南开大学
Nankai University

南　　开　　大　　学

计　算　机　学　院

计算机网络实验报告

Lab1：Socket 编写一个聊天程序

冯思程 2112213

年级：2021 级

专业：计算机科学与技术

指导教师：吴英、文静静

2023 年 10 月 20 日

摘要

本次实验，根据要求自行设计了较为完整的应用层聊天协议，利用流式 socket 调用传输层接口实现一个网络聊天程序，实现了实验要求中的全部要求。利用多线程的方法为每个用户建立一个接收线程，服务器为每个用户建立一个线程实现通信。通过设定消息格式与解析算法实现全体信息和针对特定用户的信息发送，创新的通过增加一个线程实现服务器端随时可以通过 quit 结束。

关键字：流式 Socket、多线程、协议、聊天室

目录

一、 预备工作及实验环境

(一) 实验要求与功能

实验要求的基础功能与要求:

1. 给出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. 使用流式套接字、采用多线程（或多进程）方式完成程序。
4. 程序应该有基本的对话界面，但可以不是图形界面。程序应该有正常的退出方式。
5. 完成的程序应该支持多人聊天，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。
7. 在实验中观察是否有数据丢失，提交程序源码和实验报告。

合理的自行扩展功能:

1. 可以通过特定信息格式实现针对特定用户的信息转发。(即类似可以双人私聊)
2. 在遇到错误的时候，可以准确输出报错信息。
3. 在正常关闭服务器后，可以自动让所有在线用户在三秒后自动退出服务器。

(二) 实验环境与说明

具体的实验环境配置如下:

Windows 版本	vs code 版本	g++ 版本
windows11	1.82.0	g++.exe (x86_64-win32-seh-rev0) 8.1.0

表 1: 实验环境说明表

感谢老师与助教的审查批阅与指正，辛苦!

二、 实验过程

(一) 协议设计

本次实验采用多线程、流式套接字方式完成，采用 TCP 协议，这部分的协议不进行赘述。这里注意编码应该是 GBK 格式。

1. 语法与语义协议设计

为了方便应用层与传输层之间的传输，我设计了消息的封装格式，即消息的语法。在该聊天室程序的两端 server 和 client 都遵循该聊天协议，符合协议的对等性。

消息整体最大长度为 128 个字节，其中 1 字节用于装载结束符号，所以整体消息具体内容最大是 127 个字节，不可以超过，否则会触发错误检测。

协议规定，加入聊天室的用户的用户名最大字节长度是 9 字节，不可以超过，否则会触发错误检测。

协议规定，共有五种消息类型，其格式与对应语义分别如下：

1. 用户首次加入龙神聊天室服务器的时候，连接成功后，client 会首先要求用户输入一个合法的用户名并发送给 server 端，这个消息类型的格式如下：
   [**username**]

2. 用户在退出聊天室的时候，会发送一个 **quit** 消息给服务器，表明自己退出服务器，这个消息类型格式如下：
   quit

3. 服务器也可以主动合法的退出，在服务器端主动输入 **quit** 消息后，服务器会发送一条消息给所有连接的用户端表明服务器端关闭，这条消息的格式如下：
   龙神服务器端已关闭

4. 除了上述的特殊消息格式,平时主要用来发送的消息格式由两段组成,由 **target** 和 **content** 组成，中间用英文冒号分隔开，其中 target 表示消息发送的目标，content 表示消息的具体内容。格式如下：(这里为了协议完整性，**特加规则**：如果用户试图利用单纯的 content 格式输入信息，即默认群发，如果在信息中有需要用到冒号，需使用中文冒号)
   [**target**]:[**content**]
   然后这里我实现了特定用户发送和正常群发两种功能，在特定用户发送的时候，在 target 字段中需要填写**要发送用户的用户名**；在群发的时候，你可以直接输入消息内容 (默认群发)，或者将 target 字段填写成 **all**。所以这里需要用户名不能为 all，这一点我在让用户输入的时候已经提示用户。同时这里协议也规定用户名不可以重合 (这一点默认用户均已知)。
   下面给出一些消息的例子来具体说明：
   1. fsc: 家人你好，我想想问问你在群里刚才说的一些事情（要发送用户的用户名:content）
   2. all: 大家上午好（all:content）
   3. 今天群里好热闹呀（content）

   上面第一条是特定用户发送的消息，下面两条是群发的消息。这样就实现了扩展功能。
   注意：当用户未输入任何内容仅输入回车发送时，不群发此消息但服务器日志会记录该行为。

5. 还有一种消息就是服务器根据目标 (特定用户或者群发) 转发的消息，由发出消息的用户和消息内容组成，这种消息格式如下：
   [**senduser**]: [**content**]（英文冒号后有空格）

2. 时序协议设计

下面以聊天室中只有两人为例进行说明：

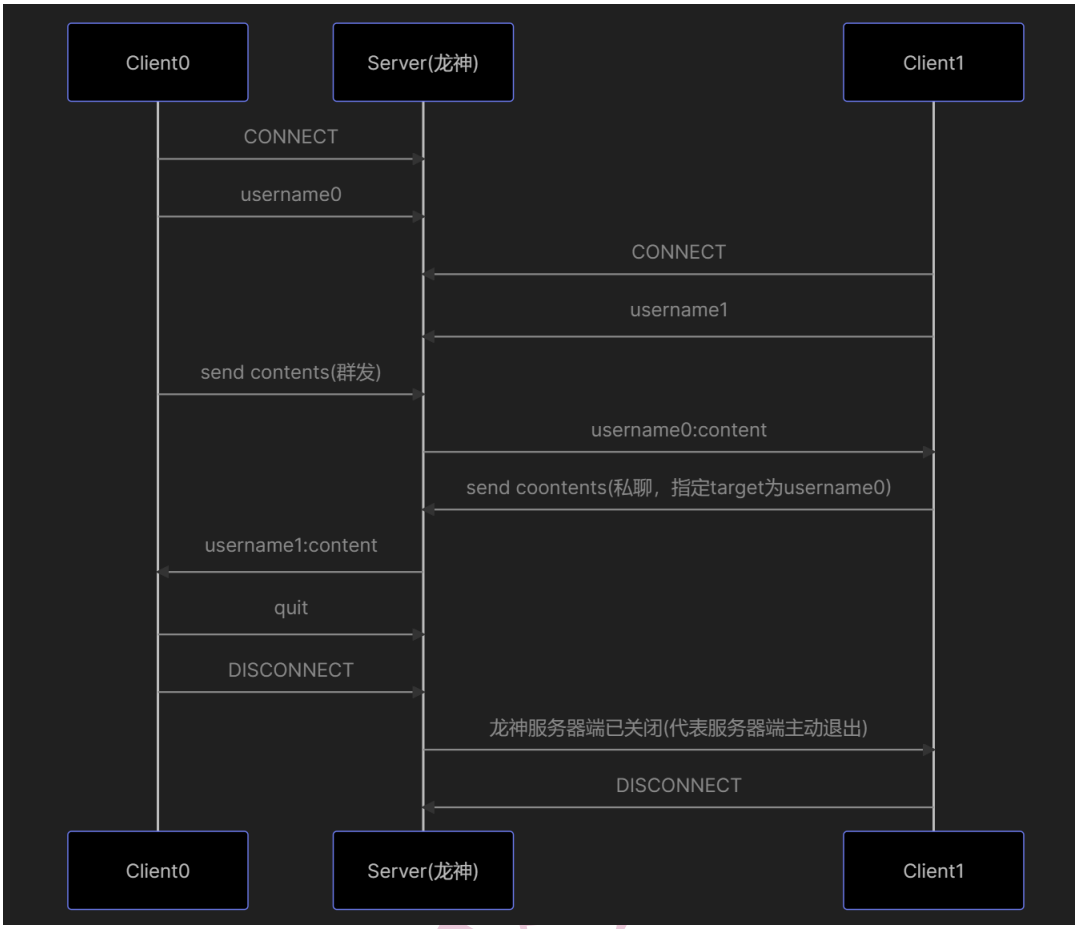

图 1: 时序协议图

首先是 Client0 连接服务器，然后输入自己的用户名 username0 用来标识自己，然后是 Client1 连接服务器，输入自己的用户名 username1 用来标识自己。

