01_Socket套字节聊天程序_实验报告

姓名: 孟笑朵
学号: 2010349

实验要求

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）对程序进行测试，提交源码；
5）撰写实验报告并提交。

实验概述

本次实验在Windows系统下，利用C/C++实现了一个**基于流式Socket套字节的多人聊天程序，实现了多个客户端和服务端之间的同时聊天功能**，程序界面为命令行界面。实验中所用到的主要技术如下：

- 重新设计了传输消息的格式，消息格式规定了不同客户端和服务端；
- 创建线程解决服务端和不同客户端之间的消息交互，创建线程解决服务端和客户端同时收发消息避免阻塞；
- 设计了以服务端为主体，通过服务端进行不同客户端之间的消息通信的消息交互模式；

全部代码及实验报告已上传[我的计算机网络作业github仓库](NK-MXD/ComperNet: 我的计算机网络和网络技术与应用仓库 (github.com))

> 本次实验借鉴了网上的一些博文, 链接如下:
>
> 1. (29条消息) C++实现流式socket聊天程序_Roslin_v的博客-CSDN博客
> 2. (29条消息) c++网络编程：实现简单的聊天——基于服务器/客户端的模式_圣颖君的博客-CSDN博客
> 3. (29条消息) C++网络编程（一）：TCP套接字编程_你喜欢梅西吗的博客-CSDN博客_c++套接字
> 4.(29条消息) C++ SOCKET多线程编程实现聊天小程序_NKU丨阳的博客-CSDN博客

协议设计

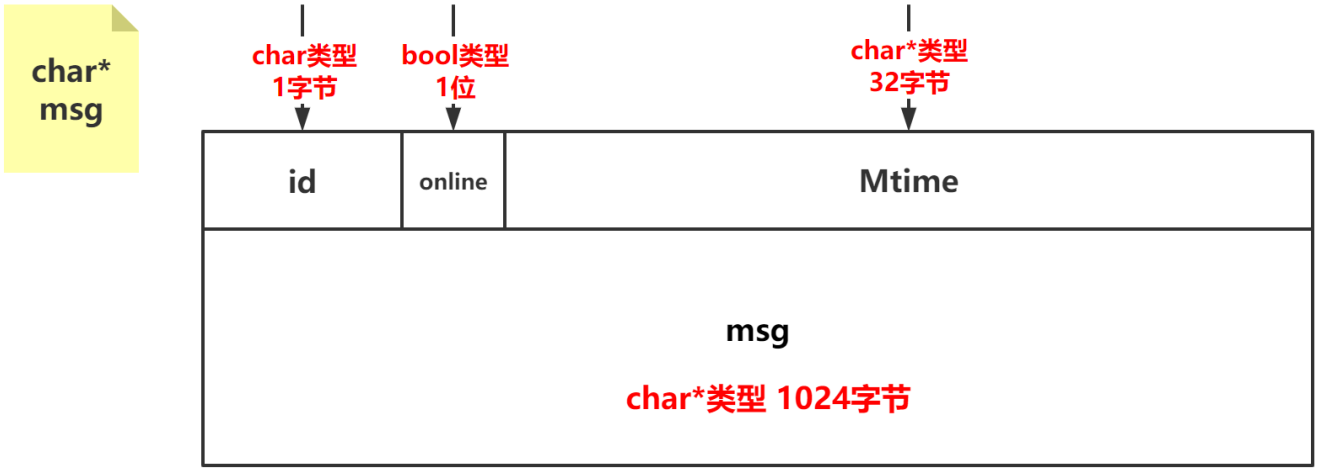
消息的类型，语法和语义

套字节的消息的接收函数`recv`和发送函数`send`参数如下：

```C++
send(
    sockfd, //指定发送端套接字描述符。
    buff,   //存放要发送数据的缓冲区
    nbytes, //实际要发送的数据的字节数
    flag,   //一般设置为0
)
recv(
    sockfd, //指定发送端套接字描述符。
    buff,   //用来存放recv函数接收到的数据的缓冲区
    nbytes, //指明buff的长度
    flag,   //一般设置为0
)
```

可以发现,套字节消息的传递是以字节为单位进行传递的,在本程序中,将客户端和服务端之间的消息交互用`字符数组char*`的格式进行传递,实际发送的字节数即为字符数组的长度。

如下图所示为消息交互的字符数组`msg`的语法格式:



语义说明如下:

```
char id: 表示发送消息的客户端id或者服务器id
bool online: 表示发送消息的客户端或服务器是否在线
char* Mtime[32]: 表示客户端或者服务端发送消息的时间
char* msg[1024]: 表示客户端或者服务器发送消息的具体内容
```

说明: 消息中的`id`默认为`'0'`,只有在服务端进行发送消息和服务端转发来自其它客户端的消息时使用。
例如: 当`客户端1`接受到消息中`id`为`'0'`时,说明为服务器发送服务器聊天内容,当客户端接受到的消息中`id`为`'4'`时说明为服务器转发`客户端4`的消息内容,此时在`客户端1`显示的消息应当为`客户端4`所发的消息。
进行上述设计的原因在于，正常情况下，我们只能进行客户端和服务端的通信，而无法让两个客户端之

间进行直接通信，需要**借助服务端转发不同客户端的消息来间接实现不同客户端之间的通信，这种情况下就需要区分服务端所发消息是来自服务端本身还是转发自客户端。**

转化为代码为如下自定义结构体格式：

```c
struct message {
    char id;
    bool online;//针对客户端表示是否在线，针对服务端表示聊天室是否正常运行
    char Mtime[32] = { NULL };//消息发送的时间
    char msg[1024] = { 0 };//表示客户端或者服务器发送消息的具体内容
};
```

