计算机网络 第一次实验

> 姓名：管昀玫
>
> 学号：2013750
>
> 专业：计算机科学与技术

利用Socket，设计和编写一个聊天程序

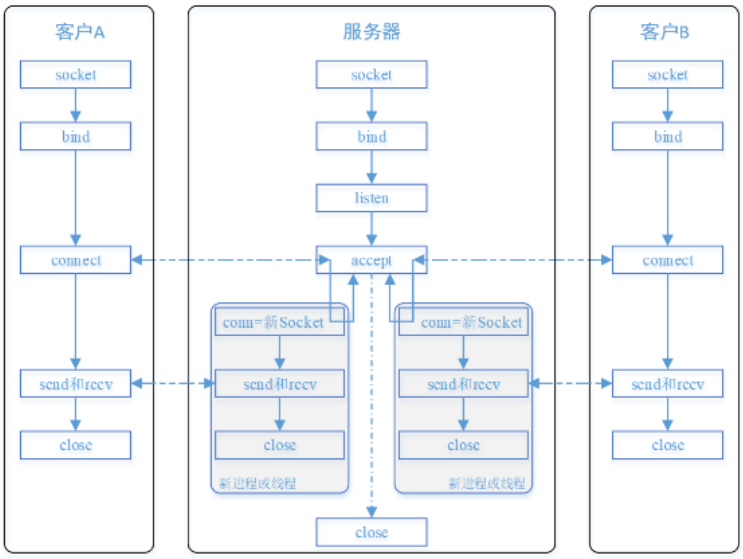
1. 使用流式Socket，设计一个两人聊天协议，要求聊天信息带有时间标签。请完整地说明交互消息的类型、语法、语义、时序等具体的消息处理方式。
2. 对聊天程序进行设计。给出模块划分说明、模块的功能和模块的流程图。
3. 在Windows系统下，利用C/C++对设计的程序进行实现。程序界面可以采用命令行方式，但需要给出使用方法。编写程序时，只能使用基本的Socket函数，不允许使用对socket封装后的类或架构。
4. 对实现的程序进行测试。
5. 撰写实验报告，并将实验报告和源码提交至本网站。

两人聊天

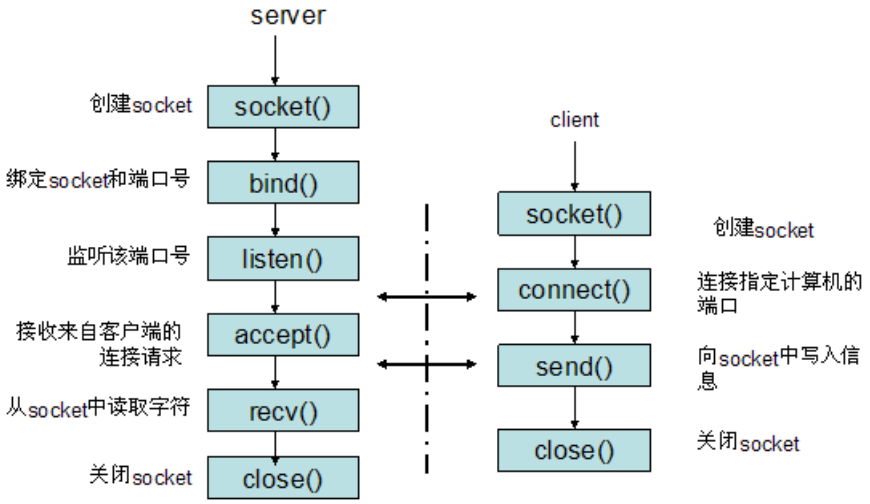
协议设计

`socket`又称套接字，是网络应用层与传输层的编程接口，根据传输层协议的不同，可分为`UDP Socket`和`TCP Socket`，本次实验采用`TCP Socket`。

流式套接字（SOCK_STREAM）：该类套接字提供了面向连接的、可靠的、数据无错并且无重复的数据发送服务，而且发送的数据时按顺序被接受的。所有利用该套接字进行传递的数据均被视为连续的字节流且无长度限制。这对数据的稳定性、正确性和发送/接受顺序要求严格的应用十分适用，TCP协议使用该类接口，但其对线路的占用率相对比较高。流式套接字的实现屡见不鲜，如远程登录（TELNET）、文件传输协议（FTP）等使用了流式套接字。



一个通用的`TCP Socket`连接过程如下所示：



会话的流程大致如下：

- 服务器和客户端都需要用`WSAStartup()`函数来启动；服务器端先创建套接字`socket()`并使用`bind()`将其与ip地址和端口绑定，之后就开始循环监听。服务器端使用`listen()`使`socket`进入监听状态，并设置等待队列最大长度为5；使用`accept()`函数接受新来的请求，并使用一个`socket`存放新创建的`socket`。
- 客户端也需要创建套接字并绑定服务器信息，再使用`connect()`连接远程服务器。客户端使用`send()`函数发送数据，并判断是否是`quit`信息；
- 服务器端和客户端都使用`CreateThread()`创建一个线程专门用来接收消息，并使用`send()`和`recv()`来发送和接受消息，期间开始通话；一旦判断为退出指令，则使用`CloseHandle()`关闭线程，使用`closesocket()`关闭`socket`，最终使用`WSACleanup()`释放Socket DLL资源。

消息信息

- 发送方：将发送方的消息进行封装，在消息前加上固定长度的发送时间信息(使用`time(0)`)，其中时间信息长度固定为25。
- 接收方：收到消息以后，将消息拆分成两部分，前一部分是时间信息，后一部分是消息内容。显示时先显示时间，在新的一行显示消息内容。（这里在处理收到的消息时利用的C++中`cout`的特性，可以直接将收到的字符串按照预计的格式进行输出，所以这里直接调用了`cout`）同时接收方会对传递过来的消息进行处理，如果发现收到的消息后半部分是`quit`，则关闭当前会话。
- 消息类型：string类型
- 语法：发送方输入消息内容；接收方显示：发送方信息 + 时间 + 消息内容
- 语义：
  - 输入IP：若是server，则获取本地IP；若是client，则获得目的IP
  - 输入聊天内容：双方发送和接收消息内容
  - quit：结束本次对话

程序设计

本次聊天程序为两人聊天程序，其中两个人在聊天时分饰两个角色：一个作为Server，一个作为Client。其中Server需要长时间在线，接收来自Client发送的聊天请求。