Client0 发送一条群发消息，首先发给 Server，通过 Server 的解析，发现这是一条群发消息，然后转发给除了 Client0 之外的所有用户 (这里就是 Client1)。然后 Client1 想向 username0 发送一条私聊消息，首先需要按照上文指定的私聊格式发送给 Server，然后通过 Server 的解析，发现这是一条私聊消息，然后只转发给指定发送目标用户 (这里是 username0)。

然后 Client0 发送 quit 消息给服务器，表示自己退出服务器，同时服务器日志会记录其退出，然后服务器端命令行输入 quit(即服务器关闭)，服务器会发送一条 **龙神服务器端已关闭**给所有在线用户 (这里是 Client1)，然后所有在线用户会等待三秒后自动退出聊天室程序。

(二) 程序设计

下面会展示并说明代码中的核心模块。

1. 服务器端

客户端结构体 定义客户端信息结构体，依次声明客户端套接字、数据缓冲区、用户名、IP、客户端标志成员、用于标识转发的范围 (用于实现特定用户转发功能)。

客户端结构体

```
//定义客户端信息结构体，依次声明客户端套接字、数据缓冲区、用户名、IP、客户端标志成员、用于标识转发的范围(用于实现特定用户转发功能)
struct Client
{
	SOCKET ClientSocket;
	char buffer[128];
	char username[10];
	char IP[20];
	UINT_PTR flag;
	char targetarea[20];
} realclient[20];    //创建一个实例，最多同时容纳20个用户同时在线
```

处理连接线程函数 调用 accept 函数等待客户端的连接，通过 accept 函数创建新的套接字。接收用户名并输出同时也记录下用户名，然后记录下用户 IP，并使用 socket 描述符作为一个独特的标识符来标记或区分不同的客户端。然后遍历其他客户端并创建进程，对于状态改变的客户端，会重新创建接受消息线程。

处理连接线程函数

```
DWORD WINAPI acceptthread(void* data)
{
	int flag[20] = { 0 };//标志数组，一一对应客户端，代表其状态
	while (true)//服务器一直等待客户端的连接
	{
		if (realclient[i].flag != 0)    //找到空闲客户端位置
		{
			i++;
			continue;
		}

		/*调用accept函数等待客户端连接，其会阻塞进程，这里通过accept函数创建的新套接字
		不需要使用过bind函数绑定，因为新套接字会自动集成accept函数提供的服务器地址信息*/
		if ((realclient[i].ClientSocket = accept(ServerSocket, (SOCKADDR*)&caddr, &caddrlen)) == INVALID_SOCKET)
		{
			char errMsg[512];
			FormatMessageA(
				FORMAT_MESSAGE_FROM_SYSTEM | FORMAT_MESSAGE_IGNORE_INSERTS,
				NULL,
				WSAGetLastError(),
				MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),
				errMsg,
				sizeof(errMsg),
				NULL
			);
```

```
                cout << "[龙神]Accept错误，报错信息：" <<errMsg<<
                    endl;
                closesocket(ServerSocket);
                WSACleanup();
                return -1;
            }

            //接收用户名并输出，然后记录下用户IP，并使用socket描述符作为
                一个独特的标识符来标记或区分不同的客户端
            recv(realclient[i].ClientSocket, realclient[i].username,
                sizeof(realclient[i].username), 0);
            cout << "[龙神]用户[" << realclient[i].username << "]" << "
                连接成功" << endl;
            memcpy(realclient[i].IP, inet_ntoa(caddr.sin_addr), sizeof(
                realclient[i].IP));
            realclient[i].flag = realclient[i].ClientSocket;
            i++;

            //遍历其他客户端并创建进程，对于状态改变的客户端，会重新创建
                接受消息线程
            for (int j = 0; j < i; j++)
            {
                if (realclient[j].flag != flag[j])
                {
                    if (HandleR[j])
                        CloseHandle(HandleR[j]);
                    HandleR[j] = CreateThread(NULL, 0, (
                        LPTHREAD_START_ROUTINE)resendthread, &
                        realclient[j].flag, 0, 0);
                }
            }
            for (int j = 0; j < i; j++)
                flag[j] = realclient[j].flag;//同步flag，防止线程重复
                    启动
            Sleep(500);
        }
        return 0;
}
```

消息接收转发线程函数 首先是查找到发送信息的客户端线程，然后对消息进行拆包并解析，这里实现了不同功能的转发 (群发和私聊)，通过第一个英文冒号进行分解成 target 和 content 字段，并根据情况进行消息转发。这里也可以检测到来自客户端发送的 quit 消息，接收到 quit 消息则清除其套接字并会关闭对应客户端的线程。