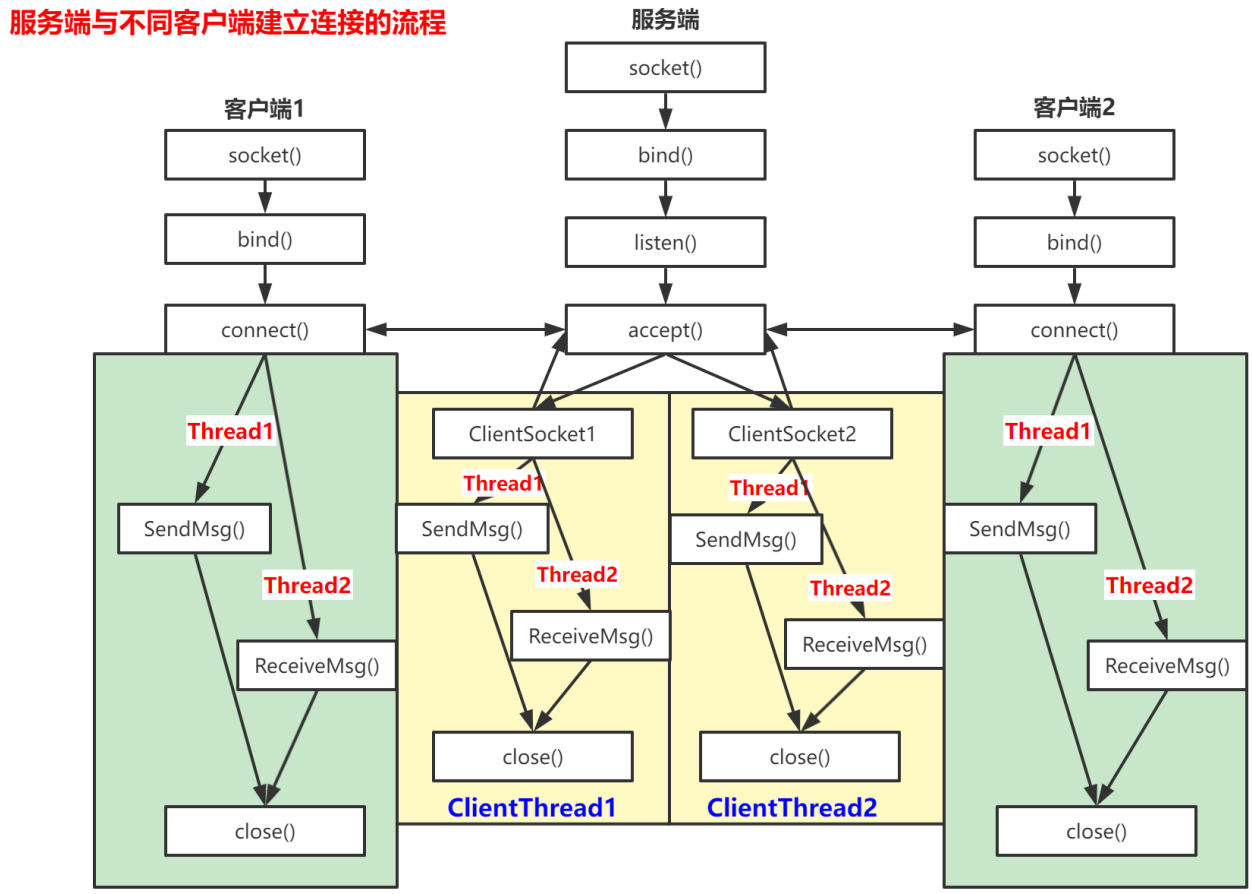
另外，我们需要定义将输入的要发送的字符串转化为对应`message`格式的函数`sendchar2msg(char*)`，再将转化得到的`message`格式的数据转化为`char*`字符串的函数`msg2char(message)`，需要接受消息时将对应的字符串`char*`转化为`message`的函数`receivechar2msg(char*)`，如下所示：

```c
message sendchar2msg(char* buf) {
    message m;
    char tmp[32] = { NULL };
    time_t t = time(0);
    strftime(tmp, sizeof(tmp), "%Y-%m-%d %H:%M:%S", localtime(&t));
    strcpy(m.Mtime, tmp);
    if (strcmp(buf, "exit") == 0) {
        m.online = false;
    }
    else {
        m.online = true;
    }
    strcpy(m.msg, buf);
    return m;
};

char* msg2char(message m){
    char charmsg[2048];
    memset(charmsg, 0, sizeof(charmsg));
    charmsg[0] = m.id;
    if (m.online == true) {
        charmsg[1] = '1';
    }
    else {
        charmsg[1] = '0';
    }
    strcat_s(charmsg, m.Mtime);
```