由此，本程序大致分为三个部分：Server、Client、main，其中main是程序的入口，负责聊天程序的启动，以及选择自身作为Server还是Client。

Main

main的部分比较简单，只需要获取命令行输入的参数，根据参数选择对应Server或者Client部分的启动即可。

Server

Server分为两部分代码，一个是Server.h，用于main进行include、提供接口，一个是Server.cpp，为程序的主要部分。接下来对Server.cpp部分实现思路进行阐述。

对于Server端，其功能顺序为：建立Socket - 绑定本地信息 - 启动监听 - 监听到请求之后进行accpet并分配新的socket用于负责对话 - 进行通话 - 对话结束后关闭Socket

- **建立socket**：使用WinSock2.h和WS2tcpip.h库，建立serverSocket，并检查是否创建成功。
- **绑定服务端信息**：按照Socket中对socket绑定的要求，将IP、port、family等信息输入后使用bind函数进行绑定（这里为了方便起见，端口号固定使用12260）
- **监听**：利用 while(true) 的形式进行阻塞式的监听，使用的函数就是listen函数，并设置一个队列长度是5
- **创建新socket并进行通信**：利用accept函数，创建一个clientSocket用来接收。在成功接收之后向客户端发送一条信息，告诉客户端已经成功建立链接；并在Server的命令行上输出一条显示连接成功的消息。如果接收失败，则回显error信息。

  连接成功之后创建一个新的线程，新线程只用来负责接收消息并显示，也就是线程中有一个 while(true) ，循环体内部就是 recv 和解析过后 cout 。主线程则是一直进行发送，在 while(true) 内用 send 函数。
- **会话结束**：在通讯过程中，如果有一方发送的消息是quit，则结束本次通话。

  结束通话后，Server继续释放刚刚建立的socket，并回到listen中继续监听。

Client

Client同样分成client.h和client.cpp，其中client.cpp为主要功能部分代码，接下来对client.cpp部分实现思路进行阐述。

对于客户端，其内容相比服务器端就简单一些，其功能顺序为：建立Socket - 设定本地的地址信息 - 设定远端的地址信息 - 发送请求 - 进行通话 - 对话结束后关闭Socket。

- **建立socket**：过程同Server中的建立socket步骤
- **设置本地地址信息**：同样将IP、port等进行输入，但是注意这里和server的区别是这里没有bind函数
- **设定远端的地址信息**：设置方式同本地，只是内容换成服务器端的地址信息
- **发送请求**：使用connect函数建立链接，建立成功则会接收到远端发来的信息并进行相应显示；建立失败回显error信息。
- **进行通信**：和Server端方式相同，创建一个新的线程进行接收和内容显示、本线程进行发送。
- **会话结束**：在通讯过程中，如果有一方发送的消息是quit，则结束本次通话。通话结束以后释放之前建立的两个socket，并结束本程序。

程序实现

详细代码见main.cpp, server.h, server.cpp, client.h, client.cpp，本处只针对核心代码进行讲解。

```
DWORD WINAPI recvMessagec(LPVOID) {
    while (1) {
        ::memset(CrecvBuf, 0, CBUF_SIZE);//清空接收数组缓冲区
```

```
        //recv(SOCKET s, char* buf, int len, int flags) flags一般设置为0
        if (recv(LocalhostSocket, CrecvBuf, CBUF_SIZE, 0) < 0) {//使用recv函数接受
接收信息
            cout << "Connect failed. Error code is " << SOCKET_ERROR << ".\n";
            system("pause");
        }
        else {
            if (isRecvQuit(CrecvBuf)) {//server quit
                cout << "Server has stopped this session, bye.\n";
                connected = false;
                break;
                ExitThread(1);
            }
            else
                cout << "From Server, at " << CrecvBuf << endl;//显示接收的消息
        }
    }
    return 0;
}
```

此为创建接收线程后在线程内使用的接收消息的函数。首先清空接收数组的缓冲区，然后使用`recv()`函数接受信息，并根据返回值判断是否是quit。`recv()`函数如果执行失败，会返回SOCKET_ERROR；如果在等待协议接收数据时网络中断了，那么它返回0。正常工作时，会把接收到的消息存储在CrecvBuf中，并打印出来。打印信息包括消息来源、时间信息、具体消息内容。

```
bool isQuit(char* a, unsigned int len) {
    if (len == 4 && a[0] == 'q' && a[1] == 'u' && a[2] == 'i' && a[3] == 't')
        return true;
    else
        return false;
}
bool isRecvQuit(char* a) {
    int len = 0;
    while (a[len] != '\0')
        len++;
    if (len == 29 && a[25] == 'q' && a[26] == 'u' && a[27] == 'i' && a[28] ==
't')
        return true;
    else
        return false;
}
```

这两个函数分别用于判断是否主动结束通信或是被动结束通信。

```
int getPort() {
    cout << "Please input your port to connect with others which is in
range(12000,16000).\n";
    cout << "If you wanna use default port, please input 0\n";
    u_short tempPort;
    cin >> tempPort;
    while (tempPort < 12000 || tempPort>16000) {
        if (tempPort == 0) {
            return ServerPort;
        }
```

```
        cout << "Invalid input. Please input in range(12000,16000) or 0.\n";
        cin >> tempPort;
    }
    return tempPort;
}
```

此函数用于获取端口号。为了方便，本次实验统一使用端口12260。

```
void getServerIP(char* p) {
    cout << "Please use 'ipconfig' to get your ip and input\n";
    char ip[16];
    cin.getline(ip, sizeof(ip));
    int index = 0;
    while (ip[index] != '\0') {
        p[index] = ip[index];
        index++;
    }
}
```

此函数用于得到serverIP。需要手动输入。

```
    // 创建socket，参数顺序:IPv4，流式套接字，指定协议
    ServerSocket = socket(AF_INET, SOCK_STREAM, 0);
    // 初始化地址信息
    // inet_addr用来将IP地址转换成网络字节序
    ServerAddr.sin_family = AF_INET;
    getServerIP(ServerIP);
    inet_pton(AF_INET, ServerIP, &ServerAddr.sin_addr.S_un.S_addr);//将IP地址从字符串格式转换成网络地址格式，支持Ipv4和Ipv6.
    //inet_pton src:是个指针，指向保存IP地址字符串形式的字符串。dst:指向存放网络地址的结构体的首地址
    ServerAddr.sin_port = htons(ServerPort);//将一个无符号短整型的主机数值转换为网络 字节顺序，即大尾顺序(big-endian)
    // 绑定socket和地址
    bind(ServerSocket, (SOCKADDR*)&ServerAddr, sizeof(SOCKADDR));
```