消息接收转发线程函数

```
//接收并发送数据的线程函数，WINAPI表示这个函数的调用约定是标准调用约定
DWORD WINAPI resendthread(void* data)
```

```
{
        bool first = true;
        SOCKET client = INVALID_SOCKET;
        int flag = 0;
        for (int j = 0; j < i; j++) {
                if (*(int*)data == realclient[j].flag)//找到那个接受线程的客户端
                {
                        client = realclient[j].ClientSocket;
                        flag = j;
                }
        }
        char temp[128] = { 0 };  //声明一个临时字符数组存消息
        string target, content;//消息可以分两段，发送目标用户和内容。
        while (true)
        {
                memset(temp, 0, sizeof(temp));//每次循环会清空temp
                if (recv(client, temp, sizeof(temp), 0) == SOCKET_ERROR){
                        continue;
                }
                //获取第一个英文冒号前的内容为发送目标用户，冒号后是信息内容
                string contents = temp;
                target = contents.substr(0, contents.find(':'));
                content = contents.substr(contents.find(':') + 1 == 0 ?
                    contents.length() : contents.find(':') + 1);
                if (content.length() == 0)//这表示contents中无英文冒号，此时target中存了完整的contents
                {
                        strcpy(realclient[flag].targetarea, "all");
                        strcpy(temp, target.c_str());
                }
                else
                {
                        strcpy(realclient[flag].targetarea, target.c_str());
                        strcpy(temp, content.c_str());
                }

                memcpy(realclient[flag].buffer, temp, sizeof(realclient[flag
                    ].buffer));//把内容存到buffer成员中

                if (strcmp(temp, "quit") == 0)   //这里是为了实现用户可以正常退出聊天室的功能，如果检测到用户发送quit请求，那么直接关闭线程，不打开转发线程
                {
                        //依次关闭套接字、线程句柄，并把位置空出留给以后进入的线程使用
                        closesocket(realclient[flag].ClientSocket);
                        CloseHandle(HandleR[flag]);
```

```
44                  realclient[flag].ClientSocket = 0;
45              HandleR[flag]=NULL;
46                  cout << "[龙神]用户[" << realclient[flag].username
                        << "]" << "离开龙神聊天室" << endl;
47              }
48              else if (first == true)//确保一个用户第一次连接的时候消息不会
                    被转发, 因为这次连接是发送用户名
49              {
50                  first = false;
51                  continue;
52              }
53              else
54              {
55                  //格式化输出时间, 以及用户名和发送的消息内容
56                  time(&t);
57                  strftime(str, 20, "%Y-%m-%d %X", localtime(&t));
58                  cout << '(' << str << ")  [" << realclient[flag].
                        username << "]:" << temp << endl;
59
60                  char temp[128] = { 0 };
61                  memcpy(temp, realclient[flag].buffer, sizeof(temp));
                        //复制内容
62                  sprintf(realclient[flag].buffer, "%s: %s", realclient
                        [flag].username, temp); //把发送信息的用户名添加
                        进转发的信息里
63                  if (strlen(temp) != 0) //如果数据不为空则转发
64                  {
65                      // 向所有用户发送(群发), 即向除自己之外的所有
                            客户端发送信息
66                      if (strcmp(realclient[flag].targetarea, "all"
                            ) == 0)
67                      {
68                          for (int j = 0; j < i; j++){
69                              if (j != flag){
70                                  if (send(realclient[j
                                        ].ClientSocket,
                                        realclient[flag].
                                        buffer, sizeof(
                                        realclient[j].
                                        buffer), 0) ==
                                        SOCKET_ERROR){
71                                      return -1;
72                                  }
73                              }
74                          }
75                      }
76                      // 向特定用户发送(私聊)
77                      else{
```

```
78                                  for (int j = 0; j < i; j++){
79                                      if (strcmp(realclient[flag].
                                            targetarea, realclient[j
                                            ].username) == 0){
80                                          if (send(realclient[j
                                                ].ClientSocket,
                                                realclient[flag].
                                                buffer, sizeof(
                                                realclient[j].
                                                buffer), 0) ==
                                                SOCKET_ERROR){
81                                                  return -1;

82                                          }
83                                      }
84                                  }
85                              }
86                          }
87                      }
88              }
89              return 0;
90      }
```

main 函数中的错误处理–以 bind 阶段为例 这里用 **WSAGetLastError** 函数获取到报错代码，然后利用 **FormatMessageA** 函数进行错误信息的格式化，然后输出。

main 函数的错误处理–以 bind 阶段为例

```
1       //bind阶段，将服务器套接字与地址端口绑定，下文包括错误处理
2       if (SOCKET_ERROR == bind(ServerSocket, (SOCKADDR*)&saddr, sizeof(
            saddr)))
3       {
4               char errMsg[512];
5       FormatMessageA(
6           FORMAT_MESSAGE_FROM_SYSTEM | FORMAT_MESSAGE_IGNORE_INSERTS,
7           NULL,
8           WSAGetLastError(),
9           MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),
10          errMsg,
11          sizeof(errMsg),
12          NULL
13      );
14              cout << "[龙神]␣Bind绑定出现错误，报错信息：" <<errMsg<< endl
                    ;
15              closesocket(ServerSocket);//这里发生错误需要额外关闭套接字
16              WSACleanup();
17              return -1;
18      }
```

main 函数中实现服务器端主动退出功能 在服务器命令行中输入 **quit** 即可实现服务器主动关闭，在输入 quit 之后，会发送一条 **龙神服务器端已关闭**消息到所有在线客户端，然后依次关闭套接字，线程句柄等，最后关闭服务器。同时客户端接收到服务器发来的服务器关闭消息后会识别出来并让客户端在三秒后自动退出。

main 函数中实现服务器端主动退出功能

```
        //实现实验要求：服务器可以主动退出，输入quit即可关闭服务器
        char ssignal[8];
        cin.getline(ssignal, 8);
        if (strcmp(ssignal, "quit") == 0)
        {
                cout << "[龙神]龙神服务器准备关闭" << endl;
                const char* cshutsignal = "龙神服务器端已关闭";
                for (int j = 0; j <= i; j++)
                {
                        if (realclient[j].ClientSocket != INVALID_SOCKET){
                                //对于每个有效客户端，发送服务器关闭消息后关闭连接
                                send(realclient[j].ClientSocket, cshutsignal,
                                     strlen(cshutsignal), 0);
                                closesocket(realclient[j].ClientSocket);
                        }
                }
                CloseHandle(Handle);
                closesocket(ServerSocket);
                WSACleanup();
                cout << "[龙神]龙神服务器已经关闭" << endl;
        }
```

客户端相应代码

```
// 检查服务器是否发送关闭信号
                        if (strcmp(bufferget, "龙神服务器已关闭") == 0)
                        {
                                cout << "龙神服务器已关闭，三秒后该界面自动退出" << endl;
                                closesocket(*(SOCKET*)data);
                                break;
                        }
```

2. 客户端

接收线程函数 其会不断等待接收信息，如果接收到非服务器关闭的消息后，会按照格式输出时间，发送用户和消息内容，这里需要每次利用光标迁移去覆盖掉之前写出的等待用户输入信息的字符串，为了美观。

接收线程函数

```
//接收线程
```

```
DWORD WINAPI rdeal(void* data)
{
        char bufferget[128] = { 0 };
        while (true)//一直等待接收
        {
                if (recv(*(SOCKET*)data, bufferget, sizeof(bufferget), 0) ==
                    SOCKET_ERROR){
                        break;
                }
                if (strlen(bufferget) != 0)
                {
                        // 检查服务器是否发送关闭信号
                        if (strcmp(bufferget, "龙神服务器端已关闭") == 0)
                        {
                                cout << "龙神服务器已关闭，三秒后该界面自动退
                                    出" << endl;
                                closesocket(*(SOCKET*)data);
                                break;
                        }
                        //'\b'光标迁移，功能是清除上一行输出的等待用户输入信
                            息的字符串，例如：(2023-10-17 20:24:03) [fsc]：
                        for (int i = 0; i <= strlen(username) + 50; i++){
                                cout << "\b";
                        }
                        // 打印时间以及接收到的消息
                        time(&t);
                        strftime(str, 20, "%Y-%m-%d %X", localtime(&t));
                        cout << '(' << str << ")     " << bufferget << endl;
                        //刚才把输出等待用户输入的字符串给退回了，这里需要再
                            把这个字符串打印出来
                        cout << '(' << str << ")  [" << username << "]: ";
                }
        }
        Sleep(3000);
        exit(1);
        return 0;
}
```

客户端建立过程 步骤：初始化套接字 -> 创建套接字 -> 请求连接服务器，在每一步中我都加入了错误处理，方便进行调试，提升用户友好性。在设定所连接服务器端口号时需要参照服务器设置的端口号一致，IP 地址用本机 IP 地址 127.0.0.1 。

