



南開大學
Nankai University

计算机学院
计算机网络实验报告

实验 1: 利用 Socket 编写聊天程序

姓名：谢畅

学号：2113665

专业：计算机科学与技术

2023 年 10 月 21 日

目录

1	前期准备	1
1.1	实验概述	1
1.2	前期准备技术	1
2	协议设计	1
2.1	相关原理	1
2.2	总体实现	2
2.2.1	消息定义	3
2.2.2	时序分析	4
3	代码思路以及功能分析	6
3.1	消息的处理	6
3.2	服务器端	8
3.2.1	发消息进程	8
3.2.2	处理客户端消息进程	10
3.2.3	server 中的 main 函数	12
3.3	客户端	15
3.3.1	接受消息进程	15
3.3.2	client 中的 main 函数	16
4	实验思考以及问题分析	18
5	程序展示与验证	19

1 前期准备

1.1 实验概述

本次实验是在 Windows 系统下完成的，利用 C++ 实现了基于流式套接字的多人聊天程序，实现了多个客户端和服务端之间的同时聊天功能，显示界面为命令行界面。还在此基础上实现了一些优化功能，比如聊天室人数限制、服务器对人数的统计、客户端自己发消息显示“(我自己)”，别人发消息显示“(客户 n)”，服务器与客户端的终止等较多功能。

1.2 前期准备技术

- 自定义消息类型，对消息进行封装，便于各个客户端与服务端读取信息，同时定义一系列消息的发送与解析的流程。
- 使用多线程思想，服务端主线程负责监听连接请求，为监听到的每个请求分配一个接受消息的线程进行通信，来接受其客户端发送的消息；同时还设置一个发送消息的线程，此线程可以用于服务端给各个客户端发送消息。
- 在客户端发送消息时，以服务端作为中转站，来对消息进行进一步打包和加工，再分发到各个其他客户端；

2 协议设计

2.1 相关原理

网络协议

1. 语义。语义是解释控制信息每个部分的意义。它规定了需要发出何种控制信息，以及完成的动作与做出什么样的响应。
2. 语法。语法是用户数据与控制信息的结构与格式，以及数据出现的顺序。
3. 时序。时序是对事件发生顺序的详细说明。(也可称为“同步”)。

Socket

- Stream Sockets

流式套接字，也称为 TCP 套接字，是一种面向连接的套接字类型。它提供了可靠的、双向的、基于字节流的通信方式。流式套接字使用 TCP（传输控制协议）作为底层协议。确保数据按顺序到达，且没有丢失。

- Datagram Sockets

数据报套接字，也称为 UDP 套接字，是一种无连接的套接字类型。它使用 UDP（用户数据报协议）作为底层协议。不需要建立连接，可以直接发送数据包，没有顺序。

2.2 总体实现

基于上述原理，在这次实验我实现了如下的聊天协议：

- 采取流式套接字 (stream sockets)，所以我们的底层协议是基于 TCP 的，而 TCP 协议是一种可靠的协议，确保数据按照发送的顺序到达目的地，而且不会丢失。这确保我们的聊天程序的消息顺序是按序的，而且聊天消息传输与发送是可靠的。
- 聊天消息的结构是自定义的结构体 `Mymsg` 类型，里面有 `id,name,online,Time,content` 相关变量，对应着消息发送者的 `id` 标识符，发送者名字，发送者在线与否，发送时间，发送内容。经过封装，我们的发送消息前面加上了消息头，同时限制一条消息的大小为 1080 字节。而在消息头中包含有关数据的信息，可以让接收方更有效地处理数据。
- 在聊天中设计了系统端与客户端。其中聊天程序由一个系统端提供服务，有多个聊天程序同时聊天，为了避免过多程序同时聊天，这里设置了聊天数量的限制，即同时聊天数目最大为 5 人。其中，客户端的聊天是通过服务端的消息转发实现的。
- 时序为按序的。由于底层采取流式套接字，说明是基于 TCP 进行数据传输的，显然发送消息是按序发送的，按序接受。
- 这里为了标识每个客户的 `id`，昵称，限制客户数量大小。在客户建立连接时会会有一个初始阶段，这个阶段系统端会发给客户端一个 2 字节的数据。`char ifinit[2]`，第一位标识用户的数量是否超过最大限制，第二位标识用户的 ID。如果第一位判断后，用户数量未超过限制，对应客户端会提示输入昵称；而相应地，系统端会创建线程，在起始处接受用户昵称。而在此初始阶段结束后，会开始用封装的消息进行通信。
- 设计系统与客户端退出的协议。当输入消息“exit”后，会调用 `exit` 自动退出，同时会有几秒钟的延时设置。这里系统可以终止服务，同时客户端可以退出在线模式，二者的方式都是输入“exit”。
- 打印输出消息采取特定的函数，在其中检查了消息的合法性，避免了打印错误消息。针对本人发送的消息，在本人界面显现出 `Time (我自己)[昵称]:Message`；而针对他人发送消息，显现 `Time (客户 ID)[昵称]:Message`。其中 `Time` 代表发送时间，`Message` 表示发送消息内容。

2.2.1 消息定义

语法：结构体设计

这里我设计了客户端与服务端之间信息交互的消息格式，定义了 Mymsg 结构体，如下图所示。

My msg 消息结构体

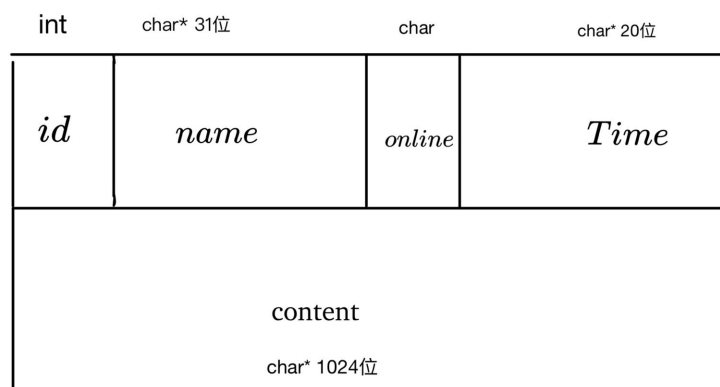


图 2.1: 自定义消息类型 Mymsg

这里先详细介绍一下 Mymsg 中各个属性值的含义。其中 *id* 代表发送消息者的 id，这里为 0 代表 [系统提醒] 的发送，不为 0 代表客户端的 id 值。而 *name* 代表消息发送者的名字，其中系统端名字设为默认——系统提醒。而 *online* 代表该客户是否发送“exit”退出，如果退出，那么 *online* 为“0”，会直接将该客户端退出；如果系统端 *online* 为“0”，代表系统终止服务，会强制将所有客户端退出；若为“1”，代表在线，继续接受消息/发送消息即可。最后 *Time* 是发送消息的时间，直接在封装 Msg 时提取时间即可。

