计算机网络实验一

利用Socket，设计和编写一个聊天程序

学号：2011165 姓名：李欣

一、环境

平台：　　vs2019

程序语言：C++

系统：　　Windows11

二、实现概览

在**本实验**中，实现了可**容纳20人**的**聊天室（多线程）**，能够进行**群发消息**，**私聊**个人，**更改用户名**和**退出。**

1.服务端：

服务端是实现客户端间通信的桥梁，主要承担转发功能，具体就是承担判断客户发来的群聊/私聊/改名/退出等请求，并对这些请求做进一步的处理的任务，在适当的时候输出日志。

2.客户端：

客户端向服务端**发送**群聊消息/私聊消息/退出请求/改名请求，**接收**服务端发来的群聊或私聊信息，需要启动2个线程。

三、协议设计

实验实验传输层使用**TCP**，**数据流形式**传输。

设计了一个**多人的聊天室**，**1个服务器端最多**对应**20个客户端。**

客户端之间的通讯通过服务器，服务器判断客户发来的群聊/私聊/改名/退出等请求，并对这些请求做进一步的处理；客户端接收服务器端传来的消息。

对于每条**消息**，长度**不超过100个字符**，以**换行或'\0'结尾**，**超过**100个字符的部分会被**自动忽略。**

消息的形式：

1、以quit开头：则证明该客户端退出聊天室。在服务端会有对应的输出。

时：分：秒[用户退出]：用户[用户名]退出了群聊

2、以sendone开头

接收缓冲区**第8个字符开始直至空格**为用户需要**私聊的用户**

服务端会有相应的日志记录，除了私聊客户端，其他客户端不会有消息输出。

3、以changename开头

接收缓冲区**第11个字符开始直至空格**为用户需要更改的**新名字**

服务端会有相应的日志输出。

4、其余形式

为群发的消息，在每个客户端会有对应的输出，在服务端也会有相应的日志输出。

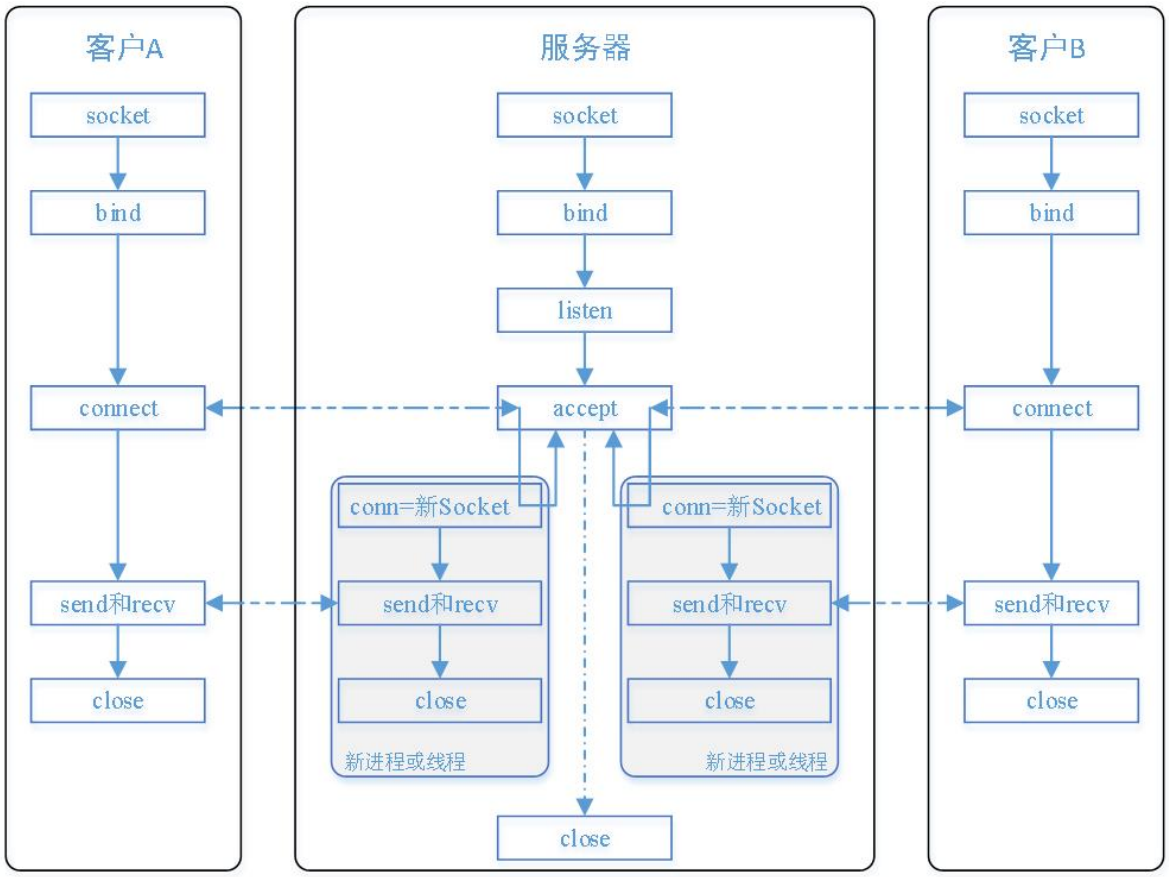
5、每发送成功/失败一条消息，都会有提示

时 ：分 ：秒 [消息发送] ：成功/失败

客户端开启**2个线程**，分别**接收消息**和**发送消息**

服务端由连接的线程数决定（<=20）

流程图：



四、模块功能

1、server端和client端都需要做的操作

根据流程图，我们知道，在server和client端，都要进行socket初始化和bind操作，核心代码如下

SOCKET初始化

```
    //初始化WSDATA
    WSADATA ws;
    //初始化Socket DLL，协商使用的Socket版本
    //MAKEWORD(2, 2) 调用2.2版本，调用希望使用的最高版本
    //ws 可用Socket的详细信息
    int result;
    result = WSAStartup(MAKEWORD(2, 2), &ws);
    //cout << result;
    H_M_S();
    if (result == success)
```

```
{
    cout << " [WSADATA初始化]: 成功,调用的socket最高版本为2.2! " << endl;
}
else
{
    cout << " WSADATA初始化：失败;错误代号: " << WSAGetLastError() << endl;
    WSACleanup();
    return 0;
}

//初始化一个socket
SOCKET s;
//初始化socket，ipv4，数据流，TCP协议
s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
