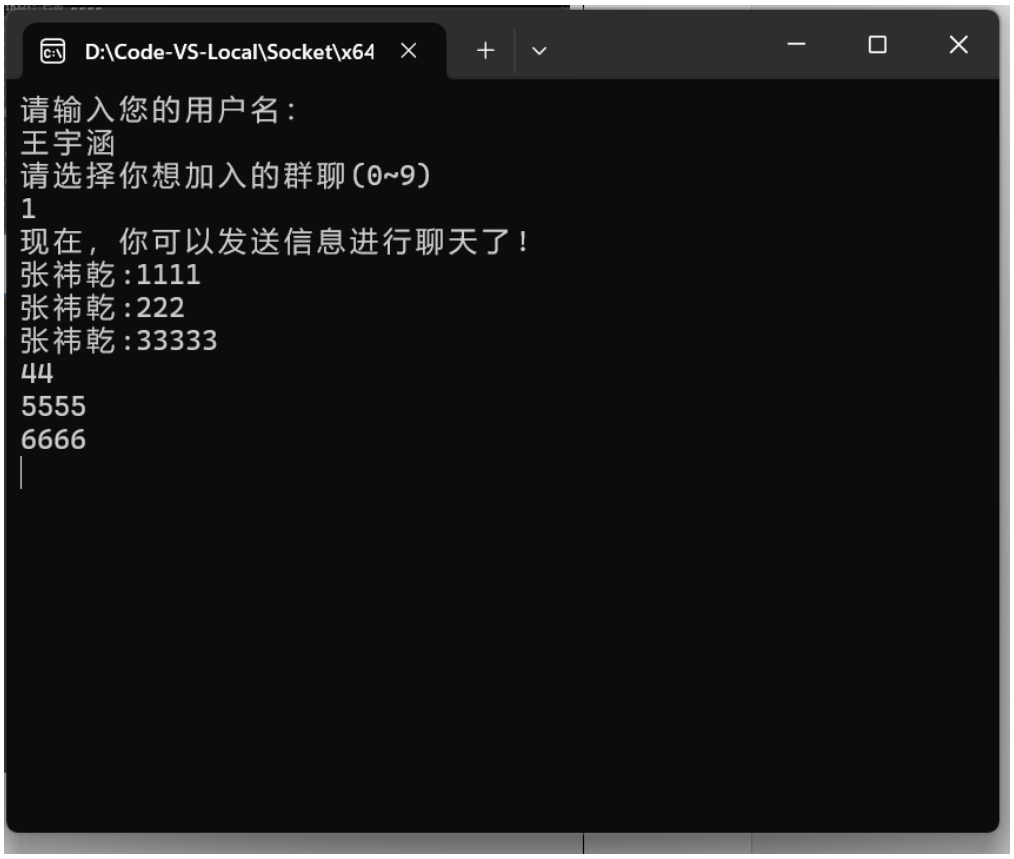
当客户端加入同一群聊时：


D:\Code-VS-Local\Socket\x64
服务器建立成功，以下是聊天室的聊天内容：
群聊1:张祎乾:1111
群聊1:张祎乾:222
群聊1:张祎乾:33333
群聊1:王宇涵:44
群聊1:王宇涵:5555
群聊1:王宇涵:6666
D:\Code-VS-Local\Socket\x64
请输入您的用户名：
张祎乾
请选择你想加入的群聊(0~9)
1
现在，你可以发送信息进行聊天了！
1111
222
33333
王宇涵:44
王宇涵:5555
王宇涵:6666
D:\Code-VS-Local\Socket\x64
请输入您的用户名：
王宇涵
请选择你想加入的群聊(0~9)
1
现在，你可以发送信息进行聊天了！
张祎乾:1111
张祎乾:222
张祎乾:33333
44
5555
6666