语义：消息的发送与接受流程

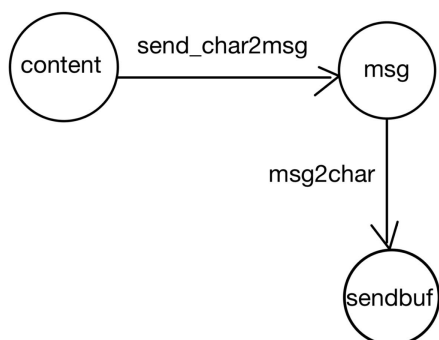


图 2.2: 消息的发送

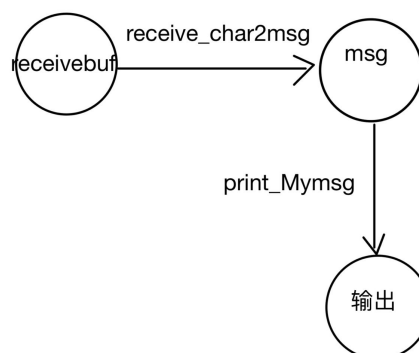


图 2.3: 消息的接受

这里我们的消息发送与接受流程，如图 2.2 与图 2.3 所示：

由于我们使用的是流式套接字，在其中消息的接受函数与发送函数是 `recv`、`send`。这两个函数中第二个参数传送的数据是以字节为单位传输，以 `char*` 格式传送。因此在传送自定义消息类型时，需要用到相关的转换函数。

在图2.2中，很明显可以看出消息发送的流程。

- 首先对发送内容进行封装。将 `char` 数组 (限制最大 1024 字节)，转换为 `msg`，`msg` 结构体中包含一些额外的标识信息。
- 然后由于采取的套接字发送函数是 `send`，需要将 `msg` 转为 `char*` 再继续发送。这里转换采取的是 `memcpy`，直接将字节数拷贝，稳定可靠。

在图2.3中，可以看到消息的接受流程。

- 首先我们 `recv` 函数收到的是 `char *receivebuf`。要对收到的字节数进行转换。
- 然后调用函数 `receive_char2msg`，运用 `memcpy` 直接将字节数拷贝到一个 `msg` 结构体，这样就得到了对应的消息 `msg`。
- 最后调用特定的输出函数 `print_Mymsg`，用特定的格式打印一个 `msg`。

2.2.2 时序分析

这里服务端与客户端的连接时序逻辑进行如下的分析。

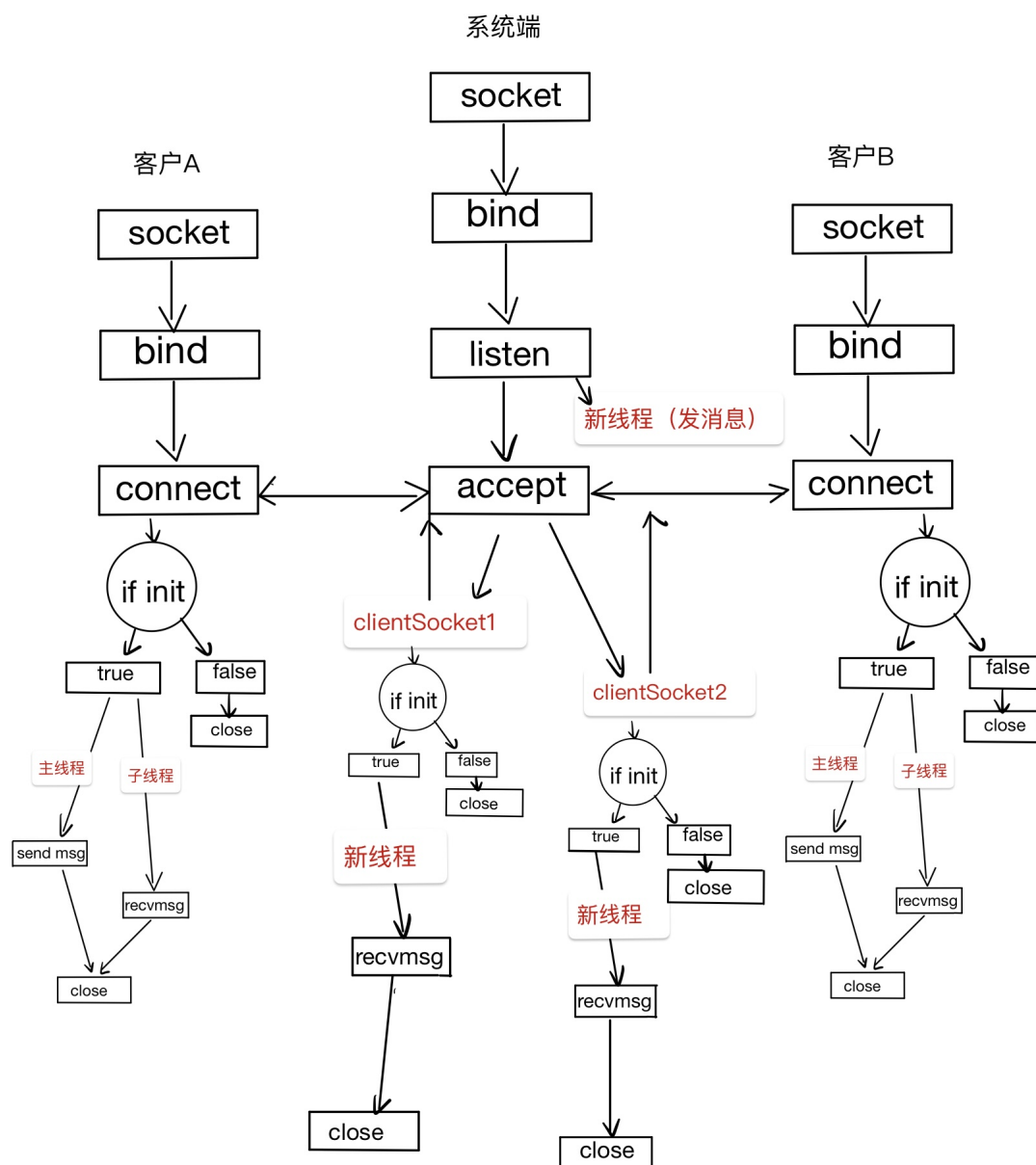


图 2.4: 服务端与不同客户端连接图

如上图，由于我们的聊天室有限制人数的功能，这里的时序逻辑在 tcp 的基础上进行相关改进。在这里增加了一个 *if_init* 的控制信号。具体作用是当进行新的连接时，得到了新的 socket，这个时候先判断聊天室人数，如果超过限制，服务端控制 *if_init* 信号，将信号发送给对应客户端，同时等待 1 秒后将此 socket 关闭。

而客户端在 connect 成功后，还需要进行控制信号的 recv 操作，如果 *if_init* 信号为 false，直接将 socket 关闭，退出即可，并在屏幕上显示相关信息。

如果没有超过限制数，那么与正常 tcp 时序逻辑一致，只不过在这里我们增加了一个新的线程用来发送系统端的消息。在客户端连接申请后，系统端接受并同意，创建新的 socket 负责此客户端，同时建立新的线程，这里新线程用来接受此客户端消息，并转发给其他客户端。