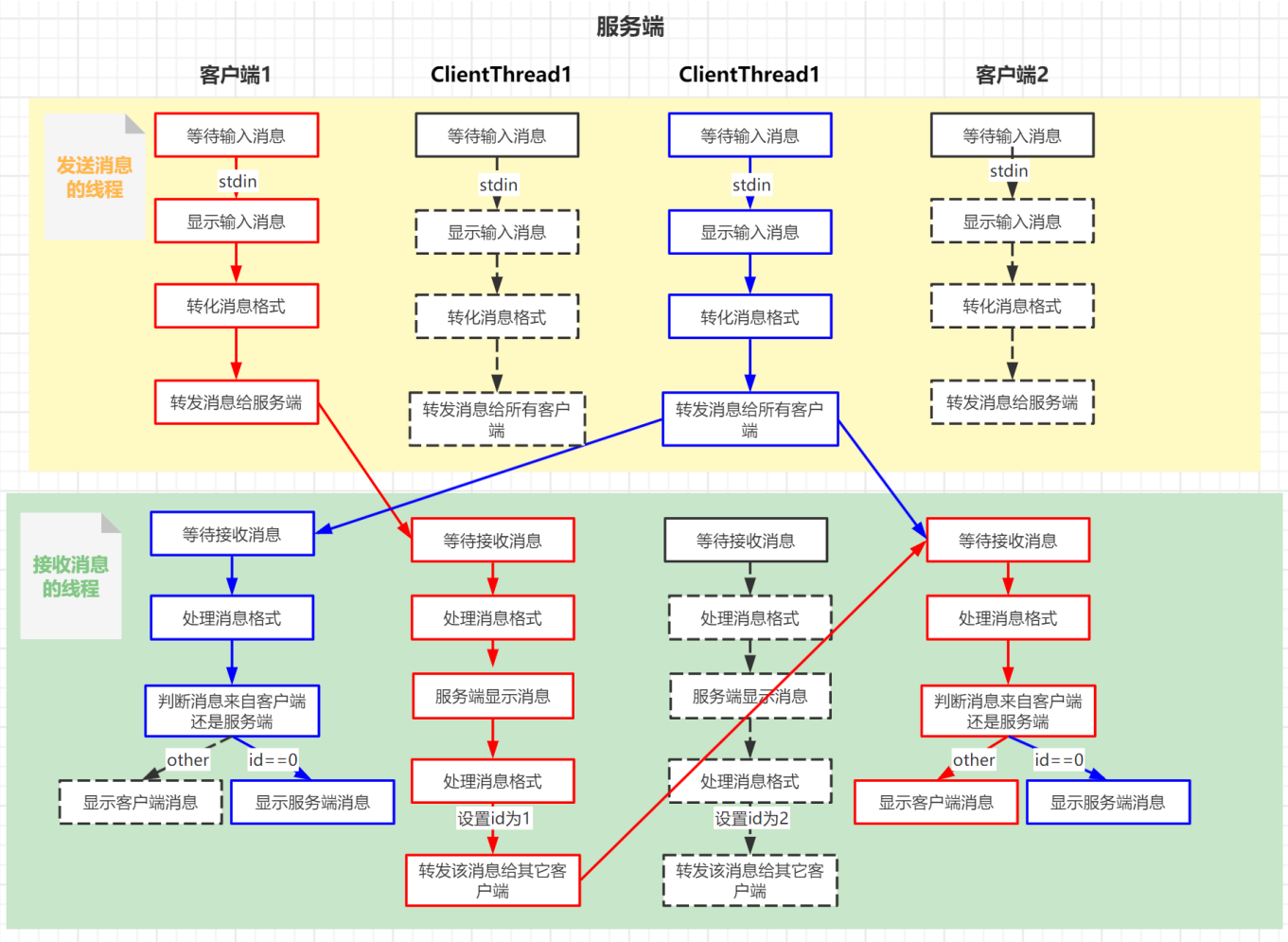
```
    strcat_s(charmsg, m.msg);
    return charmsg;
};

message receivechar2msg(char* buf) {
    message m;
    m.id = buf[0];
    if (buf[1] == '0') {
        m.online = false;
    }
    else {
        m.online = true;
    }
    strncpy(m.Mtime, buf + 2, 19);
    strcpy(m.msg, buf + 21);
    return m;
};
```

交互消息的时序

在进行消息交互时，我们需要先建立服务端和客户端之间的连接，并创建服务端和客户端之间收发消息的线程，如下图所示为服务端和不同客户端建立连接的过程示意图：



在此基础上，我们以两个客户端和服务端进行通信的过程为例，下面我们来建立服务端和客户端之间收发消息的时序关系图：



如上图所示，其中蓝色的线为服务端发送消息所有客户端接收消息的时序逻辑，红色的线为客户端发送消息到其它客户端和服务端的时序逻辑。

说明： 其中在转化消息格式的时候，添加了消息发送的时间。

程序设计

模块的划分和功能

Server服务器模块划分功能

1. `main`主函数部分: 负责`socket`的创建、初始化和绑定，进行监听，`持续连接客户端`，关闭连接，停止服务器。
2. `SendMsg`子线程部分：负责消息的发送。
3. `ReceiveMsg`子线程部分：负责消息的接收。

如下所示是实现`Server`服务端中 `main`主函数部分`持续连接客户端`的代码：

```C
int clientnum = 0;//标记聊天的客户端id
SOCKET client[MAX_CLIENT_NUM];

...
/*5. 服务器监听成功,连接请求*/
while (1) {
    sockaddr_in revClientAddress;//套接字的地址，端口
    SOCKET revClientSocket = INVALID_SOCKET;//客户端连接到套字节
    int addlen = sizeof(revClientAddress);
    if ((revClientSocket = accept(SeverSocket, (sockaddr*)&revClientAddress,
&addlen)) == INVALID_SOCKET)
    {
        cout << "啊欧接受客户端连接失败了！别灰心再试一次" << endl;
        return 0;
    }
    else
        cout << "【系统消息】找到一位一起聊天的小伙伴q(≧▽≦q)，跟他打个招呼吧!" <<
endl;
    client[clientnum++] = revClientSocket;
    cout << "【系统消息】客户端 " << clientnum << " 上线啦!" << endl;
    chat((LPVOID)revClientSocket);
}
```