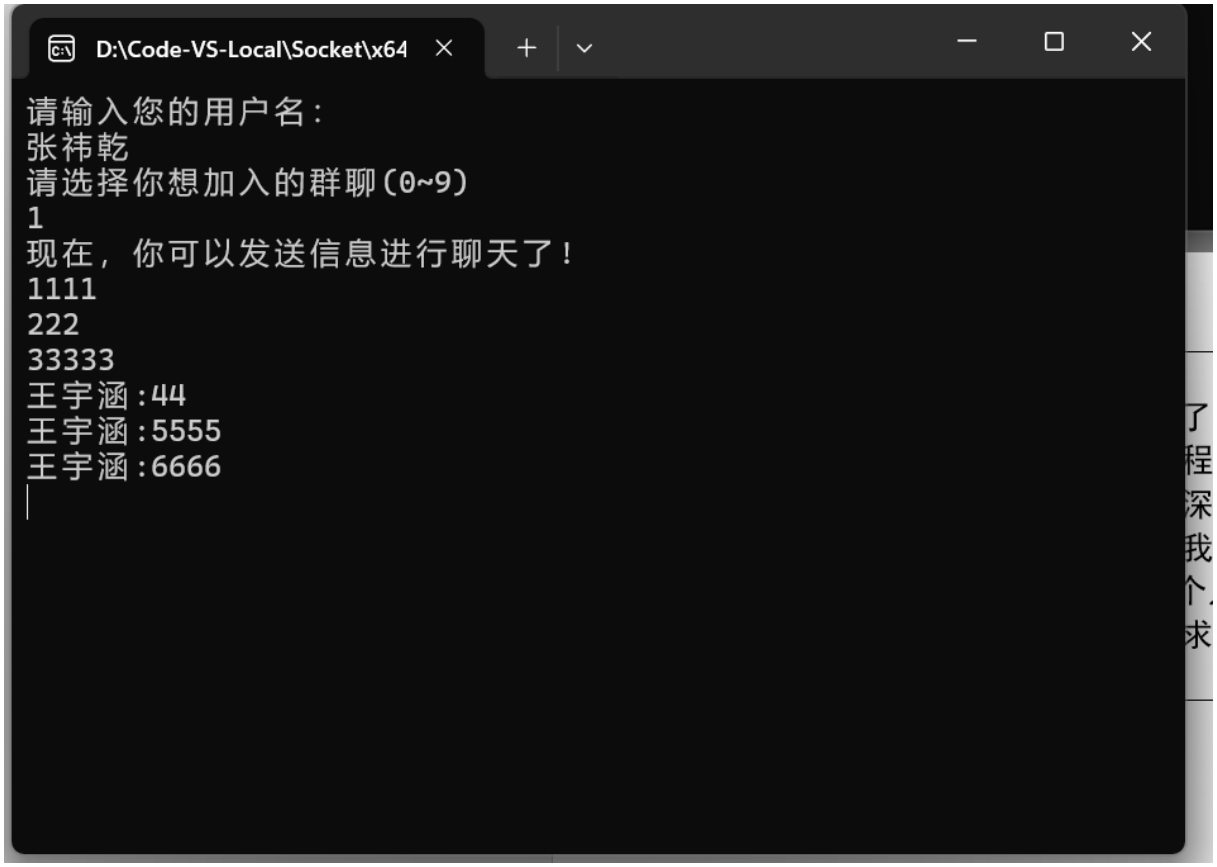
服务器端：


D:\Code-VS-Local\Socket\x64
服务器建立成功，以下是聊天室的聊天内容：
群聊1:张祎乾:1111
群聊1:张祎乾:222
群聊1:张祎乾:33333
群聊1:王宇涵:44
群聊1:王宇涵:5555
群聊1:王宇涵:6666