//s的返回值
//0,1,2分别表示标准输入、标准输出、标准错误。
//所以其他打开的文件描述符都会大于2, 错误时就返回 -1. 这里INVALID_SOCKET 也被定义为 -1
H_M_S();
if (s == -1)
{
    cout << " SOCKET创建失败;错误代号: " << WSAGetLastError() << endl;
    WSACleanup();
    return 0;
}
else
    cout << " [SOCKET创建]:    成功! " << endl;
```

bind(执行绑定操作)

```
//绑定bind
//sockaddr_in用于socket定义和赋值;
sockaddr_in sa_in;
//bzero(&sa_in, sizeof(sa_in)); bzero需要在linux下使用！！！
//初始化sa_in
memset(&sa_in, 0, sizeof(sa_in));
H_M_S();
cout << " [输入ip地址]:    使用默认ip请输入localhost: ";
cin >> my_ip;
if ((string)my_ip == "localhost")
{
    memset(&my_ip, 0, sizeof(my_ip));
    strcat(my_ip, "127.0.0.1");
}
//cout << my_ip << endl;
H_M_S();
cout << " [请输入端口号]:  请在1024-5000之间选择: ";
while (1)
{

    int cin_port;
    cin >> cin_port;
    if (cin_port > 5000 || cin_port < 1024)
    {
        H_M_S();
        cout << " 端口不在支持范围内，请重新输入: ";
    }
    else
```

```
        {
            port = cin_port;
            break;
        }
    }
    //Address family一般来说AF_INET（地址族）PF_INET（协议族）,windows下基本无差别
    sa_in.sin_family = AF_INET;
    //ip_address
    sa_in.sin_addr.s_addr = inet_addr(my_ip);
    //port
    sa_in.sin_port = htons(port);
    cout << endl;
    cout << "[ip地址]:     " << my_ip << endl;
    cout << "[port端口号]: " << port << endl;

    //执行绑定操作
    result = bind(s, (SOCKADDR*)&sa_in, sizeof(SOCKADDR));
    if (result == success)
    {
        cout << "[ip地址]:     绑定成功" << endl;
        cout << "[port端口号]: 绑定成功" << endl;
    }
    else
    {
        closesocket(s);//释放socket
        WSACleanup();//释放dll
        cout << "ip与port绑定失败;错误代号：" << WSAGetLastError() << endl;
        return 0;
    }
```