说明: 在这里我们使用一个数组`client`来保存不同客户端的连接,使用数组下标来标识不同客户端。

如下所示是实现`Server`服务端中 `SendMsg`子线程部分代码:

```C
//发送数据的线程函数
void SendMsg(LPVOID lpParameter)
{
    //服务器发送的聊天内容:
    while (1)
    {
        //服务器发送数据
        SOCKET revClientSocket = (SOCKET)lpParameter;
        char sendbuf[2048] = {};        //输入缓冲区
        cin.getline(sendbuf, 2048);
        //发送的数据是char类型的
        //char* +time +online -> char*
        char *sendmsg;      //发送缓冲区
        message m = sendchar2msg(sendbuf);
        m.id = '0';
        sendmsg = msg2char(m);
```

```
        cout << m.Mtime << "  【服务器\\^o^/】:" << m.msg << endl;
        for (int i = 0; i < MAX_CLIENT_NUM; i++) {
            if (client[i] != NULL) {
                SOCKET revClient = client[i];
                if (send(revClient, sendmsg, strlen(sendmsg), 0) ==
SOCKET_ERROR)
                {
                    cout << "发送消息失败！" << endl;
                }
            }
        }
        if (m.online == false) {
            cout << "聊天室关闭，欢迎下次闲聊byebye~\(≧▽≦)/~" << endl;
            cout << "服务端退出，程序将在3秒后退出！" << endl;
            Sleep(3000);
            exit(100);
        }
    }
}
```

说明：

1. 当服务端发送消息时，会向所有的客户端发送消息。
2. 当服务端下线时（即在命令行中输入`exit`），服务端广播消息，并退出程序。

如下所示是实现`Server`服务端中`ReceiveMsg`子线程部分代码：

```C
//接受消息的线程函数
void ReceiveMsg(LPVOID lpParameter)
{
    int receiveid = clientnum;//标识当前客户端
    while (1)
    {
        //服务器接受数据
        SOCKET revClientSocket = (SOCKET)lpParameter;
        char recvbuf[2048] = {};        //接收缓冲区
        //接受的数据是char类型的,需要转化为msg类型的数据
        //char(+time +online) -> string -> message =>输出
        if (recv(revClientSocket, recvbuf, 2048, 0) == SOCKET_ERROR)
        {
            cout << WSAGetLastError() << endl;
            cout << "【系统消息】数据接收失败，客户端 "<< receiveid<<" 掉线了
{{{(>_<)}}}" << endl;
```

```
        }
        else {
            if (strlen(recvbuf) != 0) {
                //服务器显示其它客户端的消息:
                message m = receivechar2msg(recvbuf);
                cout << m.Mtime << "  【客户端 " << receiveid << " (*^_^*)】:"
<< m.msg << endl;
                //服务器向其它客户端转发内容
                m.id = receiveid + 48;
                char* sendmsg1 = msg2char(m);
                for (int i = 0; i < MAX_CLIENT_NUM; i++) {
                    if (client[i] != NULL && i != receiveid - 1) {
                        SOCKET revClient = client[i];
                        if (send(revClient, sendmsg1, strlen(sendmsg1), 0) ==
SOCKET_ERROR)
                        {
                            cout << "【系统消息】发送消息失败！" << endl;
                        }
                    }
                }
                if (m.online == false) {
                    cout << "【系统消息】客户端 " << receiveid << " 下线了，欢迎
下次闲聊byebye~\(≧▽≦)/~" << endl;
                    client[receiveid - 1] = NULL;
                    break;
                }
            }
        }
    }
}
```

说明： 服务器接受其它客户端的数据时，一方面需要显示该客户端发送的消息内容，另一方面需要将该客户端发送的消息转发至其它客户端。

Client客户端模块划分和功能

1. `main`主函数部分: 负责`socket`的创建、初始化和绑定，`连接服务端`，关闭连接，终止客户端。
2. `SendMsg`子线程部分：负责消息的发送。
3. `ReceiveMsg`子线程部分：负责消息的接收。

如下所示是实现`Client`客户端中 `SendMsg`子线程部分代码：

```C
void SendMsg(void* param)
{
    //客户端发送数据给服务器
    SOCKET clientSocket = *(SOCKET*)(param);
    char initbuf[2048] = "【新人入室】大家好呀~ 很高兴能和大家闲聊 =)";
    char* sendmsg;      //发送缓冲区
    message m = sendchar2msg(initbuf);
    sendmsg = msg2char(m);
    if (send(clientSocket, sendmsg, strlen(sendmsg), 0) == SOCKET_ERROR)
    {
        cout << "发送消息失败! ";
    }
    else {
        cout << m.Mtime << " 【客户端我\\ ^ o ^ / 】:" << m.msg << endl;//标识自己发出的消息
    }
    while (1)
    {
        char sendbuf[2048] = {};        //发送缓冲区
        cin.getline(sendbuf, 2048);
        //发送的数据是char类型的
        //char* +time +online -> char*
        char* sendmsg;      //发送缓冲区
        message m = sendchar2msg(sendbuf);
        sendmsg = msg2char(m);
        if (send(clientSocket, sendmsg, strlen(sendmsg), 0) == SOCKET_ERROR)
        {
            cout << "发送消息失败! ";
        }
        else{
            cout << m.Mtime << " 【客户端我\\ ^ o ^ / 】:" << m.msg << endl;//标识自己发出的消息
            if (m.online == false) {
                cout << "【系统消息】欢迎下次闲聊byebye~\(≧▽≦)/~" << endl;
                cout << "【系统消息】客户端退出，程序将在3秒后退出! " << endl;
                Sleep(3000);
                exit(100);
            }
        }
    }
}
```