下面对服务端客户端收发消息的时序逻辑进行分析。

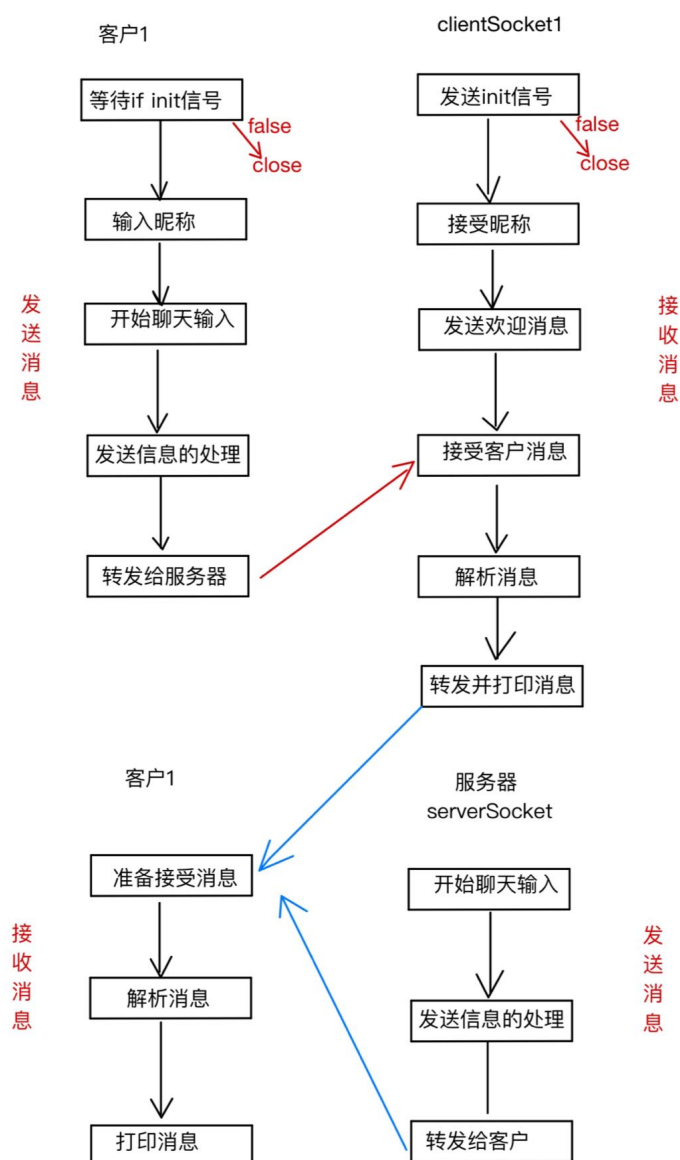


图 2.5: 服务器与客户端收发消息的时序逻辑

- 可以看到客户端有两个线程，主线程用来发消息，子线程用来接收消息。
- 而服务器端，每个客户对应一个 clientSocket，里面有一个专门的线程用于接受该客户消息；而服务器端设定了一个线程用于给每个客户发送系统提醒的消息。

3 代码思路以及功能分析

3.1 消息的处理

1. 消息结构体定义

```

1 #define msg_size 1080 //封装的 Mymsg 的字节大小
2 #define content_size 1024//发送消息的最大大小
3 #define idMax 100
4 //封装消息的结构体
5 struct Mymsg {
6     int id; //标识符
7     char name[31]; //昵称限制在 31 字节内
8     char online; //判断是否在线
9     char Time[20]; //发送消息的时间
10    char content[1024]; //发送内容
11 }; //1080 字节

```

2. 消息 Msg 转为 char*

```

1 //需要把封装的 message 转化为 char 去发送
2 char* msg2char(Mymsg message) {
3     char temp[msg_size];
4     memcpy(temp, &message, sizeof(Mymsg)); //直接将字节数拷贝
5     return temp;
6 }

```

3. 封装内容为一条 Msg

这里是将发送的内容封装成一条 Msg，首先直接将具体内容拷贝，然后获取现在的时间，将其转为字符串写入 Msg，最后判断消息内容是否为 exit，如果是 exit，将 online 位置'1'，否则置'0'。

```

1 //把消息内容封装成 message
2 Mymsg send_char2msg(char* content, int _id, char* _name) {
3     Mymsg temp;
4     strcpy_s(temp.content, content); //消息内容复制
5     char timeStr[20] = { 0 };
6     time_t t = time(0);
7     strftime(timeStr, sizeof(timeStr), "%Y-%m-%d %H:%M:%S", localtime(&t));
8     strcpy(temp.Time, timeStr);
9
10    //判断 content 是否为 exit
11    if (strcmp(content, "exit") == 0) {
12        temp.online = '0'; //不在线

```

```

13     }
14     else {
15         temp.online = '1';
16     }
17     memcpy(&temp, &_id, sizeof(int));
18     strcpy(temp.name, _name);
19     return temp; //返回封装好的 message
20 }

```

4. 将接受的 buf 转为 msg

```

1 //接收的 char* 转为 message
2 Mymsg receive_char2msg(char* receive_buf) {
3     Mymsg temp;
4     memcpy(&temp, receive_buf, sizeof(Mymsg));
5     return temp;
6 }

```

5. 打印输出消息

```

1 //格式输出
2 void print_Mymsg(Mymsg t) {
3     if (t.id == 0) {
4         cout << t.Time << " " << "[" << t.name << "]: " << t.content << endl;
5     }
6     else if (t.id >= 1 && t.id <= idMax) {
7         //虽然总数只有 5 个, 但是 id 可能为 100, 这里设置了一个较为合理的值
8         cout << t.Time << " (客户" << t.id << ") [" << t.name << "]: "
9         << t.content << endl;
10    }
11    else {}
12 }

```

3.2 服务器端

3.2.1 发消息进程

这里设置了系统也可以发消息, 因此创建了一个发消息的进程, 具体作用是系统输入, 将输入消息发送给所有客户, 如果系统发送的是 exit, 那么会提醒系统终止服务。

- 首先函数通过传递给线程的参数，获取到 serverSocket 值，通过此 socket 发送消息。
- 获取系统端输入的内容。将内容封装成 send_msg 传输。
- 判断服务器发送的消息是否为"exit"，如果是 exit，最后传输一句话，提醒客户端，服务器要终止。将这句话传输并在服务器打印后，先睡眠 5 秒钟。这里是为了让处理客户端通信的 socket 先行关闭，然后最后关闭 server_socket，释放资源后退出。

```

1  //系统提醒的消息发送
2  DWORD WINAPI ThreadSend(LPVOID lpParameter) {
3      SOCKET send_socket = (SOCKET)lpParameter;
4      int ret = 0;
5      while (1) {
6          char bufsend[content_size] = { 0 };
7          cin.getline(bufsend, content_size);
8          Mymsg send_msg = send_char2msg(bufsend, 0, server_name);
9          print_Mymsg(send_msg);
10         for (auto i : client)
11             ret = send(i.first, msg2char(send_msg), msg_size, 0);
12         //判断服务器是否终止
13         if (send_msg.online == '0') {
14             server_exitFlag = 1;
15             string str2 = " 聊天服务器将于此刻关闭，请各位客户准备退出，倒计时 5 秒";
16             char* p = (char*)str2.data();
17             Mymsg tmp = send_char2msg(p, 0, server_name);
18             for (auto i : client)
19                 ret = send(i.first, msg2char(tmp), msg_size, 0);
20             //强制退出
21             print_Mymsg(tmp);
22             Sleep(5000); //让所有接受进程释放掉 receive_socket
23             closesocket(send_socket); //send_socket 是 server_socket
24             WSACleanup();
25             exit(100);
26         }
27     }
28     return 0;
29 }