2、Server端

在**完成绑定**操作**后**，服务端需要开启**监听（listen）功能。**

```
    //监听
    result = listen(s, 20);
    if (result == success)
    {
        cout << "\n现在是: ";
        H_M_S();
        cout << endl;
        cout << "[聊天室初始化]: 成功!" << endl;
        cout << "[容量]:          20人" << endl;
        cout << "[监听]:          开始!" << endl;
    }
    else
    {
        closesocket(s);//释放socket
        WSACleanup();//释放dll
        cout << "监听失败; 错误代号：" << WSAGetLastError() << endl;
        return 0;
    }
```

如果能够开启监听，那么接下来就是**等待客户端的连接请求（accept）**，每接收到一个客户端的连接请求，就**新建一个socket并开启一个线程**与客户端进行对接。（这里仅展示**关键代码**）

```
//对接客户端的socket
    SOCKET c_s;
//等待连接请求
    c_s = accept(s, (SOCKADDR*)&client_sa_in, &addr_temp);
//开启新线程
    HANDLE cthread = CreateThread(NULL, 0, my_thread, LPVOID(c_s), 0, NULL);
```

在此，我认为，**服务器端最重要的任务是完成线程内对消息的处理。**由于本实验的实现主要处理四种消息：群聊消息，私聊消息，改名请求和退出请求。

我们发送的消息是**字符数组**，在网络中以**字符流**的形式传播，根据协议，我们**处理消息的次序依次**为：

退出请求 → 改名请求/私聊消息（优先级相同） → 群发消息

一些重要的变量：

首先我们需要了解到，服务器实现这些功能需要的变量（用于存储重要的信息），如：map类型的sockets，用于存储与服务器连接的客户端对接的所有sockets；其余变量均有注释解释。

```
    char my_ip[100];//ip地址
    int port;//端口号
    char message[100];//发送的消息
    map<string, string>users;//记录客户端的姓名
    map<SOCKET, int>sockets;//记录每个客户端对接的socket
```

时间标签

```
//时间函数，打印当前时间 年月日
void Y_M_D()
{
    SYSTEMTIME y_m_d;
    GetLocalTime(&y_m_d);
    cout << "今天是--" << y_m_d.wYear << "年" << y_m_d.wMonth << "月" <<
y_m_d.wDay << "日" << endl;
}

//时间函数，打印当前时间 时分秒
void H_M_S()
{
    SYSTEMTIME h_m_s;
    GetLocalTime(&h_m_s);
    cout << h_m_s.wHour << ":" << h_m_s.wMinute << ":" << h_m_s.wSecond;
}
```

退出请求：

判断是否为退出请求：

```
bool judgequit(char* c)
{
    if (c[0] == 'q' && c[1] == 'u' && c[2] == 'i' && c[3] == 't' && c[4] ==
'\0')
        return true;
    else
        return false;
}
```

执行退出

执行退出时注意要将退出的线程对应的socket在map中**删除（erase）**掉!

```
        //判断quit break,销毁该socket
        if (judgequit(receivebuffer))
        {
            // mymap[cs] = 0;
            if (users.find(to_string(cs).data()) == users.end())
            {
                H_M_S();
                cout << " [用户退出]: 用户[" << to_string(cs).data() << "]退出了群
聊" << endl;
                sockets.erase(cs);
                break;
            }
            else
            {
                H_M_S();
                cout << " [用户退出]: 用户[" << users[to_string(cs).data()] << "]
退出了群聊" << endl;
                sockets.erase(cs);
                break;
            }
        }
```

改名请求:

判断是否为改名请求

```
//判断是否为改名请求
bool judge_changename(char* c) {
    if (c[0] == 'c' && c[1] == 'h' && c[2] == 'a' && c[3] == 'n' && c[4] == 'g'
&& c[5] == 'e' && c[6] == 'n' && c[7] == 'a' && c[8] == 'm' && c[9] == 'e')
        return true;
    return false;
}
```

改名

```
        //判断改名
        if (judge_changename(receivebuffer))
        {
            cout << endl;
            //新改的名字
            string newName = newname(receivebuffer);
            //记录新名字，insert也可以
            users[to_string(cs).data()] = newName;
            //打印新名字
            H_M_S();
            cout << " [用户改名]: " << to_string(cs).data() << " 修改昵称为 " <<
newName << "" << endl;
            continue;
        }
```