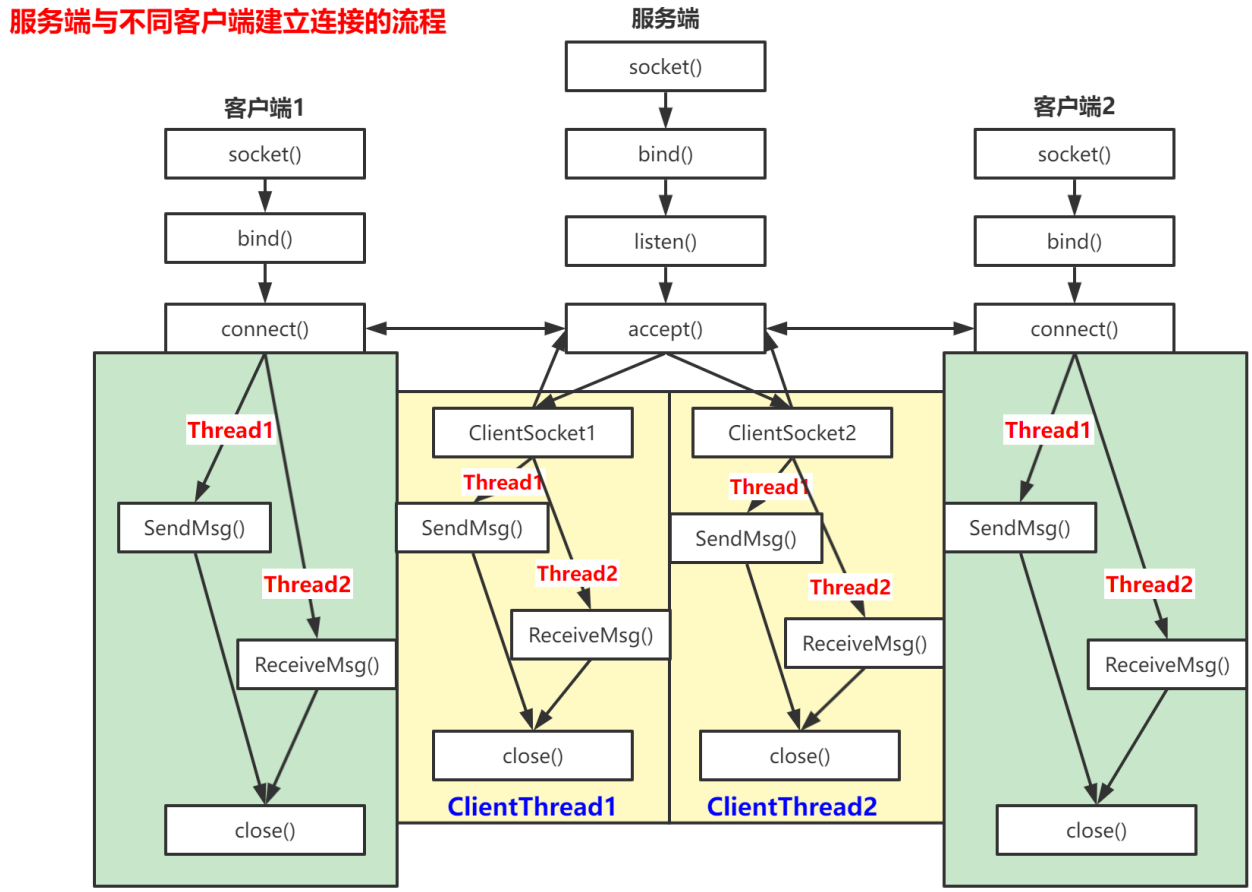
如下所示是实现`Client`客户端中`ReceiveMsg`子线程部分代码：

```c
void ReceiveMsg(void* param)
{
    while (1)
    {
        //客户端接受来自服务器的数据
        //  1. 服务器发送的聊天内容
        //  2. 服务器传递的其它客户端的内容
        SOCKET clientSocket = *(SOCKET*)(param);
        char  recvbuf[2048] = {};       //接收缓冲区
        //接受的数据是char类型的,需要转化为msg类型的数据
        //char(+time +online) -> string -> message =>输出
        if (recv(clientSocket, recvbuf, 2048, 0) == SOCKET_ERROR)
        {
            cout << "【系统消息】服务端掉线了{{{(>_<)}}}" << endl;
            cout << "【系统消息】程序将在3秒后退出" << endl;
            Sleep(3000);
            exit(100);
        }
        else {
            if (strlen(recvbuf)!=0) {
                message m = receivechar2msg(recvbuf);
                if (m.id == '0') {
                    //1. 服务器发送的聊天
                    cout << m.Mtime << "  【服务器(*^_^*)】:" << m.msg << endl;
                    if (m.online == false) {
                        cout << "【系统消息】聊天室关闭，欢迎下次闲聊byebye~\
(≧▽≦)/~" << endl;
                        cout << "【系统消息】客户端退出，程序将在3秒后退出！" <<
endl;
                        Sleep(3000);
                        exit(100);
                    }
                }
                if (m.id != '0') {
                    //2. 服务器转发的其它客户端的内容
                    cout << m.Mtime << "  【客户端 "<<m.id<<" (*^_^*)】:" <<
m.msg << endl;
                    if (m.online == false) {
                        cout << "【系统消息】客户端 " << m.id << " 下线了，欢迎下
次闲聊byebye~\(≧▽≦)/~" << endl;
                    }
                }
            }
```

```
            }
        }
    }
```