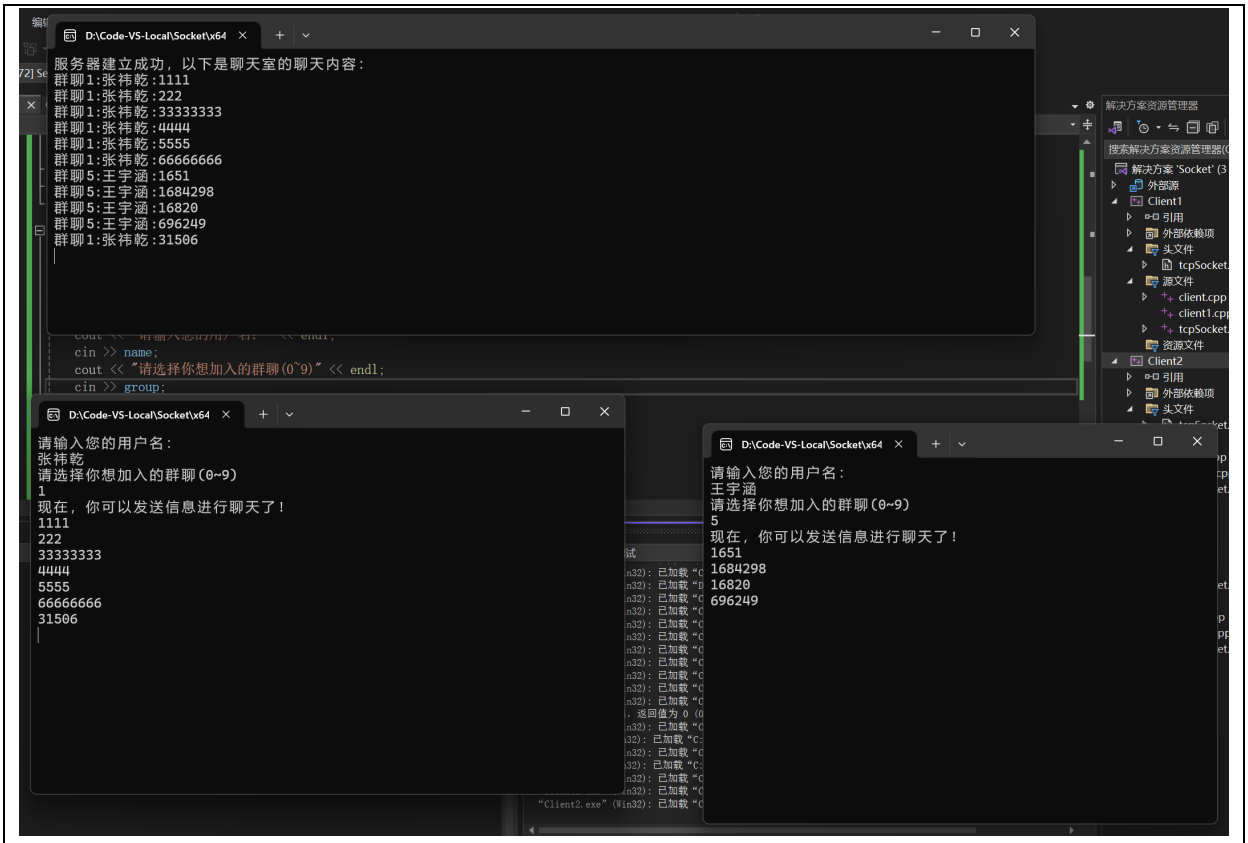
客户端 1：



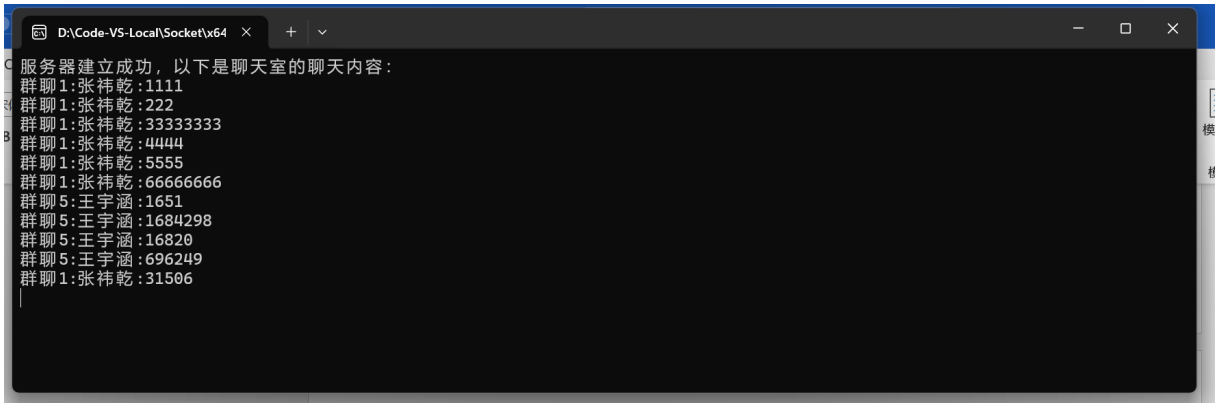
客户端 2：



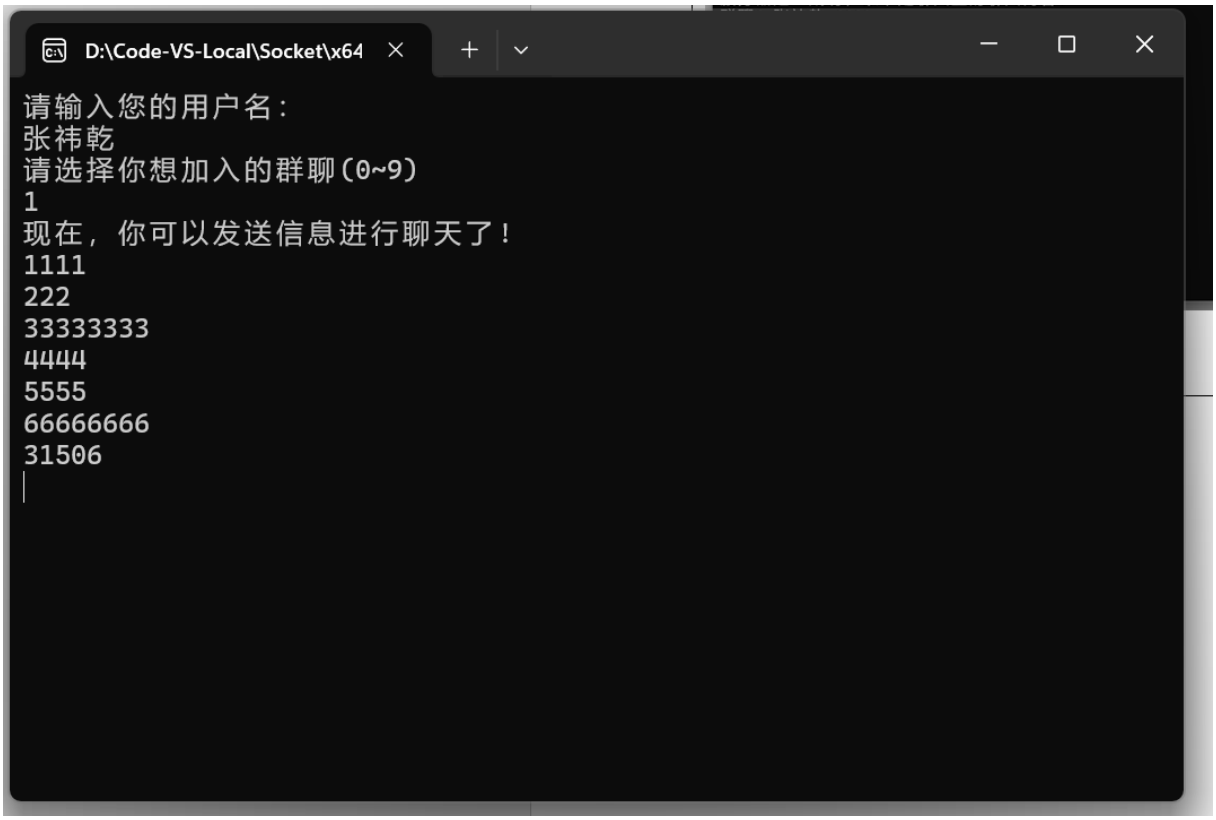
当客户端加入不同群聊时：


D:\Code-VS-Local\Socket\x64
服务器建立成功，以下是聊天室的聊天内容：
群聊1:张祎乾:1111
群聊1:张祎乾:222
群聊1:张祎乾:33333333
群聊1:张祎乾:4444
群聊1:张祎乾:5555
群聊1:张祎乾:66666666
群聊5:王宇涵:1651
群聊5:王宇涵:1684298
群聊5:王宇涵:16820
群聊5:王宇涵:696249
群聊1:张祎乾:31506
cin >> name;
cout << "请选择你想加入的群聊(0~9)" << endl;
cin >> group;
D:\Code-VS-Local\Socket\x64
请输入您的用户名：
张祎乾
请选择你想加入的群聊(0~9)
1
现在，你可以发送信息进行聊天了！
1111
222
33333333
4444
5555
66666666
31506
D:\Code-VS-Local\Socket\x64
请输入您的用户名：
王宇涵
请选择你想加入的群聊(0~9)
5
现在，你可以发送信息进行聊天了！
1651
1684298
16820
696249
解决方案资源管理器
Client1
Client2
client.cpp
client1.cpp