客户端建立过程

```
WSADATA wd = { 0 };//存放套接字信息
        SOCKET ClientSocket = INVALID_SOCKET;//客户端套接字

        //用于初始化套接字，请求是2.2winsock版本，返回非零值表示初始化失败，
```

```
    下文代码包含初始化错误处理
if (WSAStartup(MAKEWORD(2, 2), &wd))
{
        char errMsg[512];
FormatMessageA(
    FORMAT_MESSAGE_FROM_SYSTEM | FORMAT_MESSAGE_IGNORE_INSERTS,
    NULL,
    WSAGetLastError(),
    MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),
    errMsg,
    sizeof(errMsg),
    NULL
);
        cout << "[龙神]WSAStartup创建失败, 报错信息: " <<errMsg<<
            endl;
        return -1;
}

/*根据实验要求, 这里我创建了IP地址类型为AF_INET（IPV-4协议）,
服务类型为流式(SOCK_STREAM), 使用TCP协议的套接字*/
ClientSocket = socket(AF_INET, SOCK_STREAM, 0);
//socket函数的错误处理
if (ClientSocket == INVALID_SOCKET)
{
        char errMsg[512];
FormatMessageA(
    FORMAT_MESSAGE_FROM_SYSTEM | FORMAT_MESSAGE_IGNORE_INSERTS,
    NULL,
    WSAGetLastError(),
    MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),
    errMsg,
    sizeof(errMsg),
    NULL
);
        cout << "[龙神]Socket创建错误, 报错信息: " <<errMsg<< endl;
        return -1;
}

/*初始化并设置服务器地址, 并设定监听端口, 使用IPv4地址, 这里地址是我
    设置好的IP地址。*/
SOCKADDR_IN saddr = { 0 };
USHORT port = PORT;
saddr.sin_family = AF_INET;
saddr.sin_port = htons(port);
saddr.sin_addr.S_un.S_addr = inet_addr(IP);

//连接服务器, 包含报错信息
if (SOCKET_ERROR == connect(ClientSocket, (SOCKADDR*)&saddr, sizeof(
```

```
            saddr)))
50      {
51              char errMsg[512];
52      FormatMessageA(
53          FORMAT_MESSAGE_FROM_SYSTEM | FORMAT_MESSAGE_IGNORE_INSERTS,
54          NULL,
55          WSAGetLastError(),
56          MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),
57          errMsg,
58          sizeof(errMsg),
59          NULL
60      );
61              cout << "[龙神]Connect连接错误, 报错信息: " <<errMsg<< endl;
62              closesocket(ClientSocket);
63              WSACleanup();
64              return -1;
65      }
```

首次连接发送用户名 获取到输入的用户名，然后发送给服务器端。

首次连接发送用户名

```
1       // 发送名字
2       string name;
3       cin>>name;
4       while (name.length() > 9) {
5       cout << "输入用户名非法, 请重新输入: ";
6               cin>>name;
7   }
8       strcpy(username, name.c_str());
9       send(ClientSocket, username, sizeof(username), 0);//发送用户名到服务
            器
```

发送消息以及主动退出模块 首先先创建接收线程，然后获取到发送的消息，并发送。如果是 **quit** 消息，则会在发送消息后关闭客户端。这里我进行了消息的超限检查，如果消息长度超限，则会报错超限并拒绝将消息发送给服务器端 (即不会出现在日志中也不会转发给其他用户)。

发送消息以及主动退出模块

```
1       // 创建接收线程
2       HANDLE recvthread = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)
            rdeal, &ClientSocket, 0, 0);
3
4       char bufferSend[128] = { 0 };
5       bool start = false;
6       while (true)
7       {
8               if (start){
9                       //输出
```

```
                time(&t);
                strftime(str, 20, "%Y-%m-%d %X", localtime(&t));
                cout << "(" << str << ")  "<< "[" << username << "] :
                    ";
            }
            start = true;
            // 用户输入发送消息
            cin.getline(bufferSend, 128);

        if (cin.fail()) // 检查是否因为输入超过限制
            {
                cin.clear(); // 清除错误标志
                cin.ignore(numeric_limits<streamsize>::max(), '\n');
                    // 丢弃输入流中的剩余字符
                cout << "输入超限, 请重新输入" << endl;
                continue;
            }

            //如果用户输入quit准备退出
            if (strcmp(bufferSend, "quit") == 0)
            {
                cout << "您已离开聊天室" << endl;
                send(ClientSocket, bufferSend, sizeof(bufferSend), 0)
                    ;
                break;
            }
            send(ClientSocket, bufferSend, sizeof(bufferSend), 0);
        }
        closesocket(ClientSocket);
        CloseHandle(recvthread);
        WSACleanup();
        system("pause");
        return 0;
```

三、 运行测试

(一) 编译过程

这里编译我编写了一个 makefile 来快速编译，makefile 代码如下，这里注意编译这两个 cpp 文件的时候要链接到 lws2_32 第三方库。

makefile

```
.PHONY: server client
server:
        g++ server.cpp -o server -lws2_32
client:
        g++ client.cpp -o client -lws2_32
```

运行 **make server** 和 **make client** 两条命令后就得到了 server 和 client 的 exe 可执行文件，接下来用可执行文件进行测试。