```

3.2.2 处理客户端消息进程

这里为每个客户端创建了一个线程，线程函数负责处理该客户的相关消息与信息的传送。

1. 首先记录当前客户 id，储存昵称。
2. 对新来的客户发送欢迎消息。这里设置对已经到场的用户、当前新来的用户，不同的欢迎词。
3. 进入 while 循环，开始循环接受此客户发来的消息。接受成功则打印此消息，并且把消息转发给所有客户。
4. 如果接受失败那么退出循环。如果是系统端终止服务引起的客户退出，这里清除 socket 直接返回即可；如果是客户端退出/掉线等等原因导致，这里先删去此用户，并向所有用户广播该客户的退出。

```

1 //服务端接受消息的线程函数 负责转发消息给所有客户
2 DWORD WINAPI ThreadReceive(LPVOID lpParameter) {
3     int cur_id = client_id;//当前 Id
4     SOCKET receive_socket = (SOCKET)lpParameter;
5     //1. 储存昵称
6     char cur_clientName[31] = { 0 };//客户的名称
7     int ret = recv(receive_socket, cur_clientName, 31, 0);
8     if (ret == SOCKET_ERROR || ret <= 0) { //初始化 client 时出错
9         closesocket(receive_socket);
10        return 0;
11    }
12    client[receive_socket] = string(cur_clientName);//储存昵称
13
14    //初始化完成，现在开始用封装的 Msg 传递信息
15    //2. 系统发送欢迎消息
16    string stemp1 = (string)"client" + to_string(cur_id) + (string)" 的名字是"
17    + client[receive_socket] + (string)", ta 来了，大家多多欢迎吧!";
18    string stemp2 = (string)" 欢迎你的到来，当前你是 client" +
19    to_string(cur_id);//针对当前用户欢迎词不一样
20
21    char* welcome1 = (char*)stemp1.data();//欢迎语句
22    Mymsg first_wel1 = send_char2msg(welcome1, 0, server_name);//封装成 Mymsg
23    char* welcome2 = (char*)stemp2.data();//欢迎语句
24    Mymsg first_wel2 = send_char2msg(welcome2, 0, server_name);//封装成 Mymsg

```

```
25     for (auto i : client) {
26         if (i.first == receive_socket)
27             send(i.first, msg2char(first_wel2), sizeof(Mymsg), 0);
28         else
29             send(i.first, msg2char(first_wel1), sizeof(Mymsg), 0);
30     }
31     print_Mymsg(first_wel1); //系统端也打印
32     ret = 0;
33
34     //3. 开始循环接受此 client 消息
35     while (1) {
36         char bufRecv[msg_size];
37         ret = recv(receive_socket, bufRecv, msg_size, 0);
38         if (ret != SOCKET_ERROR && ret > 0) { //接受 0 个也代表客户端退出
39             Mymsg tmp = receive_char2msg(bufRecv);
40             tmp.id = cur_id; //发送的客户端不知道 id, 默认 0, 这里要修改
41             if (tmp.online == '0') { //标记位的判断
42                 //print_Mymsg(tmp); //客户端退出的 exit 不打印
43                 break; //客户端退出
44             }
45             else {
46                 print_Mymsg(tmp);
47                 for (auto i : client)
48                     send(i.first, msg2char(tmp), msg_size, 0); //直接把收到的转发即可
49             }
50         }
51         else {
52             break; //同样标识为客户端退出
53         }
54     }
55     if (server_exitFlag) { //系统发送退出消息
56         closesocket(receive_socket);
57         return 0; //直接退出此线程, 善后工作在发送进程中
58     }
59     //客户退出, 同步给其余所有人
60     string str2 = (string)" 离开了聊天室! ";
61     char* p = (char*)str2.data();
62     map<SOCKET, string>::iterator iter = client.find(receive_socket);
63     client.erase(iter); //退出聊天室, 要把 client 记录的映射关系删掉
```

```

64     client_total_num--; //注意 id 不会减少, 只减少总数
65     //离开的成员自动发送消息 (客户端代替转发)
66     Mymsg exit_client = send_char2msg(p, cur_id, cur_clientName);
67     print_Mymsg(exit_client);
68     for (auto i : client)
69         send(i.first, msg2char(exit_client), msg_size, 0);
70     closesocket(receive_socket); //把这个客户端的进程关掉即可
71     return 0; //准备退出此线程
72
73 }

```

3.2.3 server 中的 main 函数

1. 前期工作

- 首先初始化 *winsock2* 库, 检查初始化是否成功
- 创建服务器套接字 *serverSocket*, 绑定 ip 与端口号
- 设置监听状态, 以便接受客户端请求

```

1  cout << "##### 正在创建聊天室系统端 #####" << endl;
2  // 加载 winsock 环境
3  WSADATA wsaData;
4  if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
5      cout << " 初始化 Socket DLL 失败!" << endl;
6      return 0;
7  }
8  else
9      cout << " 成功初始化 Socket DLL, 网络环境加载成功" << endl;
10
11 // 创建套接字
12 serverSocket = socket(AF_INET, SOCK_STREAM, 0); //使用流式套接字 基于 TCP 的按顺序
13 if (serverSocket == INVALID_SOCKET) {
14     cout << " 流式套接字创建失败" << endl;
15     WSACleanup();
16 }
17 else
18     cout << " 流式套接字创建成功" << endl;
19

```

```

20 // 给服务器的套接字绑定 ip 地址和端口: bind 函数
21 serverAddr.sin_family = AF_INET;
22 serverAddr.sin_port = htons(8000);
23 serverAddr.sin_addr.s_addr = inet_addr("127.0.0.1");
24
25 int len = sizeof(sockaddr_in);
26 if (bind(serverSocket, (SOCKADDR*)&serverAddr, len) == SOCKET_ERROR) {
27     cout << " 服务器绑定端口和 ip 失败" << endl;
28     WSACleanup();
29 }
30 else {
31     cout << " 服务器绑定端口和 Ip 成功" << endl;
32 }
33 // 监听端口
34 if (listen(serverSocket, MaxClient) != 0) {
35     cout << " 设置监听状态失败! " << endl;
36     WSACleanup(); // 结束使用 Socket, 释放 socket dll 资源
37 }
38 else
39     cout << " 设置监听状态成功! " << endl;
40
41 cout << " 服务器监听连接中, 请稍等....." << endl;
42 cout << "##### 等待客户端进入 #####" << endl;

```

2. 接受客户端的请求

- 首先创立一个线程, 用于服务器端发送消息
- 进入 while 循环, 不断处理客户端的连接请求。
- 如果连接数大于 *MaxClient*, 将 *ifinit* 第 1 位置'0', 发送给对应客户端, 等待 1 秒它接受信息, 随后关闭此 socket, 复原相关计数器, *continue* 返回循环起始处。
- 如果连接数正常。那么将 *ifinit* 的第 2 位置为对应客户端 Id, 发送此消息即可完成初始化。然后为此客户创立对应的新线程, 用于处理该线程发送的消息。

```

1 // 创建一个线程, 用于服务器发送消息
2 CloseHandle(CreateThread(NULL, 0, ThreadSend, (LPVOID)serverSocket, 0, NULL));
3
4 // 循环接受: 客户端发来的连接