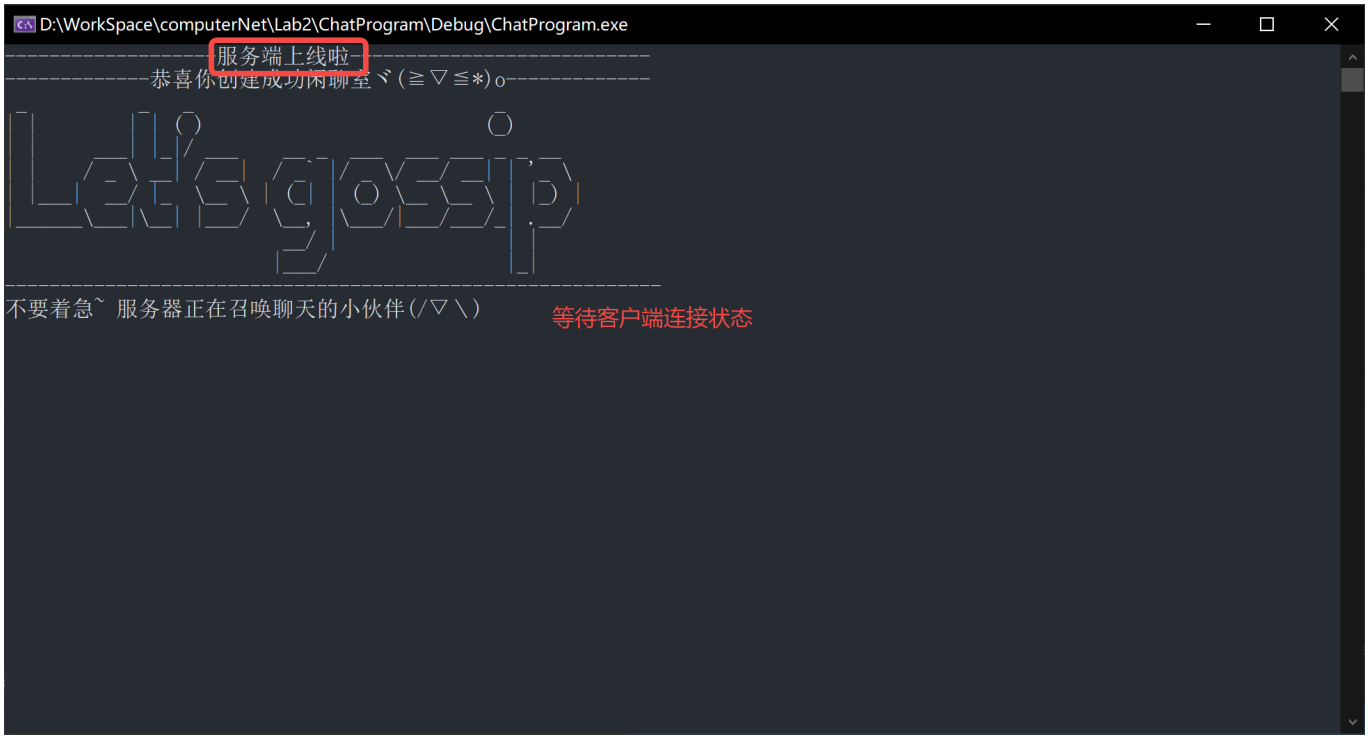
说明： 客户端接受来自服务器的数据需要判断是服务端本身发送的数据还是服务端转发其它客户端的数据。根据不同的发送消息的`id`来进行区分。