(二) 运行

1. 启动阶段

首先点击 server.exe 可执行文件，看到如下界面，说明服务器端成功启动。

注意：这里如果你想打开第二个 server 是无法打开的，因为端口号是被占用的，即使点击它也会瞬间退出。

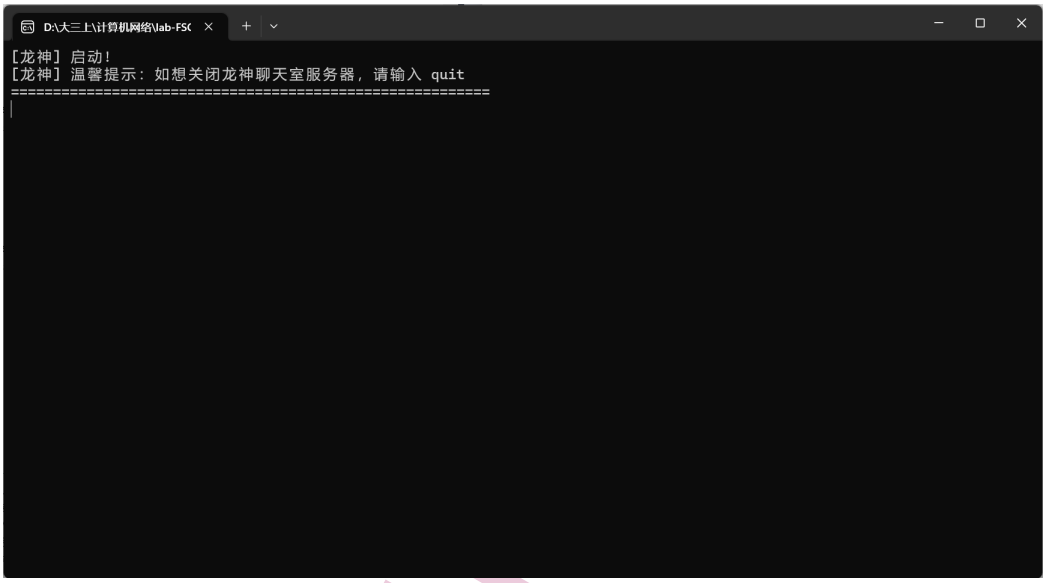

图 2: 服务器端启动

然后点击 client.exe 可执行文件，看到如下界面，说明客户端成功启动。

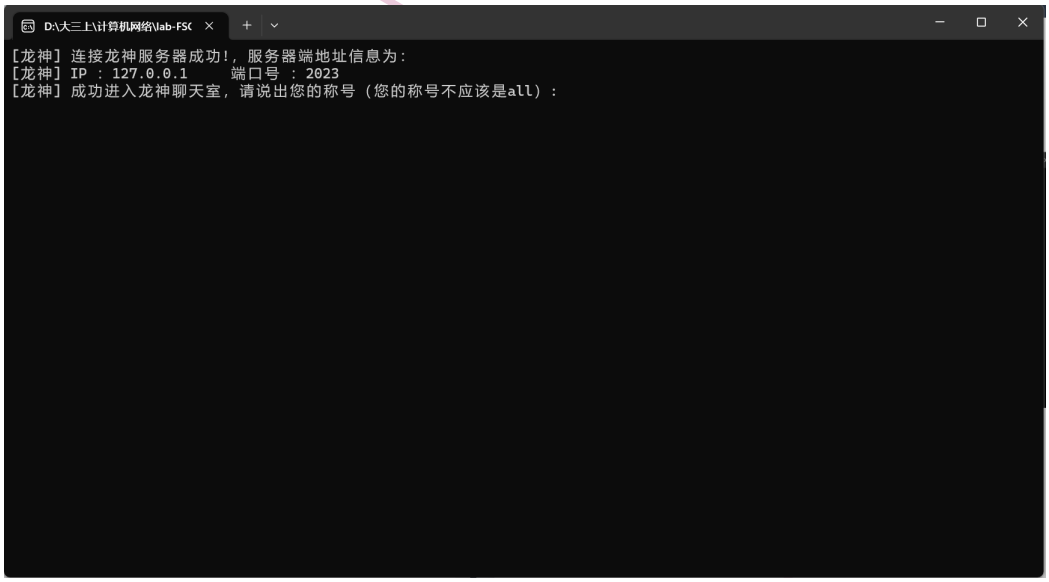

图 3: 客户端启动

首先先输入一个非法的名字，可以看到可以进行合法检测，然后合法的输入用户名，结果如下，可以看到 server 端显示该用户已经成功连接，同时客户端进行输入消息界面，可以开始发送消息：

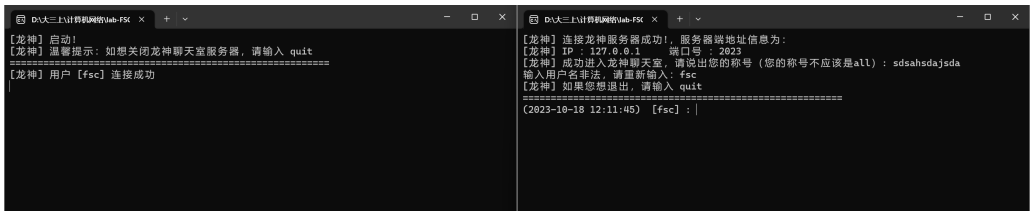

图 4: 用户名输入

然后同理再开启两个 client 端，并输入合法用户名，结果如下：

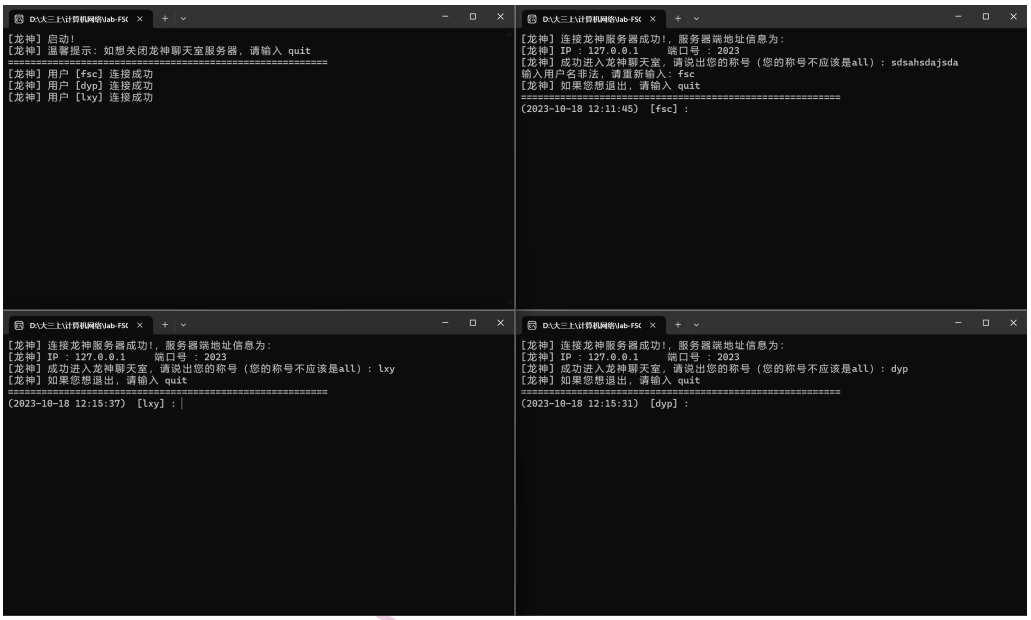

图 5: 三个客户端启动

2. 功能测试

群发功能验证 这里我分别从三名用户处发送了 **大家上午好**和 **all: 你们都是一起学习计算机网络的同学吗**和 **是的呀**三条消息，发现均可以成功按照群发的功能发送给剩余所有用户，而且在服务器日志处，记录下三人的群发记录，结果如下：