此段为server端绑定本地信息，具体作用见代码块注释。

```
// 套接字绑定本地地址信息
    LocahhostAddr.sin_family = AF_INET;
    LocahhostAddr.sin_port = htons(ClientPort);
    inet_pton(AF_INET, LocalIP, &LocahhostAddr.sin_addr.S_un.S_addr);

    // 服务器（目标地址信息）
    RemoteAddr.sin_family = AF_INET;
    cout << "Which IP would you wanna connect:\n";
    char* DstIP = getIP();
    cin.get();
    RemoteAddr.sin_port = htons(12260);
    inet_pton(AF_INET, DstIP, &RemoteAddr.sin_addr.S_un.S_addr);//将一个无符号短整型的主机数值转换为网络 字节顺序，即大尾顺序(big-endian)
```

client端的设置本地信息和远端服务器信息。

```
HANDLE recv_thread = CreateThread(NULL, NULL, recvMessage, NULL, 0, NULL);
```

创建一个线程，专门用作接受消息。

```
                if (!Sconnected)
                        break;
                    memset(SinputBuf, '\0', SBUF_SIZE);
                    cin.getline(SinputBuf, sizeof(SinputBuf));

                    // 获取当前时间
                    time_t now = time(0);
                    // 转换成字符串形式
                    char* dt = ctime(&now);

                    memset(SsendBuf, 0, SBUF_SIZE);
                    strcat(SsendBuf, dt);
                    strcat(SsendBuf, SinputBuf);

                    send(ClientSocket, SsendBuf, sizeof(SsendBuf), 0);
                    if (isQuit(SinputBuf, getLen(SinputBuf))) {
                        cout << "You stopped this session.\n";
                        Sconnected = false;
                        CloseHandle(recv_thread);
                        closesocket(ClientSocket);
                        break;
                    }
```

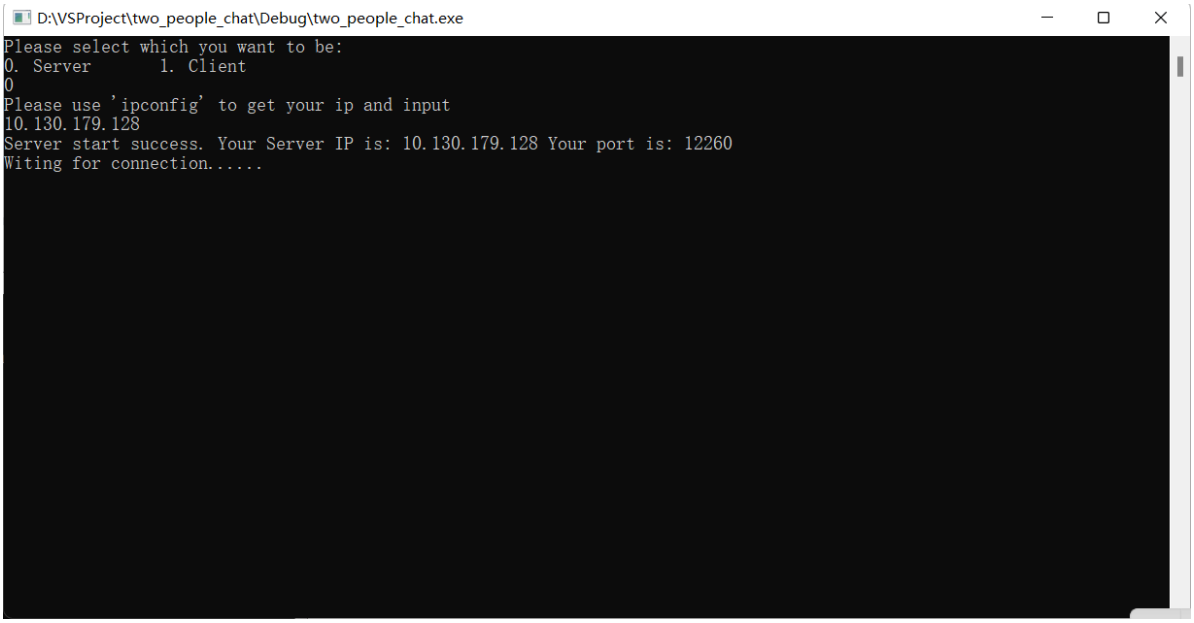
此函数用于输入消息内容，并将处理消息（加上时间信息），最后使用`send()`函数发送给接收方。此函数在循环体内运行。期间需要判断输入内容是否是`quit`，如果是，则需要将`Sconnected`置为false，并关闭线程，使用`closesocket()`函数关闭socket。

使用方法

1. 运行程序后，进入到选择页面。
2. 输入0以选择成为服务端
    1. 输入本地IP
    2. 开启以后等待连接，连接成功以后直接在命令行输入想要发送的内容，按下回车发送信息
    3. 输入quit以结束本次会话
3. 输入1以选择成为客户端
    1. 输入想要连接的IP
    2. 等待连接成功后直接在命令行输入想要发送的内容，按下回车发送信息
    3. 输入quit以结束本次会话