程序功能流程图

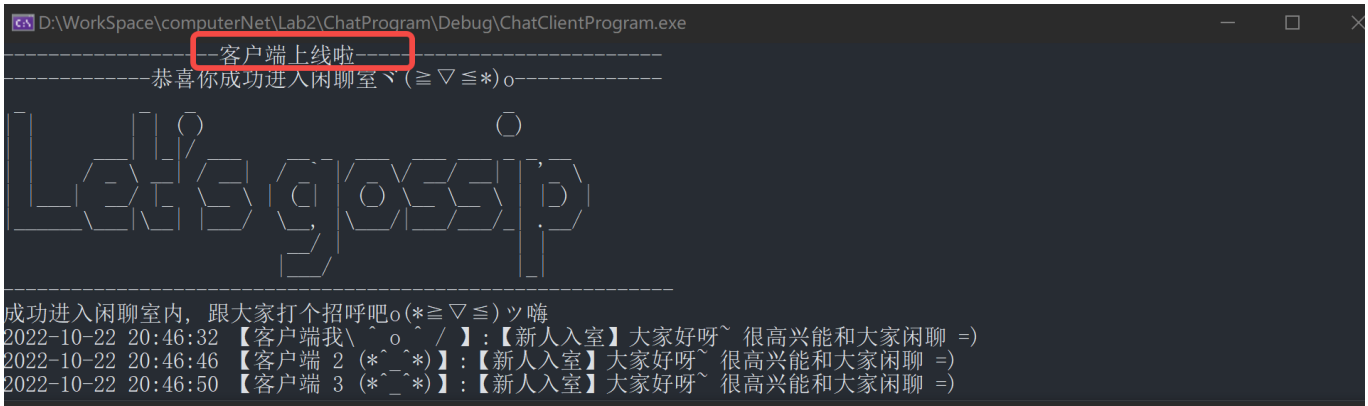


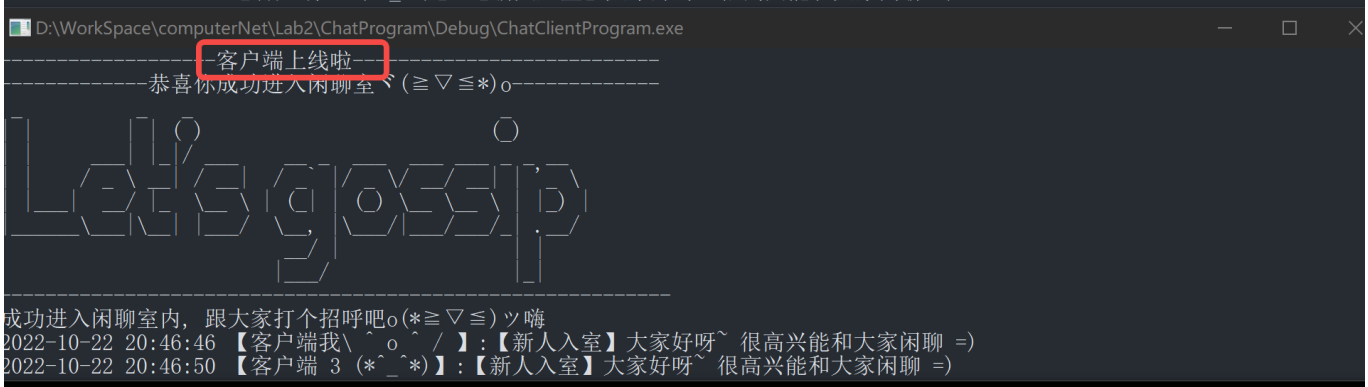
程序测试

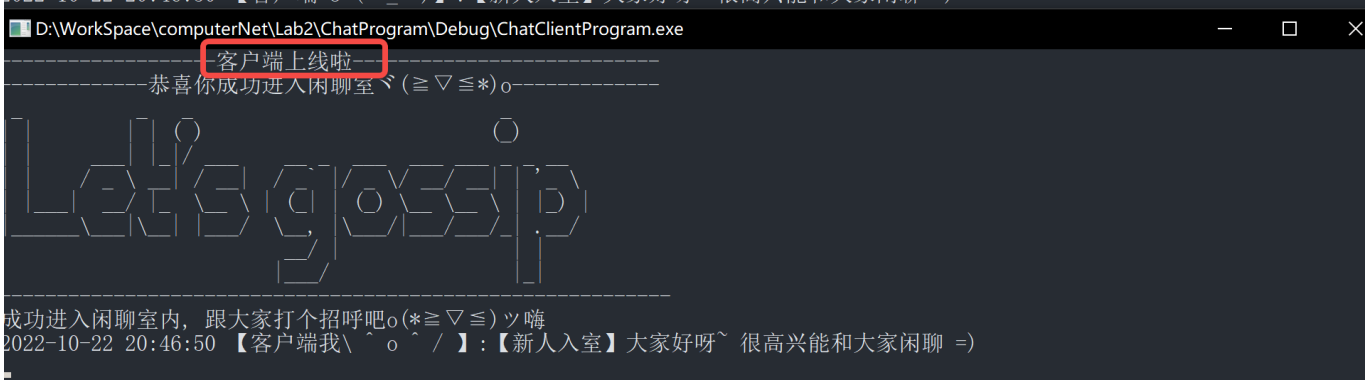
1. 首先运行服务端程序，如下图所示：



2. 运行一个或者多个客户端程序，如下图所示：

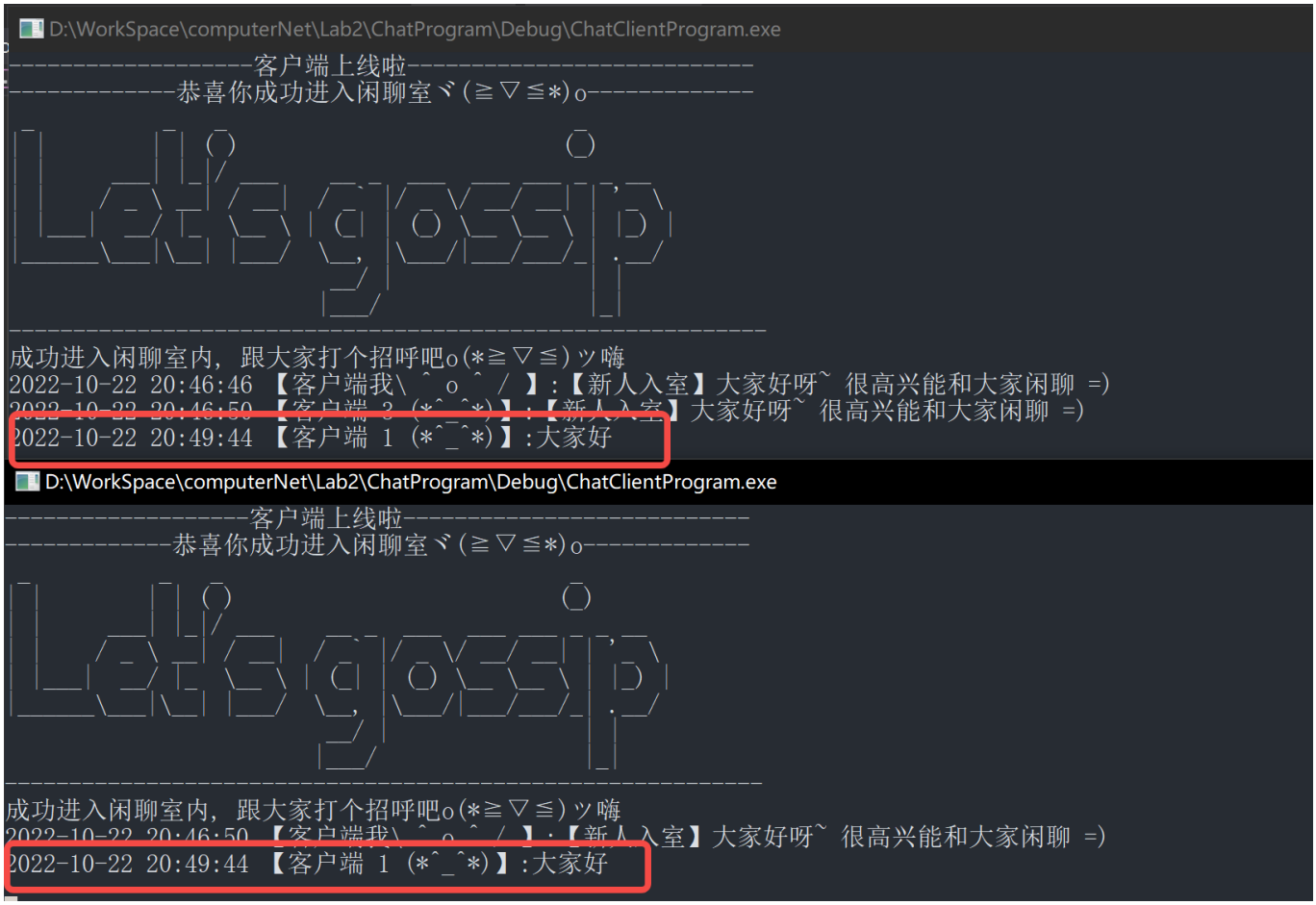

D:\WorkSpace\computerNet\Lab2\ChatProgram\Debug\ChatClientProgram.exe
----------客户端上线啦----------
----------恭喜你成功进入闲聊室ヾ(≧▽≦*)o----------
成功进入闲聊室内，跟大家打个招呼吧o(*≧▽≦)ツ嗨
2022-10-22 20:46:32 【客户端我\ ^ o ^ / 】:【新人入室】大家好呀~ 很高兴能和大家闲聊 =)
2022-10-22 20:46:46 【客户端 2 (*^_^*)】:【新人入室】大家好呀~ 很高兴能和大家闲聊 =)
2022-10-22 20:46:50 【客户端 3 (*^_^*)】:【新人入室】大家好呀~ 很高兴能和大家闲聊 =)