```

```
5  while (1) {
6
7      sockaddr_in clientAddr; //新建当前 client 的地址
8      len = sizeof(sockaddr_in);
9      //创建新的客户端的套接字
10     SOCKET cur_clientSocket = accept(serverSocket, (sockaddr*)&clientAddr, &len);
11     if (cur_clientSocket == INVALID_SOCKET) {
12         cout << " 与客户端连接失败" << endl;
13         closesocket(cur_clientSocket);
14         WSACleanup();
15         return 0;
16     }
17     client_total_num++; //客户总数 ++, 客户端连接到套接字
18     client_id++; //id++
19
20     //ifinit 作为成功连接的判断符号
21     // 如果当前连接的客户端数量已经达到 6 个, 则关闭新连接并继续等待下一个连接请求
22     char ifinit[2] = { '1', '0' };
23     if (client_total_num > MaxClient) {
24         cout << " 已达到最大连接数, 拒绝新连接" << endl;
25         ifinit[0] = '0';
26         send(cur_clientSocket, ifinit, 2, 0); //传递给客户端的标识符, 用作初始的符号
27         Sleep(1000); //等待 1 秒
28         closesocket(cur_clientSocket);
29         client_total_num--; //还原
30         client_id--; //id 也要还原
31         continue;
32     }
33     else {
34         ifinit[1] = client_id + '0' - 0;
35         send(cur_clientSocket, ifinit, 2, 0);
36     }
37     cout << "client_total_num(当前客户人数):" << client_total_num << endl;
38
39     HANDLE hthread2 = CreateThread(NULL, 0, ThreadReceive,
40     (LPVOID)cur_clientSocket, 0, NULL); //创建线程用于接受该客户端消息
41     //在线程函数里面再建立 client 的映射关系
42     if (hthread2 == NULL) //线程创建失败
43     {
```



```
44     perror("The Thread is failed!\n");
45     exit(EXIT_FAILURE);
46 }
47 else
48 {
49     CloseHandle(hthread2);
50 }
51
52 }
53 // 关闭主 socket 连接, 释放资源
54 closesocket(serverSocket);
55 WSACleanup();
56 return 0;
```

3.3 客户端

3.3.1 接受消息进程

这里为一个客户端创立了一个新线程, 用于接受服务器传来的消息。这个消息可能是服务器转发的消息, 也可能是服务器主动发送的消息。

- 首先注意这里接受的缓冲区字节数一定是固定的 `msg_size`, 防止读取数据出错。
- 判断读取的字节数是否正确。不正确那么退出程序。
- 如果正确, 将收到的缓冲区字节转为一条 `msg`。如果这条 `msg` 是系统发送的, 并且系统要退出, 那么先接受最后一条消息, 在打印完后, 睡眠 2s, 关闭资源后让程序退出。

```
1 //这里创建一个子进程来收客户端消息即可, 主进程用来发消息
2 DWORD WINAPI ThreadReceive() {
3     int ret = 0;
4     while (1) {
5         char bufrecv[msg_size] = { 0 }; //用来接受数据
6         ret = recv(clientSocket, bufrecv, msg_size, 0);
7         if (ret == SOCKET_ERROR || ret <= 0) {
8             cout << " 连接错误! 2s 后退出" << endl;
9             Sleep(2000);
10            closesocket(clientSocket);
11            WSACleanup();
```

```

12         exit(100); //强制所有线程全部退出
13     }
14     Mymsg temp = receive_char2msg(bufrecv);
15
16     if (temp.id == 0) {
17         if (temp.online == '0') { //系统端要退出
18             char exitMessage[msg_size] = { 0 };
19             recv(clientSocket, exitMessage, msg_size, 0);
20             print_Mymsg(receive_char2msg(exitMessage)); //打印 exitMessage 后退出
21             Sleep(2000);
22             closesocket(clientSocket); //释放资源
23             WSACleanup();
24             exit(100); //由于系统端终止，强制客户端退出
25             //break;
26         }
27     }
28     //说明服务器没有终止，打印 msg
29     print_Mymsg(temp);
30 }
31 return 0;
32 }

```

3.3.2 client 中的 main 函数

这里 main 函数为主线程，充当发消息的功能。

1. 前期工作

- 初始化 winsock2 库，创建客户套接字 *clientSocket*，连接客户端。
- 接受服务器发送的 2 字节控制信号，如果 *ifinit[0]* 为 '0'，说明创建失败，释放资源后直接退出；如果为 '1'，创建成功，读取 *ifinit[1]* 为 ID 号。
- 创建成功后，输入昵称。这里固定了输入昵称最大 31 字节。

```

1 .....//前面连接的代码与 server 中的类似，就不放代码了
2 cout << "##### 正在创建聊天室客户端端 #####" << endl;
3 cout << "please send a message or use \"exit\" to exit / 请发送消息或使用 exit 退出" << endl;
4 cout << " 正在初始化..... 请稍等" << endl;
5
6 char ifinit[2]; //初始 2bit, 判断是否成功

```

```

7  recv(clientSocket, ifinit, 2, 0);
8  if (ifinit[0] == '0') {
9      cout << "##### 聊天室客户端创建失败, 倒计时 3 秒退出 #####" << endl;
10     Sleep(3000); //倒计时三秒
11     closesocket(clientSocket);
12     WSACleanup();
13     return 0;
14 }
15 else {
16     cur_clientID = ifinit[1] + 0 - '0';
17     cout << "##### 聊天室客户端创建成功 #####" << endl;
18 }
19 // 发送和接受数据即可
20 string name;
21 cout << " 聊天前请输入你的昵称: ";
22 getline(cin, name); // 读入一整行, 可以有空格
23 char* client_name = (char*)name.data();
24 int ret = send(clientSocket, name.data(), 31, 0); //和 server 匹配, 31 字节
25 if (ret == SOCKET_ERROR || ret <= 0) {
26     cout << " 连接错误! 3s 后退出";
27     // 关闭连接, 释放资源
28     Sleep(3000);
29     closesocket(clientSocket);
30     WSACleanup();
31     return 0;
32 }

```

2. 发送消息

- 首先创立一个子线程, 用于接受客户端发送/转发的消息。
- 然后主线程去发送消息。这里先从客户端读取输入的内容, 封装成 msg 去发送。如果此条消息为"exit", 打印自动退出的提醒, 然后主线程释放资源, 程序关闭。

```

1  // 初始化全部完成, 这里开始使用封装的 msg 传递信息
2  ret = 0;
3  //创建子线程
4  CloseHandle(CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)ThreadReceive, NULL, 0, NULL));
5  //在线程函数里面再建立 client 的映射关系
6  while (1) {