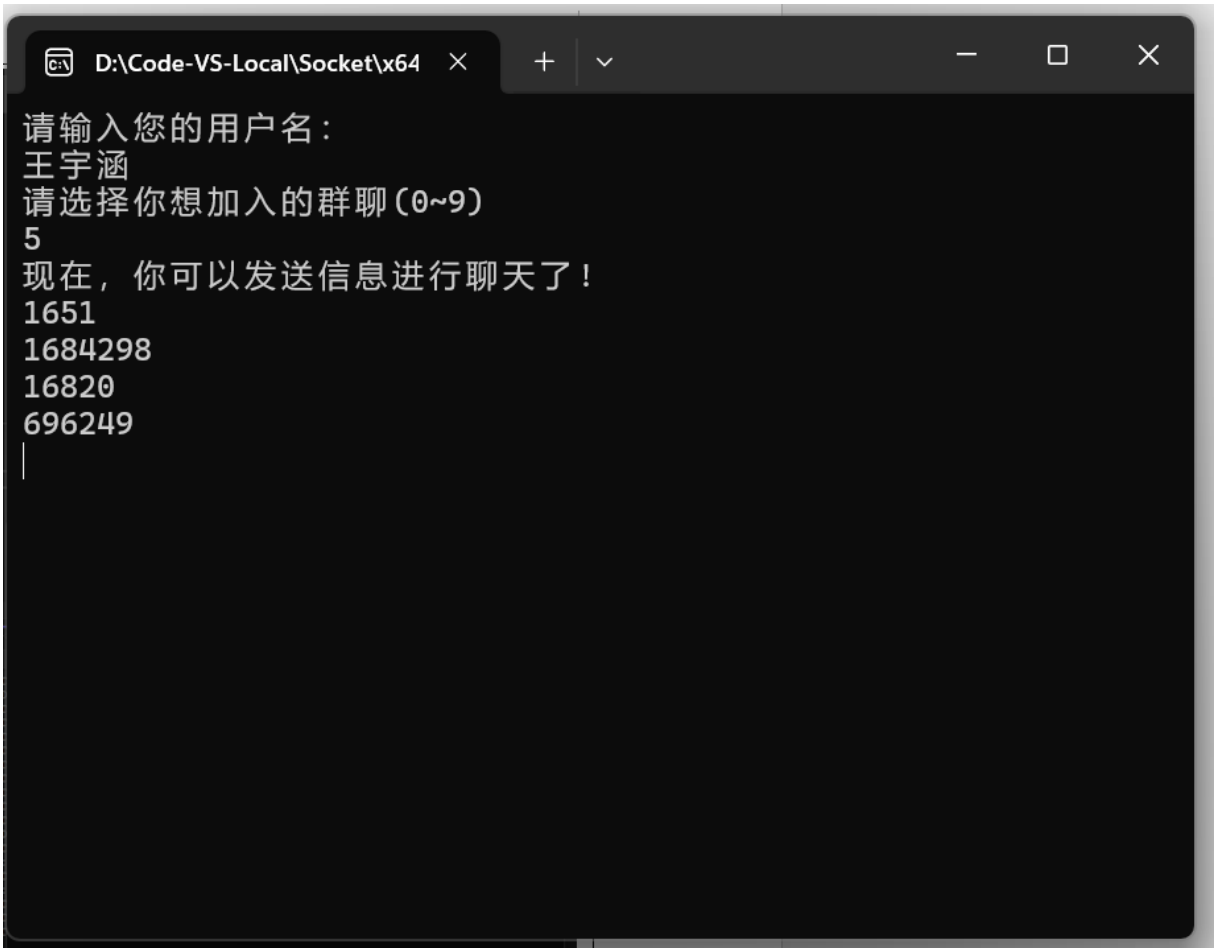
服务器端：


D:\Code-VS-Local\Socket\x64
服务器建立成功，以下是聊天室的聊天内容：
群聊1:张祎乾:1111
群聊1:张祎乾:222
群聊1:张祎乾:33333333
群聊1:张祎乾:4444
群聊1:张祎乾:5555
群聊1:张祎乾:66666666
群聊5:王宇涵:1651
群聊5:王宇涵:1684298
群聊5:王宇涵:16820
群聊5:王宇涵:696249
群聊1:张祎乾:31506

客户端 1：


D:\Code-VS-Local\Socket\x64
请输入您的用户名：
张祎乾
请选择你想加入的群聊(0~9)
1
现在，你可以发送信息进行聊天了！
1111
222
33333333
4444
5555
66666666
31506

客户端 2：


D:\Code-VS-Local\Socket\x64
请输入您的用户名：
王宇涵
请选择你想加入的群聊(0~9)
5
现在，你可以发送信息进行聊天了！
1651
1684298
16820
696249

实验体会：

通过本次实验，我们完成了基本功能的实现，对网络编程有了更深入的理解。实验不仅提高了我们的编程能力，还增强了对计算机网络相关知识的应用能力。我们进一步理解了网络编程和 Socket 套接字的应用。此外，我们还进行了多线程编程练习，因为单线程服务器无法同时监听多个客户端消息和连接请求。通过多线程学习，我们对进程和线程有了更深入的理解和区分。