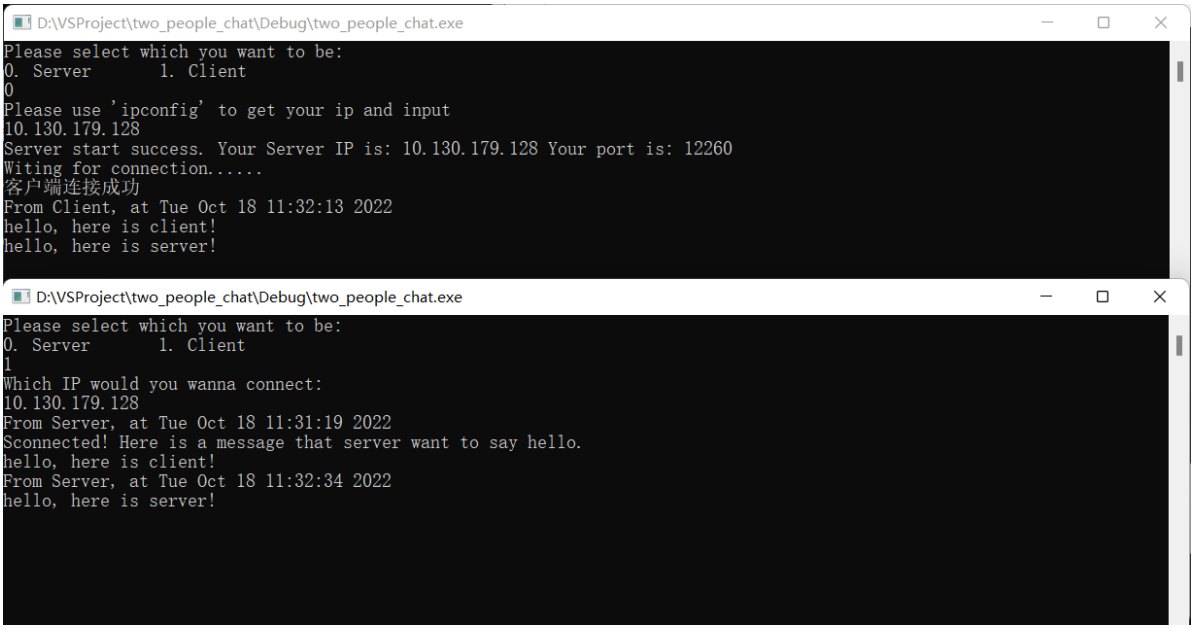
实验效果测试

等待连接页面

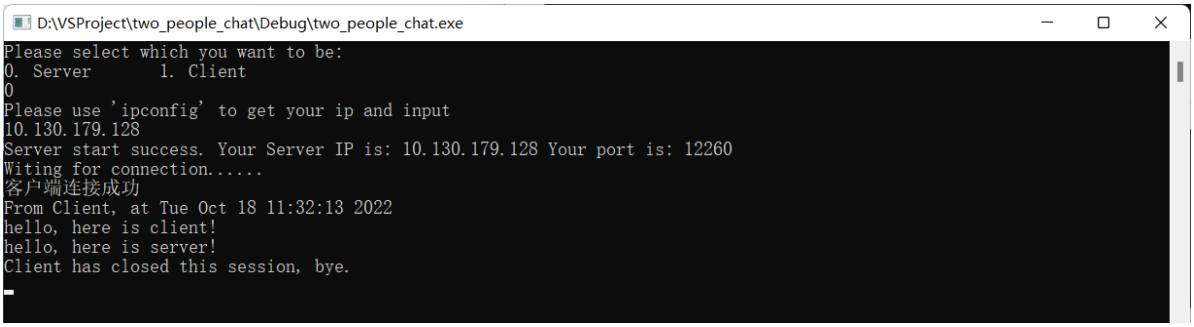


连接成功

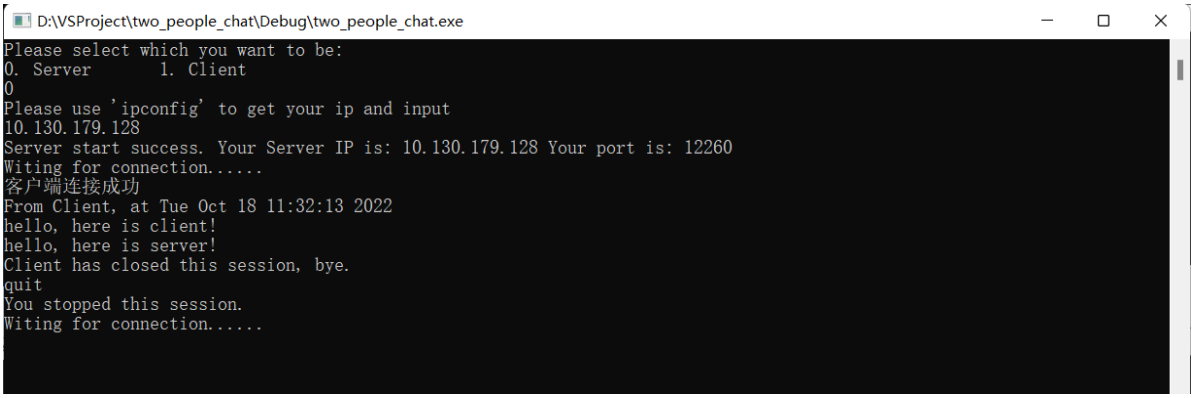
发送与接受消息



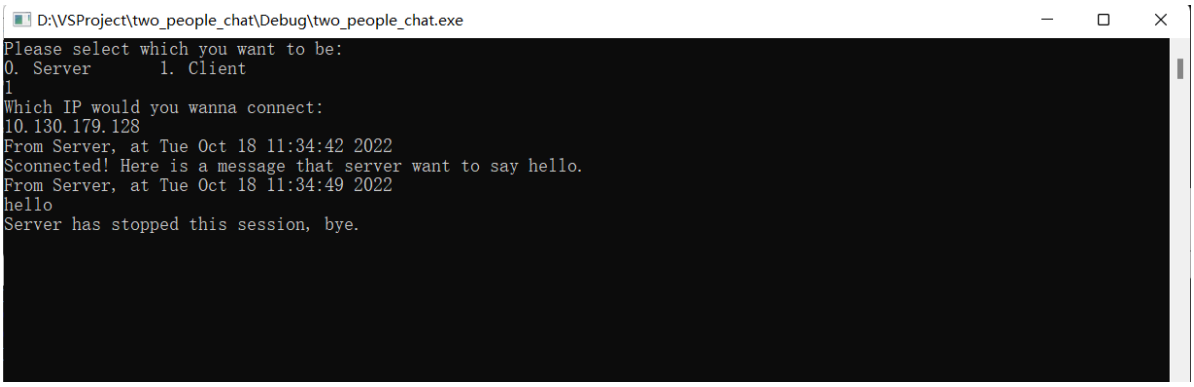
服务器端收到客户端quit消息



客户端quit之后，服务器输入quit

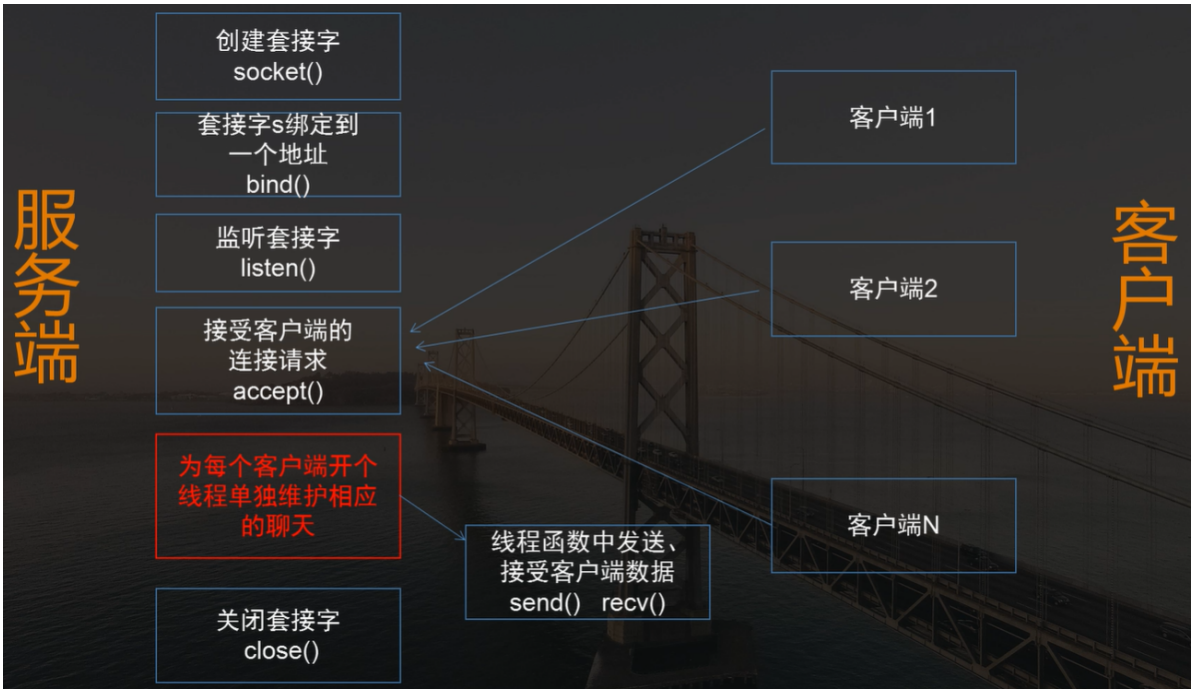


连接过程中，客户端收到服务器端的quit



多人聊天

协议设计



注意：以下所说的用户昵称长度，均是将int型的长度强制类型转换成char存入字符串中，在进行对应的检测后，会将其转换回int以进行名称读取；所有的时间信息固定长度为25；

多人聊天仍使用流式套接字。有关流式套接字内容见两人聊天处。

- **退出**：用户一方输入 quit ，Client会将单个字符q、当前用户名称的长度、名称、quit字样封装成一个字符串发送到Server，Server检查第一个字符是'q'，之后获取用户名称，并向剩下所有的用户发送一条下线提示，转发的报文格式和之前Client发来的是一样的；Client在收到提示之后，检查第一个字符为'q'之后解析用户名称并显示显现提示；
- **用户登录**：当一个用户输入聊天室IP后，用户会输入自己的用户名，在连接成功后，客户端会向Server发送一个报文，其格式为：单个字符'n'+用户名称长度+用户名称。Server收到消息后会检查第一个字符是否为'n'，若是，则读出第二位，之后根据长度依次读出名称并存放到对应保存的用户信息的数组中。
- **获取用户列表**：用户输入 `getUsrList` ，Client会直接将这个字符串发送给Server，Server在检查到第一个字符是'g'的时候将当前建立连接的用户列表进行统计并拼成一个字符串，每部分都由：用户名长度+名称组成，并将字符'u'拼接在首位；Client收到报文检查首位是'u'时，每次获取一组用户名长度+名称进行显示；
- **发送消息(消息格式)**：用户输入格式为 `username:message`，Client端会将内容拆分成username和message两段，然后获取当前时间以及当前要发送的用户信息，将这4部分按照时间、目标用户名称长度+目标用户名称、当前用户名称长度+当前用户名称、要发送的信息的顺序拼接成一个报文并发送；Server在收到报文后会进行解析，匹配已连接的用户中是否有昵称相同的（假设所有用户的昵称不相同）或发送的是公共消息：
  - 若发送的是公共消息(即username字段填写为 `open`)，则在Server出公屏显示出用户名、时间和消息内容。