```

```

7
8     char bufrecv[1024] = { 0 };
9     cin.getline(bufrecv, 1024);
10    Mymsg temp = send_char2msg(bufrecv, 0, client_name);
11    //这里的 0 不是 id, 只是默认传参, 在服务端传的时候会知道 id
12
13    ret = send(clientSocket, msg2char(temp), msg_size, 0);
14    //print_Mymsg(temp); 接受服务器发送的再打印, 这个时候才能获得 id
15    if (ret == SOCKET_ERROR || ret <= 0) {
16        cout << " 发送消息失败! 程序在 3 秒后退出" << endl;
17        Sleep(3000);
18        break;
19    }
20    else {
21        if (temp.online == '0') {
22            cout << " 自动提醒: 客户端" << client_name << " 在 3 秒后退出" << endl;
23            Sleep(3000);
24            break;
25        }
26    }
27
28 }
29 // 关闭连接, 释放资源
30 closesocket(clientSocket);
31 WSACleanup();
32 return 0;

```

4 实验思考以及问题分析

在这次实验中遇到了一些问题, 比如多打印了一句话, 或者打印出”烫烫烫....”字符。经过研究并分析、测试发现是因为:

发送消息与接收消息的字节大小不匹配! 由于我开始时会提醒用户输入昵称, 而这里如果 client.cpp 中发送的大小是 100, 而 server 对应接受的昵称字节大小为 31, 就会有 69 个字节没有被接受, 那么它会等到下一轮继续发送, 这样 server 就会虚假的接收到下一条消息, 但是 client 并没有发送消息。

```

1 client:
2 string name;

```

```

3  cout << " 聊天前请输入你的昵称: ";
4  getline(cin, name); // 读入一整行, 可以有空格
5  char* client_name = (char*)name.data();
6  int ret = send(clientSocket, name.data(), 100, 0); //和 server 匹配, 31 字节
7  .....
8  server:
9  char cur_clientName[31] = { 0 }; //客户的名称
10 int ret = recv(receive_socket, cur_clientName, 31, 0);

```

结果如下所示

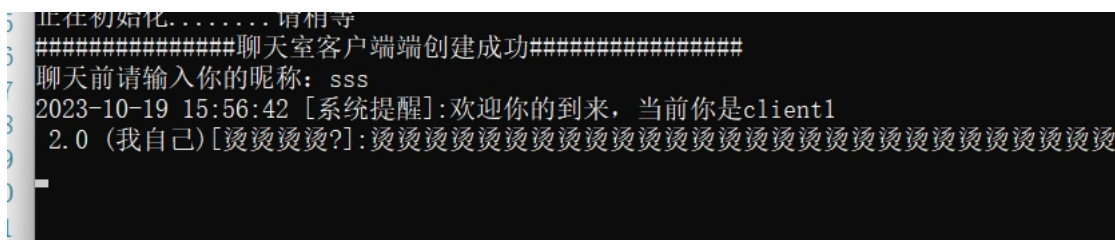


图 4.6: 错误显示

调试的证据:

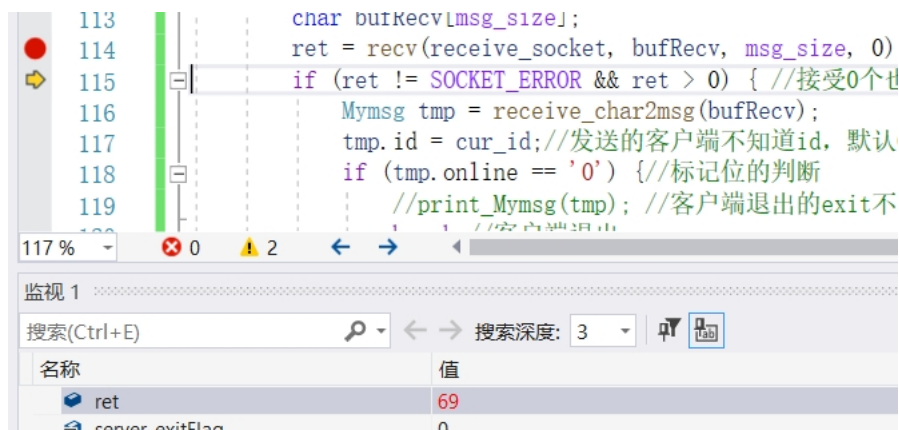


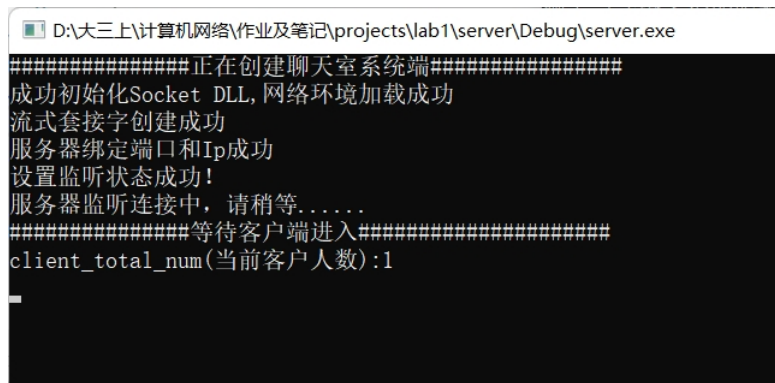
图 4.7: ret 大小

因此, 在程序中 send 与 recv 时要额外注意字节数大小的对应。如果大小没有对应上, 会造成程序乱码的输出。而经过此问题的分析, 我重新检查了字节数发送与接受的对应, 程序正确了。

5 程序展示与验证

1.server 的启动

可以看到输出一系列日志消息，以及当前系统中客户的人数。

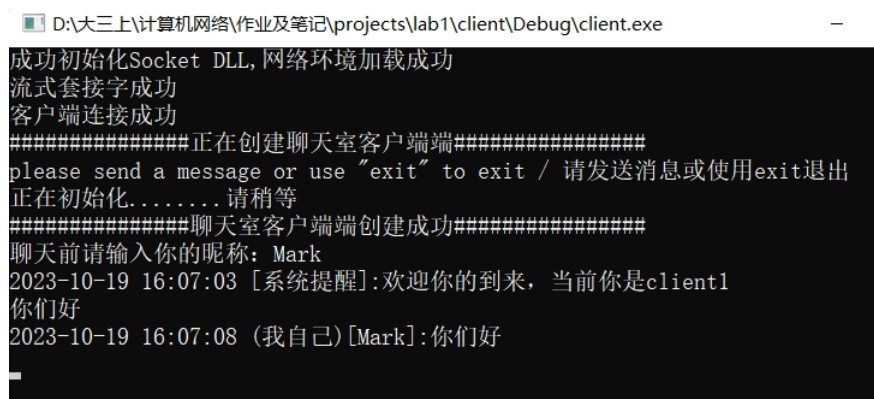


```
D:\大三上\计算机网络\作业及笔记\projects\lab1\server\Debug\server.exe
##### 正在创建聊天室系统端#####
成功初始化Socket DLL, 网络环境加载成功
流式套接字创建成功
服务器绑定端口和Ip成功
设置监听状态成功!
服务器监听连接中, 请稍等.....
#####等待客户端进入#####
client_total_num(当前客户人数):1
```