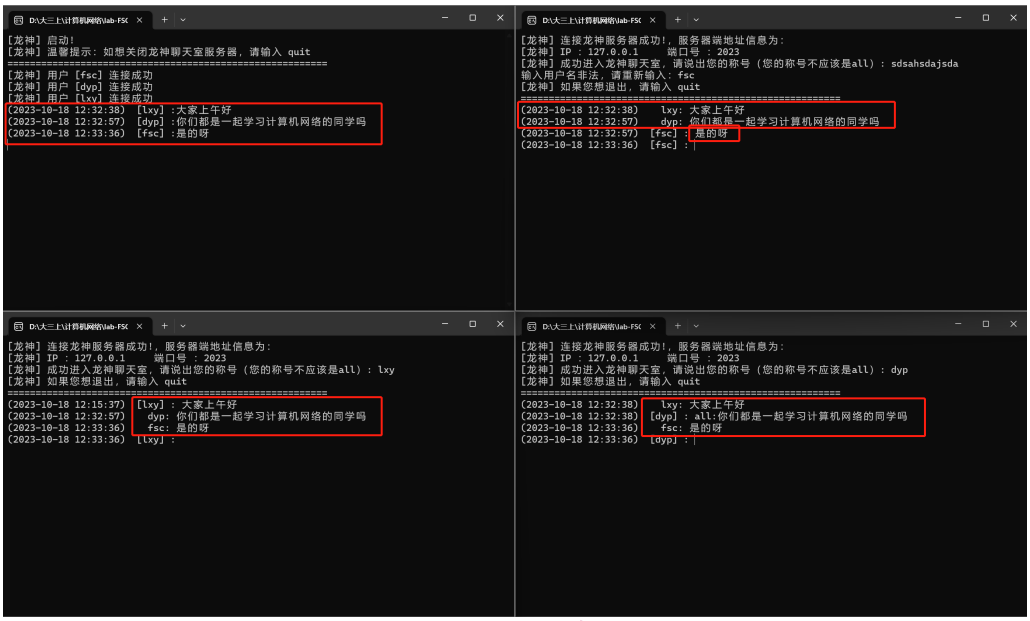

图 6: 群发结果

特定用户发送功能验证 这里我从 fsc 向 dyp 发送了 **dyp: 你早上吃了什么**和 **dyp: 我吃的是粥和油条**两条消息，然后从 dyp 向 fsc 发送了 **fsc: 我没吃早饭，没来得及**一条消息。均可以成功按照指定用户发送的功能发送给 dyp，而在用户 lxy 的界面处无法看到这条消息。而且在服务器日志处，可以记录下特定用户发送记录，结果如下:

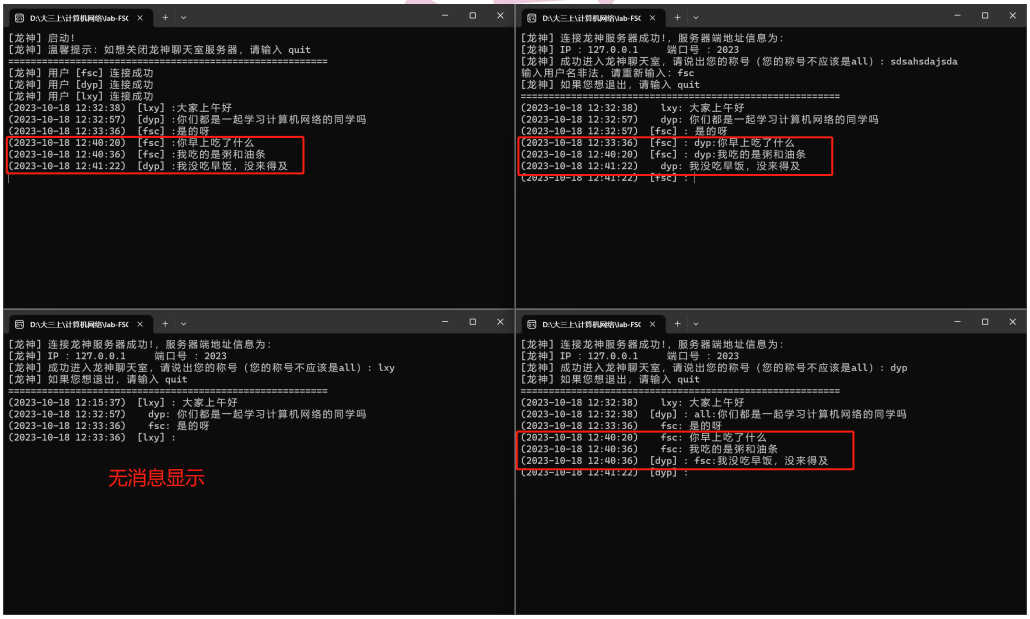

图 7: 特定用户发送结果

空白输入测试 这里我连续发送多条空白消息，发现可以在服务器日志处记录，但是不会进行转发，正确实现功能，结果如下:

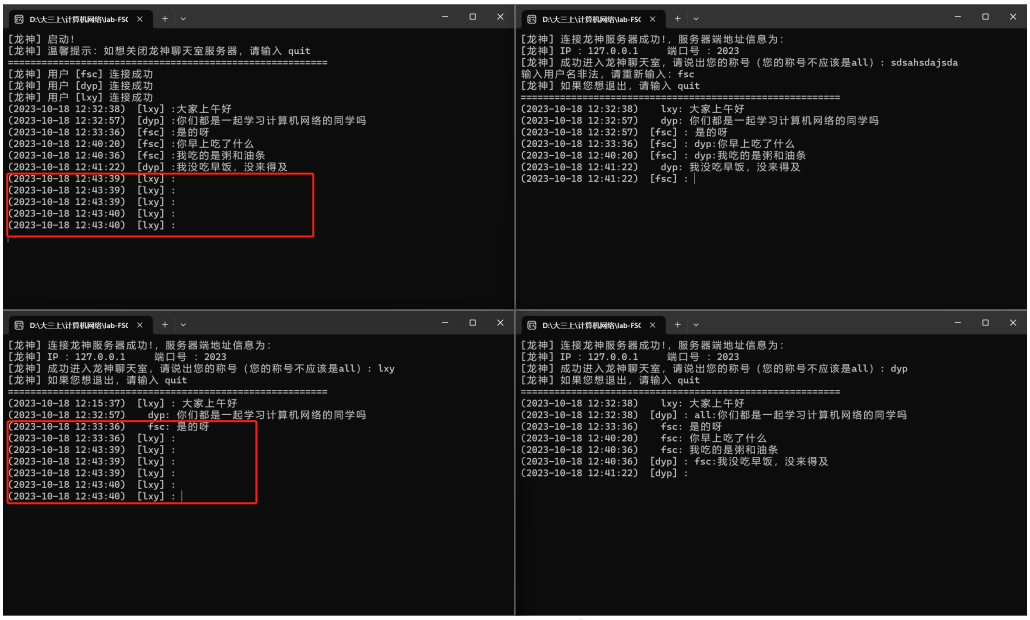

图 8: 空白输入验证结果

3. 退出测试

用户退出测试 这里我将 lxy 用户输入 **quit** 消息进行退出，发现日志成功记录，而且 lxy 客户端成功退出，结果如下:

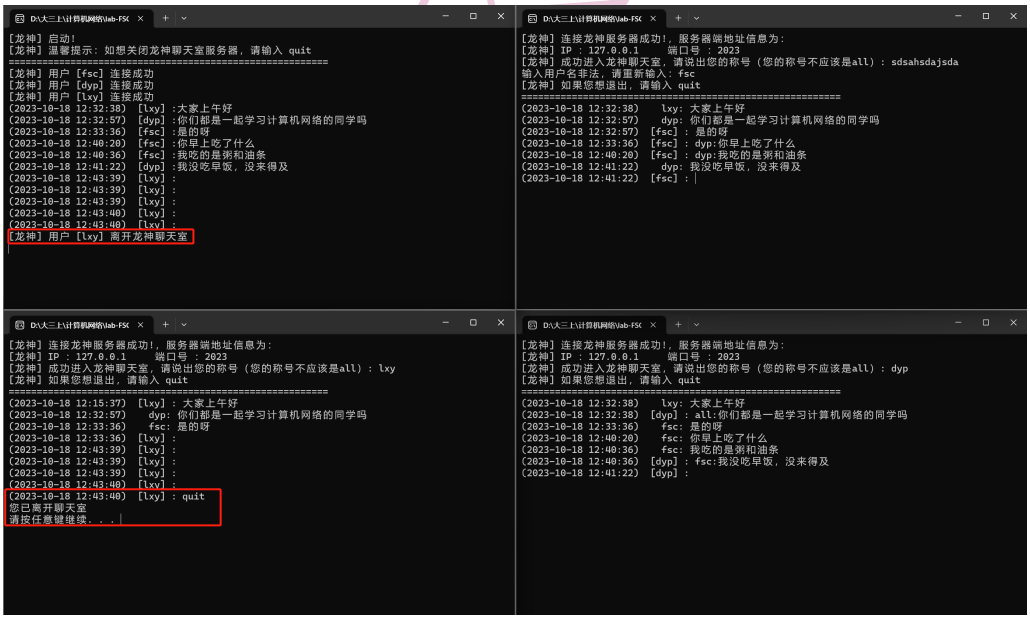

图 9: 用户退出测试

服务器关闭测试 然后这里由于之前的界面被关闭，我又开了界面进行服务器关闭测试，我在服务器命令行中输入 **quit** 消息进行服务器的关闭，发现服务器正确关闭，然后客户端也正确的等待三秒后退出，结果如下：(这里由于服务器是瞬间退出的，没有截到图)