私聊消息：

判断是否为私聊消息：

```
//判断是否为私聊请求
bool judge_sendone(char* buff) {
    if (buff[0] == 's' && buff[1] == 'e' && buff[2] == 'n' && buff[3] == 'd' &&
buff[4] == 'o' && buff[5] == 'n'&&buff[6]=='e')
        return true;
    return false;
}
```

发送私聊消息：

这里展示关键代码，主要执行逻辑为在sockets数组中根据用户名找到要私聊发送的客户端，并将消息传递给它。

```
for (auto it : sockets)
{
    if (users[to_string(it.first).data()] == s)
    {
    //……对sendbuffer进行一些处理……
        send(it.first, sendbuffer, 100, 0);
    }
}
```

群发消息：

如果不是上述消息的任何一种，那么定义该类消息为群聊消息。

关键是要对服务端连接的每个客户端（除发送群聊消息的客户端）发送消息。

关键代码

```
            for (auto it : sockets) {
                if (it.first != cs && it.second == 2) {
                    auto ans = send(it.first, sendbuffer, 100, 0);
                    if (ans == SOCKET_ERROR) {
                        cout << "[群聊信息]: " << it.first << "发送失败，" << "错误
代号：  " << endl;
                    }
                }
            }
```

3、Client端

在**完成绑定**操作**后**，客户端需要开启**连接（connect）请求。**

```
    //执行连接操作
    result = connect(s, (SOCKADDR*)&sa_in, sizeof(SOCKADDR));
```

完成连接操作后，开启两个线程：消息接收线程和消息发送线程。

```
    HANDLE t[2];
    t[0] = CreateThread(NULL, 0, re_ma, (LPVOID)&s, 0, NULL);
    t[1] = CreateThread(NULL, 0, se_ma, (LPVOID)&s, 0, NULL);
    WaitForMultipleObjects(2, t, TRUE, INFINITE);
```

一些重要的变量：

```
    char receivebuffer[100];//接收缓冲区
    char sendbuffer[100];//发送缓冲区
```

消息接收线程：

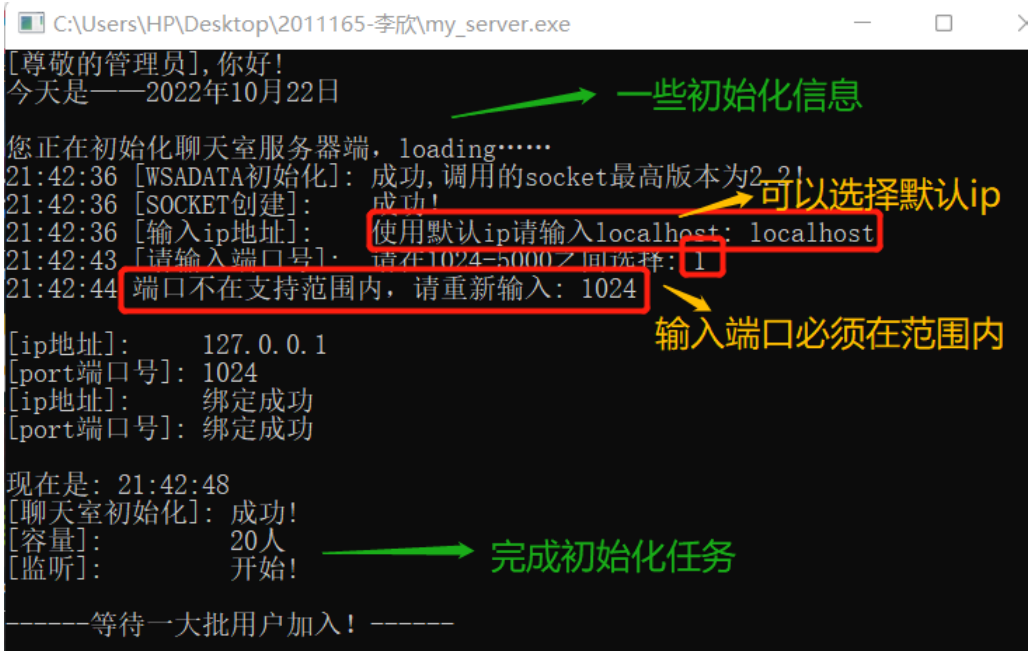
```
    DWORD WINAPI re_ma(LPVOID lparam) {
        SOCKET* s = (SOCKET*)lparam;
        int rflag;
        while (true) {
            rflag = recv(*s, receivebuffer, 100, 0);
            if (rflag > 0 && flag) {
                cout << endl;
                H_M_S();
                cout << " [新消息]:   ";
                cout << receivebuffer;
                cout << "    ---请积极发言" << endl;;
            }
            else {
                closesocket(*s);
                return 0;
            }
        }
    }
```