  - 若匹配上则向目标用户发送：时间、当前用户名称长度+当前用户名称（也就是消息来源用户）、要发送的信息拼接成一个报文发送；收到消息的Client解析收到的消息，将时间、来源、消息内容依次输出
  - 若匹配失败则向消息来源的用户返回一条error提示信息，收到错误提示的Client只需要检查前5个字符是否为“Input”即可

程序设计

本次聊天程序为多人分组聊天程序，其中有Server端和Client端之分。其中Server需要长时间在线，接收消息并作为中转站，接收来自Client发送的聊天请求并进行转发。

由此，本程序大致分为三个部分：Server、Client、main，其中main是程序的入口，负责聊天程序的启动，以及选择自身作为Server还是Client

Main

main的部分比较简单，只需要获取命令行输入的参数，根据参数选择对应Server或者Client部分的启动即可。

Server

Server分为两部分代码，一个是Server.h，用于main进行include、提供接口，一个是Server.cpp，为程序的主要部分。接下来对Server.cpp部分实现思路进行阐述。

对于Server端，其功能顺序为：建立Socket - 绑定本地信息 - 启动监听 - 监听到请求之后进行accpet并分配新的socket用于负责转发 - 对话结束后关闭Socket

- **建立socket**：使用WinSock2.h和WS2tcpip.h库，建立serverSocket，并检查是否创建成功
- **绑定服务端信息**：按照Socket中对socket绑定的要求，将IP、port、family等信息输入后使用bind函数进行绑定（这里为了方便起见，端口号固定使用12260）
- **监听**：利用 while(true) 的形式进行阻塞式的监听，使用的函数就是listen函数，并设置一个队列长度是5
- **创建新socket、新线程并进行通讯**：
    - Server会提前创建好一个socket数组，数组长度和监听队列长度相同。
    - 创建一个单独的线程并利用accept函数，使用socket数组中的一个接收，并接收用户发来的信息进行名称的登记。在成功接收之后向客户端发送一条信息，告诉客户端已经成功建立链接；并在Server的命令行上输出一条显示连接成功和当前用户数量的消息。如果接收失败，则回显error信息。
    - 连接成功之后创建一个新的线程，新线程只用来负责接收消息并显示，也就是线程中有一个 while(true) ，循环体内部就是 recv 和解析过后对应各种行为。主线程则是一直进行监听，在 while(true) 内运行 listen 函数，监听远程连接是否到来。