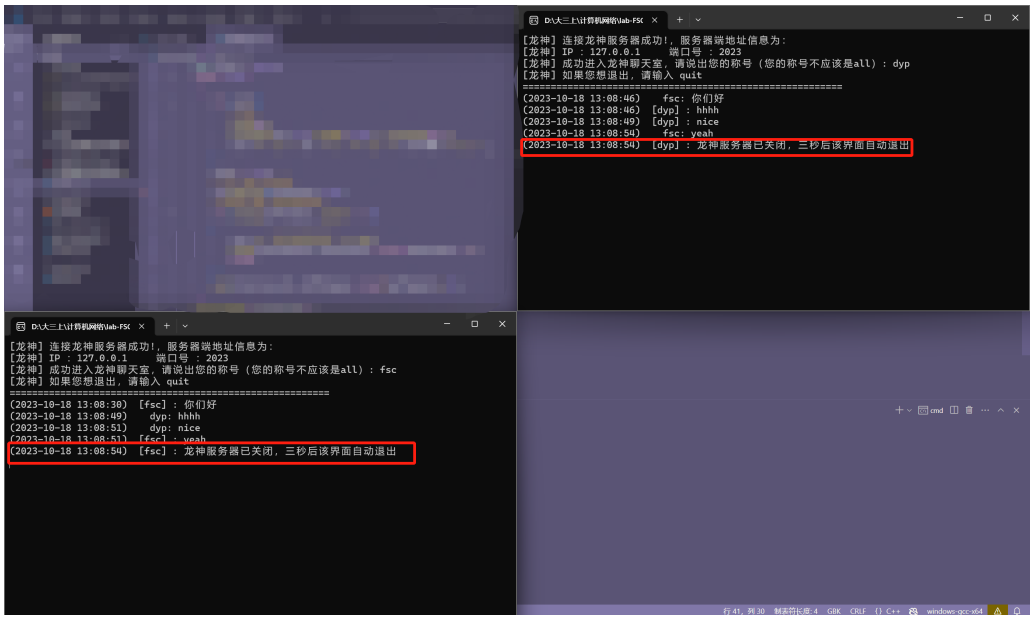

图 10: 服务器关闭测试

4. 补充：英文信息测试

我的聊天室程序是支持英文输入的，可以看到消息正确被转发，而且服务器日志正确记录下了聊天记录，下面是英文测试的结果：

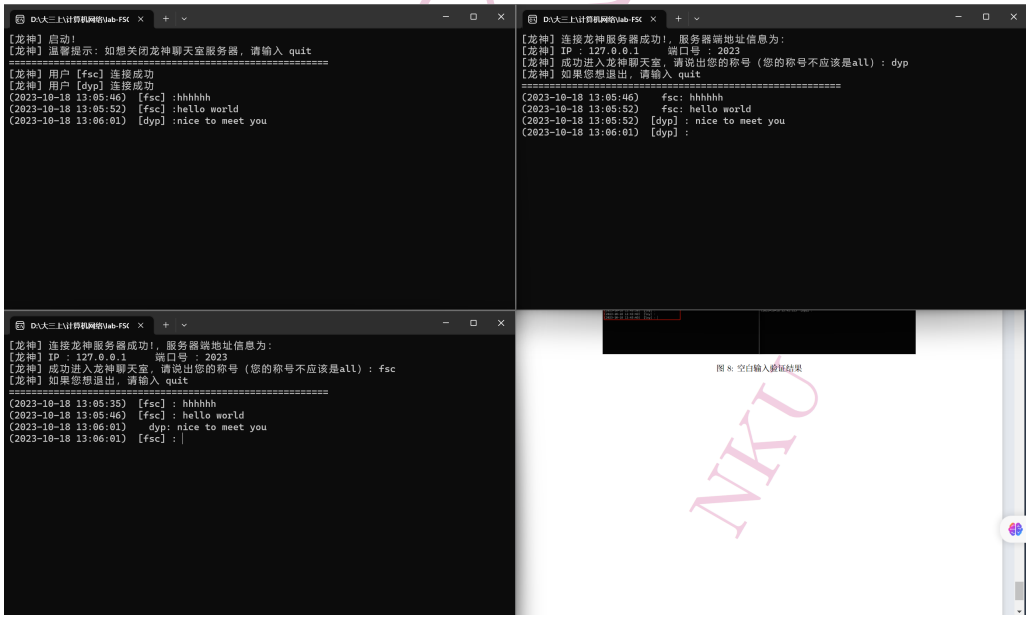

图 11: 英文消息测试

5. 补充：测试超限检查

超限信息会报错并提示用户，而且该超限信息不会被发送到服务器端，也不会被转发给其他用户，结果如下：

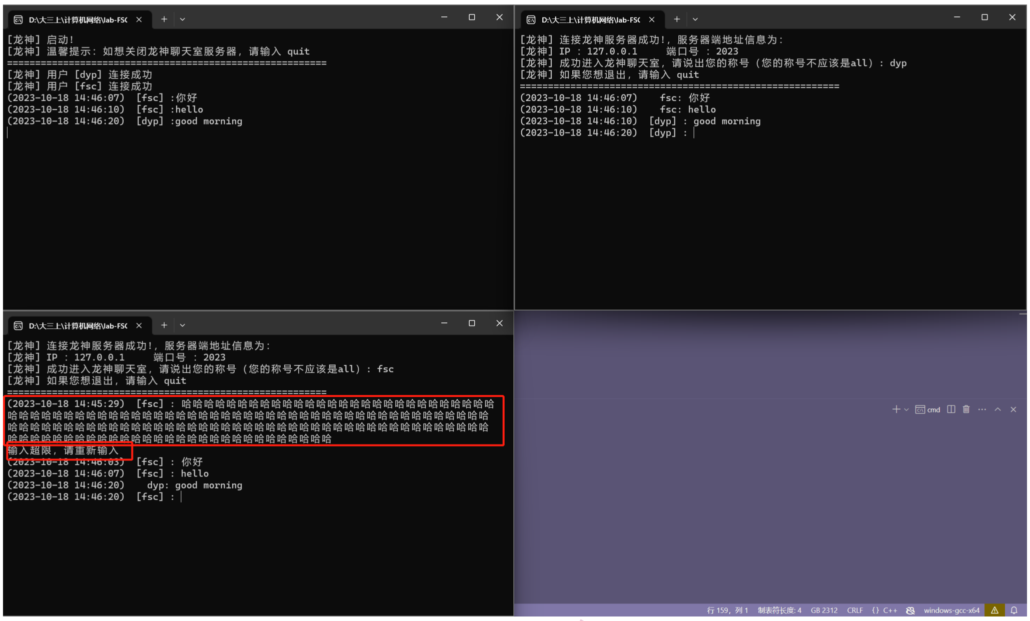

图 12: 超限测试

6. 补充：服务器端输入无效指令

如果服务器端输入了无效指令 (即非 quit)，系统会打印报错信息并自动退出服务器和客户端 (客户端不会打印信息自动退出)，结果如下：

图 13: 服务器无效指令测试

四、 数据丢失测试

根据上文中以及我私下的多次测试，只要是合法的输入都会完整的发送，而且我通过查看服务器的日志，发现合法消息的丢失率为 0，即均能以零丢失率传送信息。下面展示其中一个测试