消息发送线程：

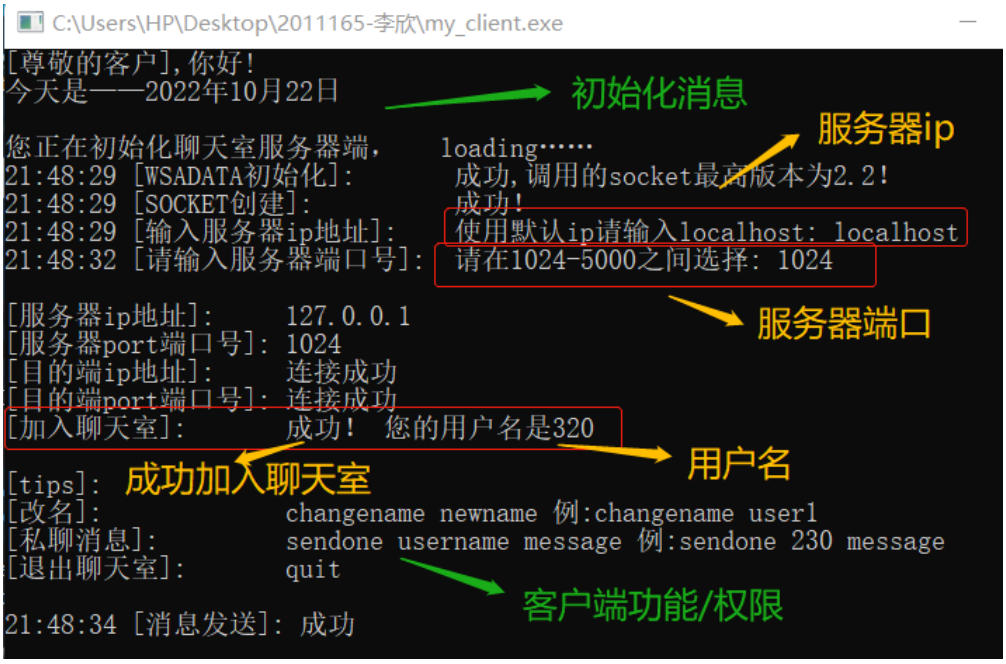
```
DWORD WINAPI se_ma(LPVOID lparam) {
    SOCKET* s = (SOCKET*)lparam;
    int sflag;
    while (true) {
        cin.getline(sendbuffer, 100);//命令行输入
        if (string(sendbuffer) == "quit") {
            sflag = send(*s, sendbuffer, 100, 0);
            flag = 0;
            H_M_S();
            cout << " [退出聊天]: 再见! " << endl;
            closesocket(*s);
            return 0;
        }
        sflag = send(*s, sendbuffer, 100, 0);
        if (sflag == SOCKET_ERROR) {
            H_M_S();
            cout << " [消息发送]: 失败; 错误代号" << WSAGetLastError() << endl;
            closesocket(*s);
            WSACleanup();
            return 0;
        }
        else {
            H_M_S();
            cout << " [消息发送]: 成功"<<endl;
        }
    }
}
```

五、程序界面展示及运行说明

服务端初始化:



客户端初始化:



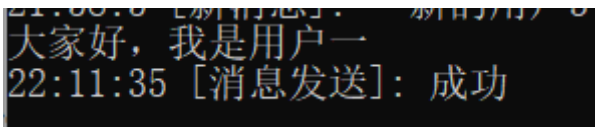
通过一次运行来具体说明，本次运行开启了三个客户端，接下来，我们通过发送消息一一观察群聊/私聊/改名和退出功能的实现与界面变化。