Client

Client同样分成client.h和client.cpp，其中client.cpp为主要功能部分代码，接下来对client.cpp部分实现思路进行阐述。

对于客户端，其内容相比服务器端就简单一些，其功能顺序为：建立Socket - 设定本地的地址信息 - 设定远端的地址信息 - 发送请求 - 进行通话 - 对话结束后关闭Socket。

- **建立socket**：过程同Server中的建立socket步骤
- **设定本地地址信息**：同样将IP、port等进行输入，但是注意这里和server的区别是这里没有bind函数
- **设定远端的地址信息**：设置方式同本地，只是内容换成服务器端的地址信息
- **发送请求**：使用connect函数建立链接，建立成功则会向Server发送一条自身信息登记的消息，并接收到远端发来的提示信息并进行相应显示；建立失败回显error信息
- **进行通讯**：和Server端方式相同，创建一个新的线程进行接收和内容显示、本线程进行发送。
- **会话结束**：在通讯过程中，如果Client发送的消息是quit，则结束本次通话。通话结束以后释放之前建立的两个socket，并结束本程序。或者是当Server关闭，则Client这边会显示连接失败。

程序实现

本部分仅针对相对于两人聊天新增核心代码进行讲解。

```
struct para {
    int number;//socket是第几号
    int len;//name长度
    char name[20];
    bool used = false;//是否使用
};
para ClientInformation[ListenMax];
```

此为`ClientInformation`结构体，内有`number`/`name`/`len`/`used`四个属性。具体属性介绍见代码块。

```
bool isName(char* a) {
    if (a[0] == 'n')
        return true;
    else
        return false;
}
```

此函数用于判断字符串是否为用户姓名。由于自定义语法为：若为姓名，则串的第一个字母为n(第一个字母固定为标识位)

```
//server的接受消息函数
DWORD WINAPI recvMessage(LPVOID tempPara) {
    para* p = (para*)tempPara;
    int num = p->number;
    while (1) {
        //memset(SrecvBuf, 0, SBUF_SIZE);
        if (recv(ClientSockets[num], SrecvBuf, SBUF_SIZE, 0) < 0) {
            //cout << "Connect failed. Error code is " << SOCKET_ERROR << ".\n";
        }
        else {
            if (isGetList(SrecvBuf)) {//用户输入的是getUsrList
                char usrList[SBUF_SIZE];
                memset(usrList, 0, SBUF_SIZE);
                usrList[0] = 'u';//第一个字母定义为u
                int posi = 1;
                for (int tempIndex = 0; tempIndex < ListenMax; tempIndex++) {//封
装usrList并发送
                    if (ClientInformation[tempIndex].used) {
                        usrList[posi] = char(ClientInformation[tempIndex].len);
                        posi += 1;
                        strcat(usrList, ClientInformation[tempIndex].name);
                        posi += ClientInformation[tempIndex].len;
                    }
                }
                send(ClientSockets[num], usrList, sizeof(usrList), 0);
            }
            else {
                if (SrecvBuf[0]=='q') {//有某个用户下线quit
```

```
                    int len = int(SrecvBuf[1]);
                    char* nameString = new char[len + 1];
                    for (int index=0; index < len; index++)
                        nameString[index] = SrecvBuf[index+2];//因为第一个字符为q，
第二个字符为用户名长度信息，用户名从第三个字符开始
                    nameString[len] = '\0';
                    cout << "用户: " << nameString << " 下线了.\n";
                    ConnectedNumber -= 1;
                    cout << "当前用户数量为 " << ConnectedNumber << "\n";
                    // 给当前在线的每个客户都发一遍消息（除了自己）
                    for (int i = 0; i < ListenMax; i++) {
                        if (i == num || ClientSockets[i] == INVALID_SOCKET)
                            continue;
                        else
                            send(ClientSockets[i], SrecvBuf, sizeof(SrecvBuf),
0);
                    }
                    closesocket(ClientSockets[num]);
                    ClientInformation[num].used = false;
                    break;
                }
                else {
                    // 重新封装消息
                    char timeString[26];//时间信息
                    int index = 0;
                    for (; index < 25; index++)
                        timeString[index] = SrecvBuf[index];
                    timeString[25] = '\0';
                    int len = int(SrecvBuf[index]);//目标用户长度
                    index += 1;
                    char* dstName = new char[len + 1];//目标用户名称
                    for (; index < 26 + len; index++)
                        dstName[index - 26] = SrecvBuf[index];
                    dstName[len] = '\0';
                    len = int(SrecvBuf[index]);//来源用户长度
                    index += 1;
                    int lastPosi = len + index;
                    int nowPost = index - 1;
                    char* srcNameString = new char[len + 2];
                    srcNameString[0] = char(len);//来源用户名称
                    for (; index < lastPosi; index++)
                        srcNameString[index - nowPost] = SrecvBuf[index];
                    srcNameString[len + 1] = '\0';

                    char tempBuf[SBUF_SIZE];//真正的消息内容
                    len = 0;
                    char mess[SBUF_SIZE];
                    while (SrecvBuf[index] != '\0') {
                        mess[len] = SrecvBuf[index];
                        len++;
                        index++;
                    }
                    mess[len] = '\0';
                    memset(tempBuf, 0, SBUF_SIZE);
                    strcat(tempBuf, timeString);
```

```
                    strcat(tempBuf, srcNameString);
                    strcat(tempBuf, mess);

                    if (isOpen(dstName)) {//公共发言
                        cout << "From:" << srcNameString + 1 << "\tTime: " <<
timeString;
                        int idx = 0;
                        while (mess[idx] != '\0') {
                            cout << mess[idx];
                            idx++;
                        }
                    cout << endl;
                    }
                    else {
                        // 非公屏发言，则要发送给对应的用户
                        int dst = 0;
                        for (; dst < ListenMax; dst++) {
                            if (ClientInformation[dst].used) {
                                if (isSameStr(ClientInformation[dst].name,
dstName)) {
                                    // 发送消息

send(ClientSockets[ClientInformation[dst].number], tempBuf, sizeof(tempBuf), 0);
                                    break;
                                }
                            }
                        }
                        if (dst == ListenMax)//用户输入有误情况处理
                        {
                            char* temp = new char(strlen(srcNameString));
                            temp = srcNameString + 1;
                            for (int j = 0; j < ListenMax; j++)
                            {
                                if (ClientInformation[j].used) {
                                    //cout << j << " " <<
ClientInformation[j].name << " " << ClientInformation[j].number << endl;
                                    if (strcmp(ClientInformation[j].name,
temp)==0)
                                    {
                                        //cout <<
strcmp(ClientInformation[j].name, temp) << endl;

send(ClientSockets[ClientInformation[j].number], errorMessage, sizeof(tempBuf),
0);
                                        break;
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
    return 0;
```

```
}
```