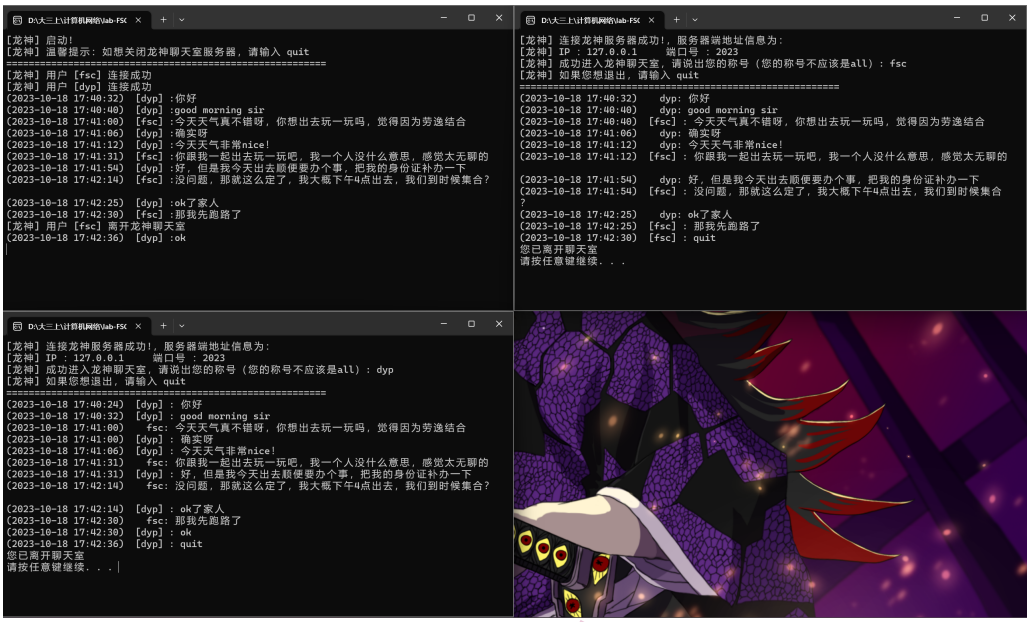

图 14: 数据丢失测试

另一方面，我安装了 **wireshark** 软件，它可以捕获到发送的数据包，下面是找到的包 (通过端口号找到，我端口号设置是 2023)：

```
2023 → 60692 [ACK] Seq=129 Ack=129 Win=8441 Len=0
2023 → 60692 [ACK] Seq=385 Ack=257 Win=8441 Len=0
2023 → 60692 [PSH, ACK] Seq=1 Ack=1 Win=8442 Len=128
2023 → 60692 [PSH, ACK] Seq=129 Ack=129 Win=8441 Len=128
2023 → 60692 [PSH, ACK] Seq=257 Ack=129 Win=8441 Len=128
2023 → 60692 [PSH, ACK] Seq=385 Ack=257 Win=8441 Len=128
2023 → 60696 [ACK] Seq=1 Ack=129 Win=8441 Len=0
2023 → 60696 [ACK] Seq=129 Ack=257 Win=8441 Len=0
2023 → 60696 [ACK] Seq=129 Ack=385 Win=8440 Len=0
2023 → 60696 [ACK] Seq=257 Ack=513 Win=8440 Len=0
2023 → 60696 [PSH, ACK] Seq=1 Ack=129 Win=8441 Len=128
2023 → 60696 [PSH, ACK] Seq=129 Ack=385 Win=8440 Len=128
```

图 15: wireshark 测试

然后发送一条消息并刷新，点击包查看具体内容，发现发送消息的内容完好，即没有丢包现象发生，如下：

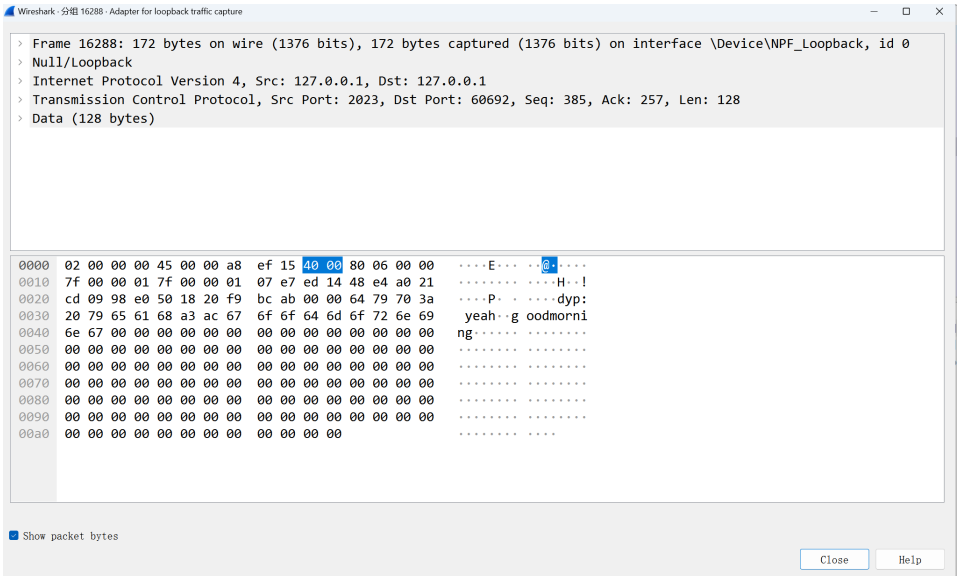

图 16: 消息包详细信息

五、 总结与思考

我在本次实验中，完成了一个聊天室的基础搭建，并根据实验要求进行了适当的合理扩展，在过程中，我也多次遇到 bug，但是我会结合在一些位置打入输出和调试的方法进行 debug，最后成功完成了这次实验，其中在为了实现扩展功能的时候（扩展功能经过询问老师，得知是要有效的扩展，而不是去实现 UI 等无关 socket 编程的扩展，于是我这里选择了解析消息并按模式发送的扩展功能），需要考虑到一些特殊情况，我针对这些特殊情况，设计了代码和协议，保证了协议的完整性和代码的规范有效性。

当然这次实验也只是一个非常基础的聊天室实现，在真正的聊天室中，例如 Discord，功能要复杂繁琐的多，可以在未来继续深入研究。