初始化：

首先是初始化3个客户端和1个服务器端，让他们实现连接。

发送群聊消息：

客户端1(320)：发送群聊消息——**大家好，我是用户一**



服务端：打印群聊日志

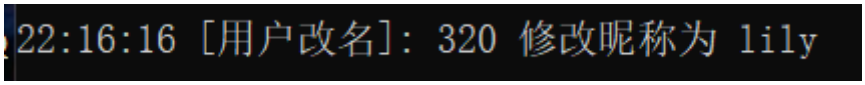
客户端2(332)：接收群聊消息

客户端3(348)：接收群聊消息

改名：

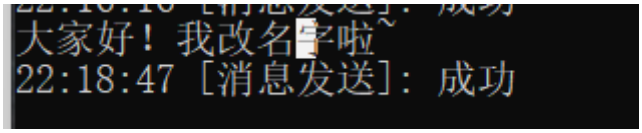
客户端1(320)：发送改名请求——**changename lily**

服务端：打印群聊日志

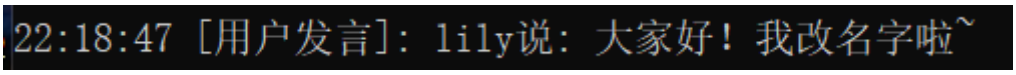


改名后发送群聊消息：

客户端1：发送消息——**大家好！我改名字啦~**



服务端：打印日志



客户端2和3：接收消息

可见，改名后，客户端间聊天以新名字进行。

发送私聊消息：

客户端2：发送消息——sendone lily 你好，我是客户端2

服务端：打印日志

客户端1(lily)：接收私聊消息

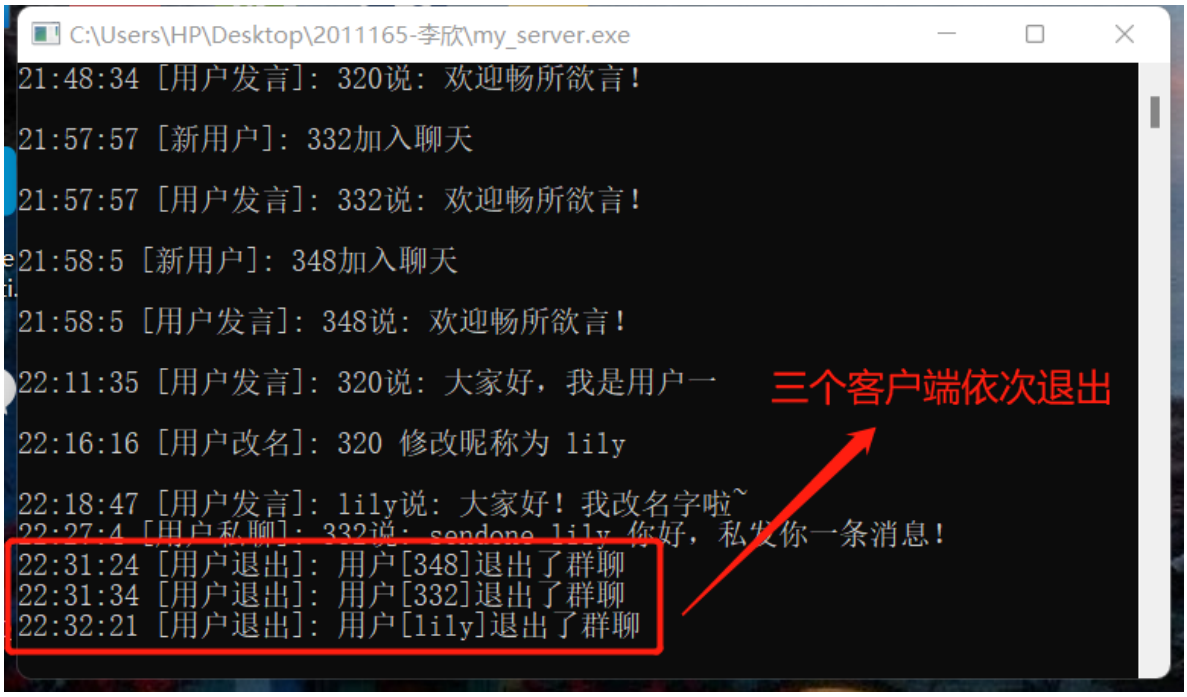
客户端3：未收到消息

为了证明程序的正确性，我们看服务器和3个客户端同一时刻下的状态

可见，私聊消息，只有指定的用户能够收到。

退出

在客户端中输入quit，客户端退出后在服务器端打印日志。



六、实验过程中遇到的问题及分析

1、要实现聊天室，需要服务端+多个客户端，需要服务器承担多种功能，接收并判定消息类型，发送消息，刚开始编程时产生了阻塞的错误，后面明白需要对每一个客户端开启一个线程进行处理。

2、出现了很多**打印**的bug，后面通过反复测试调整，并**添加相应的数据**结构**进行处理，很好的解决了这些问题。