图 5.8: server 的正常启动

2.client 的启动

可以看到初始化完成后，接收到系统的提醒消息，而且自己发送消息有 (我自己) 的标识符。



```
D:\大三上\计算机网络\作业及笔记\projects\lab1\client\Debug\client.exe
成功初始化Socket DLL, 网络环境加载成功
流式套接字成功
客户端连接成功
##### 正在创建聊天室客户端端#####
please send a message or use "exit" to exit / 请发送消息或使用exit退出
正在初始化.....请稍等
#####聊天室客户端端创建成功#####
聊天前请输入你的昵称: Mark
2023-10-19 16:07:03 [系统提醒]:欢迎你的到来, 当前你是client1
你们好
2023-10-19 16:07:08 (我自己)[Mark]:你们好
```

图 5.9: client 的正常启动

3. 客户的对话

可以看到中英文都没有数据的丢失，正常进行中英文对话。


```
D:\大三上\计算机网络\作业及笔记\projects\lab1\client\Debug\client.exe
##### 正在创建聊天室客户端#####
please send a message or use "exit" to exit / 请发送消息或使用exit退出
正在初始化.....请稍候
#####聊天室客户端创建成功#####
聊天前请输入你的昵称: Mark
2023-10-19 16:07:03 [系统提醒]:欢迎你的到来, 当前你是client1
你们好
2023-10-19 16:07:08 (我自己)[Mark]:你们好
2023-10-19 16:09:14 [系统提醒]:client2的名字是WentWorth, ta来了, 大家多多欢迎吧!
2023-10-19 16:09:24 (客户2) [WentWorth]:hello, everybody!
2023-10-19 16:09:34 [系统提醒]:client3的名字是李凡, ta来了, 大家多多欢迎吧!
2023-10-19 16:09:38 (客户3) [李凡]:你们好!
2023-10-19 16:09:46 (客户3) [李凡]:今天是个不错的日子
微软拼音 半:
```

图 5.10: 对话人物界面

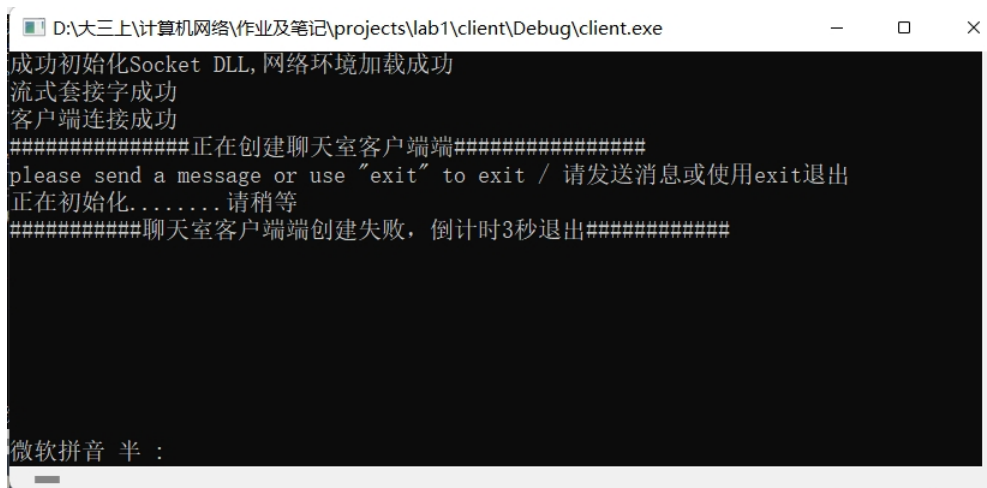
```
D:\大三上\计算机网络\作业及笔记\projects\lab1\client\Debug\client.exe
流式套接字成功
客户端连接成功
##### 正在创建聊天室客户端#####
please send a message or use "exit" to exit / 请发送消息或使用exit退出
正在初始化.....请稍候
#####聊天室客户端创建成功#####
聊天前请输入你的昵称: WentWorth
2023-10-19 16:09:14 [系统提醒]:欢迎你的到来, 当前你是client2
hello, everybody!
2023-10-19 16:09:24 (我自己)[WentWorth]:hello, everybody!
2023-10-19 16:09:34 [系统提醒]:client3的名字是李凡, ta来了, 大家多多欢迎吧!
2023-10-19 16:09:38 (客户3) [李凡]:你们好!
2023-10-19 16:09:46 (客户3) [李凡]:今天是个不错的日子
微软拼音 半:

D:\大三上\计算机网络\作业及笔记\projects\lab1\server\Debug\server.exe
##### 正在创建聊天室系统端#####
成功初始化Socket DLL, 网络环境加载成功
流式套接字创建成功
服务器绑定端口和Ip成功
设置监听状态成功!
服务器监听连接中, 请稍候.....
#####等待客户端进入#####
client_total_num(当前客户人数):1
2023-10-19 16:07:03 [系统提醒]:client1的名字是Mark, ta来了, 大家多多欢迎吧!
2023-10-19 16:07:08 (客户1) [Mark]:你们好
client_total_num(当前客户人数):2
2023-10-19 16:09:14 [系统提醒]:client2的名字是WentWorth, ta来了, 大家多多欢迎吧!
2023-10-19 16:09:24 (客户2) [WentWorth]:hello, everybody!
client_total_num(当前客户人数):3
2023-10-19 16:09:34 [系统提醒]:client3的名字是李凡, ta来了, 大家多多欢迎吧!
2023-10-19 16:09:38 (客户3) [李凡]:你们好!
2023-10-19 16:09:46 (客户3) [李凡]:今天是个不错的日子
微软拼音 半:
```

图 5.11: 对话

4. 客户端人数限制

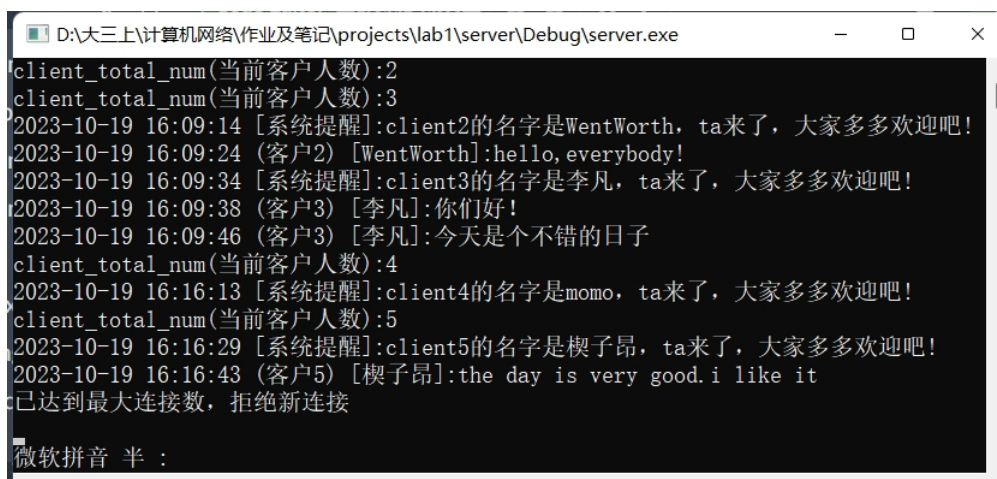
超过 5 人时, 可以看到客户端会自动退出。



```
D:\大三上\计算机网络\作业及笔记\projects\lab1\client\Debug\client.exe
成功初始化Socket DLL, 网络环境加载成功
流式套接字成功
客户端连接成功
#####正在创建聊天室客户端#####
please send a message or use "exit" to exit / 请发送消息或使用exit退出
正在初始化.....请稍候
#####聊天室客户端创建失败, 倒计时3秒退出#####

微软拼音 半 :
```