server相当于一个中转站：它把需要公屏显示的消息留下，并根据接受到的消息进行相应的动作。

1. 用户输入的是`getUsrList`：将用户信息表抽取出来，传回请求用户。
2. 某个用户下线`quit`：显示该用户下线信息，并通知所有用户该用户已下线。
3. 重新封装消息，包括时间信息、目标用户信息、来源用户信息、消息内容
    - 若是公共发言：则直接显示即可。
    - 若是私密发言：则转发给相应用户。
    - 若是错误信息：告知来源用户，输入格式错误。

```
void showList(char* usrlist) {
    int totalLen = getLen(usrlist);
    int posi = 1;
    int number = 1;
    while (posi<totalLen) {
        int len = int(usrlist[posi]);
        posi += 1;
        int lastPost = posi + len;
        cout << number << ". ";
        number += 1;
        for (; posi < lastPost; posi+=1)
            cout << usrlist[posi];
        cout << endl;
    }
}
```

此函数为客户端收到服务器端的消息后打印在线用户。

```
//client端的收到消息处理函数
DWORD WINAPI recvMessagec(LPVOID) {
    while (1) {
        ::memset(CrecvBuf, 0, CBUF_SIZE);
        if (recv(LocalhostSocket, CrecvBuf, CBUF_SIZE, 0) < 0) {
            cout << "Connect failed. Error code is " << SOCKET_ERROR << ".\n";
            system("pause");
        }
        else if(isHello(CrecvBuf))//用户连接成功 Connected!
            cout << CrecvBuf << endl;
        else if(CrecvBuf[0] == 'I' && CrecvBuf[1] == 'n' && CrecvBuf[2] ==
'p')//input destination error
            cout << CrecvBuf << endl;
        else if(CrecvBuf[0] == 'u')//getUsrList
            showList(CrecvBuf);
        else if (isServerQuit(CrecvBuf)){//server quit
            cout << "Server has stopped this chat room, bye.\n";
            break;
        }
        else if (isClientQuit(CrecvBuf)){//有人下线了
            int len = int(CrecvBuf[1]);
            char* quitUsr = new char[len + 1];
            for (int i = 0; i < len; i++)
                quitUsr[i] = CrecvBuf[i + 2];
```

```
            quitUsr[len] = '\0';
            cout << "User: " << quitUsr << " has left from sesstion.\n";
        }
        else {//收到了某人发来的消息，打印来源用户、时间、消息
            char timeString[26];
            int index = 0;
            for (; index < 25; index++)
                timeString[index] = CrecvBuf[index];
            timeString[index] = '\0';
            int len = int(CrecvBuf[index]);
            index += 1;
            char* nameString = new char[len + 1];
            for (; index < 26 + len; index++)
                nameString[index - 26] = CrecvBuf[index];
            nameString[index - 26] = '\0';
            while (CrecvBuf[index] == '\0')
                index++;
            cout << "From user: " << nameString << "\tTime: " << timeString;
            while (CrecvBuf[index] != '\0') {
                cout << CrecvBuf[index];
                index++;
            }
            cout << endl;
        }
    }
    return 0;
}
```

收到消息，根据消息内容进行以下几种动作：

1. 该用户连接成功，显示 `Connected! Welcome to our char room.`
2. 该用户输入有误，显示 `errorMessage`
3. 该用户输入的是 `getUsrList`，则打印当前在线用户表
4. 收到某人下线信息，显示。
5. 收到某人发来的消息，则打印来源用户、时间、消息。

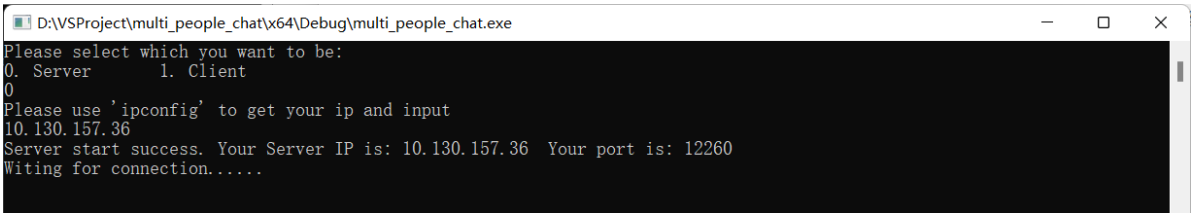
使用方法

1. 运行程序后，进入到选择页面。
2. 输入0以选择成为服务端
    1. 输入本地IP
    2. 开启以后等待连接，连接成功时命令行会显示哪个用户发起了连接、当前有多少个用户在线
3. 输入1以选择成为客户端
    1. 输入想要连接的IP
    2. 输入自己的昵称
    3. 等待连接成功后如果想要将消息发给某个人，发送格式为： `username:message` ，其中username是想要发送到的对象，message是消息内容，中间用英文的":"分隔开
    4. 若想在聊天室内发送消息（即公屏消息），发送格式为： `open:message` 。
    5. 如果想要退出聊天室，输入 quit 以结束本次会话
    6. 如果想要查看当前聊天室内有哪些用户在线，输入 `getUsrList`