图 5.12: 人数限制 _client 端



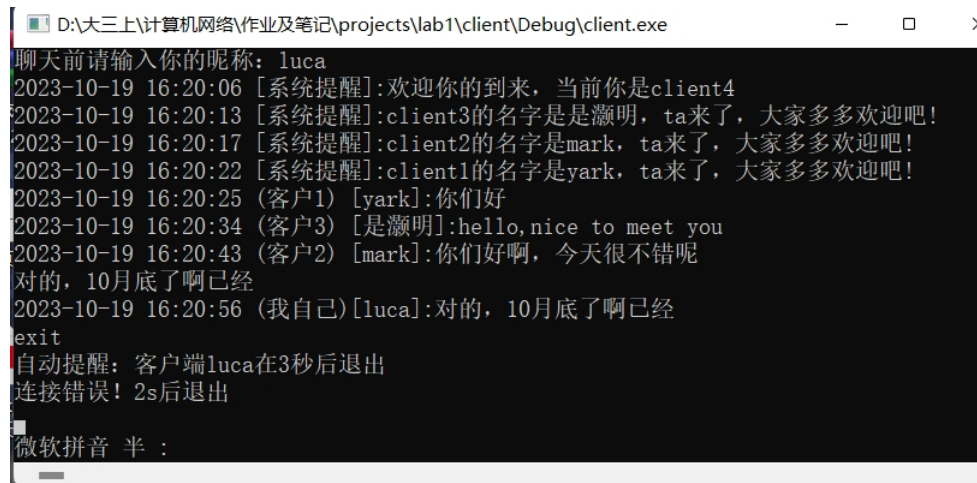
```
D:\大三上\计算机网络\作业及笔记\projects\lab1\server\Debug\server.exe
client_total_num(当前客户人数):2
client_total_num(当前客户人数):3
2023-10-19 16:09:14 [系统提醒]:client2的名字是WentWorth, ta来了, 大家多多欢迎吧!
2023-10-19 16:09:24 (客户2) [WentWorth]:hello, everybody!
2023-10-19 16:09:34 [系统提醒]:client3的名字是李凡, ta来了, 大家多多欢迎吧!
2023-10-19 16:09:38 (客户3) [李凡]:你们好!
2023-10-19 16:09:46 (客户3) [李凡]:今天是个不错的日子
client_total_num(当前客户人数):4
2023-10-19 16:16:13 [系统提醒]:client4的名字是momo, ta来了, 大家多多欢迎吧!
client_total_num(当前客户人数):5
2023-10-19 16:16:29 [系统提醒]:client5的名字是楔子昂, ta来了, 大家多多欢迎吧!
2023-10-19 16:16:43 (客户5) [楔子昂]:the day is very good.i like it
已达到最大连接数, 拒绝新连接

微软拼音 半 :
```

图 5.13: 人数限制 _server 端

5. 程序正确的退出方式

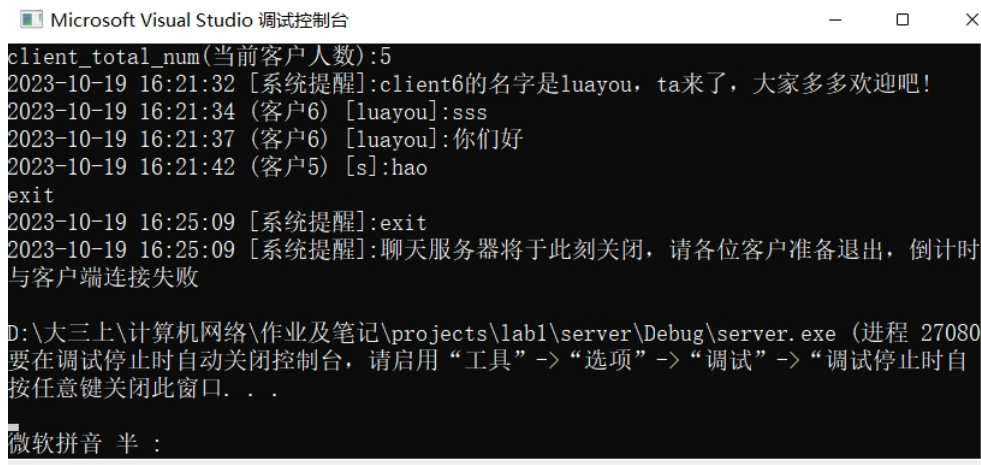
可以看到客户端使用 `exit` 后, 自动退出



```
D:\大三上\计算机网络\作业及笔记\projects\lab1\client\Debug\client.exe
聊天前请输入你的昵称: luca
2023-10-19 16:20:06 [系统提醒]:欢迎你的到来, 当前你是client4
2023-10-19 16:20:13 [系统提醒]:client3的名字是是灏明, ta来了, 大家多多欢迎吧!
2023-10-19 16:20:17 [系统提醒]:client2的名字是mark, ta来了, 大家多多欢迎吧!
2023-10-19 16:20:22 [系统提醒]:client1的名字是yark, ta来了, 大家多多欢迎吧!
2023-10-19 16:20:25 (客户1) [yark]:你们好
2023-10-19 16:20:34 (客户3) [是灏明]:hello,nice to meet you
2023-10-19 16:20:43 (客户2) [mark]:你们好啊, 今天很不错呢
对的, 10月底了啊已经
2023-10-19 16:20:56 (我自己)[luca]:对的, 10月底了啊已经
exit
自动提醒: 客户端luca在3秒后退出
连接错误! 2s后退出
微软拼音 半 :
```

图 5.14: 客户端使用 exit 退出

系统端使用 exit 后, 强制所有客户以及系统窗口一起退出



```
Microsoft Visual Studio 调试控制台
client_total_num(当前客户人数):5
2023-10-19 16:21:32 [系统提醒]:client6的名字是luayou, ta来了, 大家多多欢迎吧!
2023-10-19 16:21:34 (客户6) [luayou]:sss
2023-10-19 16:21:37 (客户6) [luayou]:你们好
2023-10-19 16:21:42 (客户5) [s]:hao
exit
2023-10-19 16:25:09 [系统提醒]:exit
2023-10-19 16:25:09 [系统提醒]:聊天服务器将于此刻关闭, 请各位客户准备退出, 倒计时
与客户端连接失败
D:\大三上\计算机网络\作业及笔记\projects\lab1\server\Debug\server.exe (进程 27080)
要在调试停止时自动关闭控制台, 请启用“工具”->“选项”->“调试”->“调试停止时自
按任意键关闭此窗口. . .
微软拼音 半 :
```

图 5.15: 系统端使用 exit 退出