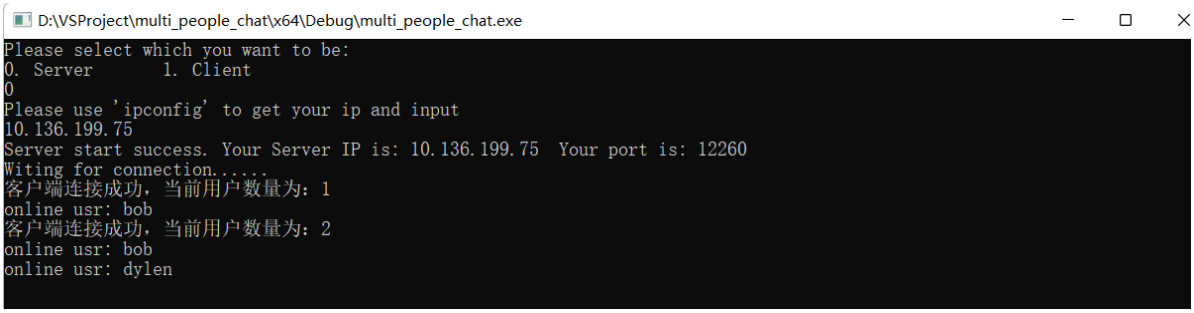
实验效果测试

Server

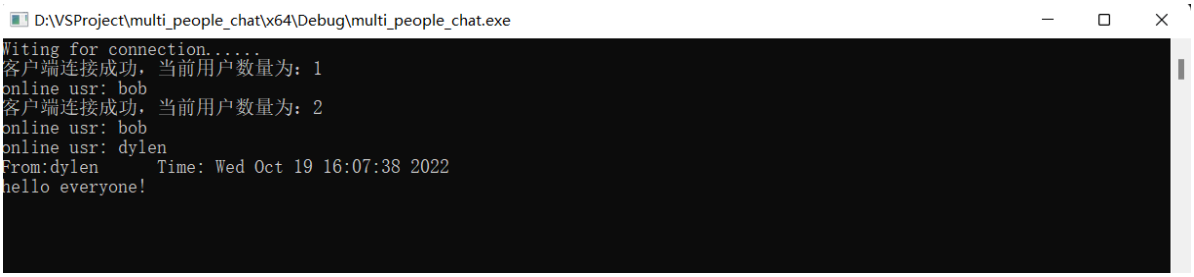
启动服务



两个用户登录



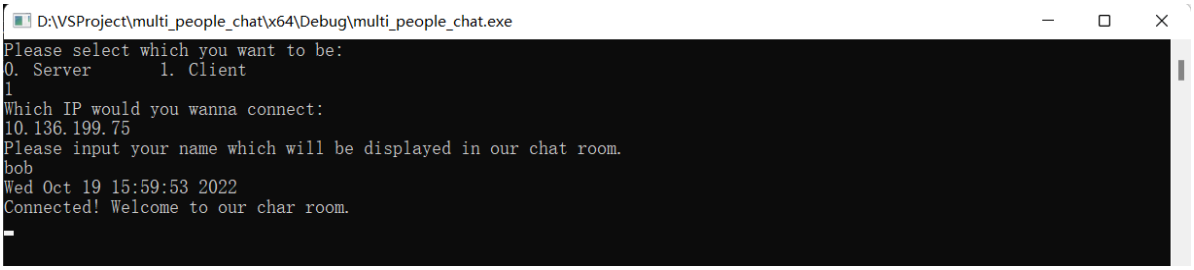
收到公屏消息



用户退出聊天室

Client

建立连接



获取用户列表（需在client端输入）

```
Please select which you want to be:
0. Server       1. Client
1
Which IP would you wanna connect:
10.130.157.36
Please input your name which will be displayed in our chat room.
bob
Tue Oct 18 22:17:40 2022
Connected! Welcome to our char room.
getUsrList
1. jack
2. bob
```

向某人发送消息

```
D:\VSProject\multi_people_chat\x64\Debug\multi_people_chat.exe
Please select which you want to be:
0. Server       1. Client
1
Which IP would you wanna connect:
10.136.199.75
Please input your name which will be displayed in our chat room.
bob
Wed Oct 19 15:59:53 2022
Connected! Welcome to our char room.
dylen:hello!
```

收到用户消息

```
D:\VSProject\multi_people_chat\x64\Debug\multi_people_chat.exe
Please select which you want to be:
0. Server       1. Client
1
Which IP would you wanna connect:
10.136.199.75
Please input your name which will be displayed in our chat room.
dylen
Wed Oct 19 16:00:01 2022
Connected! Welcome to our char room.
From user: bob  Time: Wed Oct 19 16:06:35 2022
hello!
```

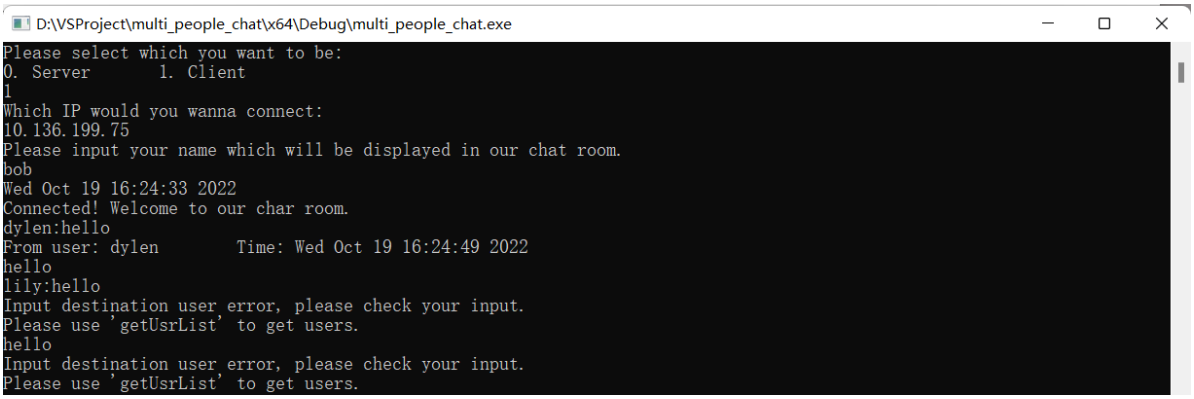
发送公屏消息

```
D:\VSProject\multi_people_chat\x64\Debug\multi_people_chat.exe
Please select which you want to be:
0. Server       1. Client
1
Which IP would you wanna connect:
10.136.199.75
Please input your name which will be displayed in our chat room.
dylen
Wed Oct 19 16:00:01 2022
Connected! Welcome to our char room.
From user: bob  Time: Wed Oct 19 16:06:35 2022
hello!
open:hello everyone!
```

发送用户名错误

```
D:\VSProject\multi_people_chat\x64\Debug\multi_people_chat.exe
Please select which you want to be:
0. Server       1. Client
1
Which IP would you wanna connect:
10.136.199.75
Please input your name which will be displayed in our chat room.
dylen
Wed Oct 19 16:00:01 2022
Connected! Welcome to our char room.
From user: bob  Time: Wed Oct 19 16:06:35 2022
hello!
open:hello everyone!
lily:how are you?
Input destination user error, please check your input.
Please use 'getUsrList' to get users.
```

错误输入



其中一个用户(bob)退出：下图分别为bob/dylen/server视角