D:\WorkSpace\computerNet\Lab2\ChatProgram\Debug\ChatClientProgram.exe
----------客户端上线啦----------
----------恭喜你成功进入闲聊室ヾ(≧▽≦*)o----------
成功进入闲聊室内，跟大家打个招呼吧o(*≧▽≦)ツ嗨
2022-10-22 20:46:46 【客户端我\ ^ o ^ / 】:【新人入室】大家好呀~ 很高兴能和大家闲聊 =)
2022-10-22 20:46:50 【客户端 3 (*^_^*)】:【新人入室】大家好呀~ 很高兴能和大家闲聊 =)

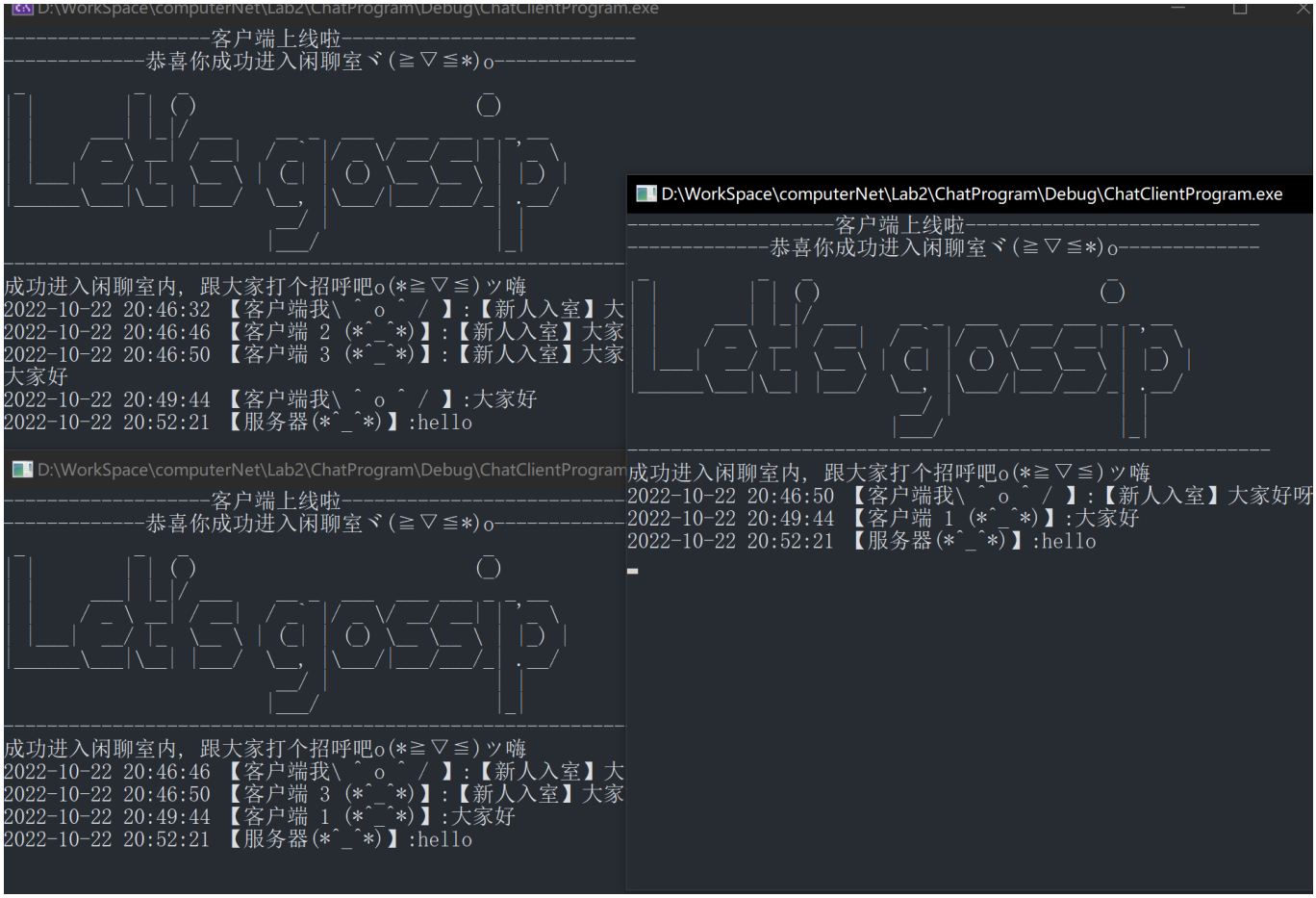

D:\WorkSpace\computerNet\Lab2\ChatProgram\Debug\ChatClientProgram.exe
----------客户端上线啦----------
----------恭喜你成功进入闲聊室ヾ(≧▽≦*)o----------
成功进入闲聊室内，跟大家打个招呼吧o(*≧▽≦)ツ嗨
2022-10-22 20:46:50 【客户端我\ ^ o ^ / 】:【新人入室】大家好呀~ 很高兴能和大家闲聊 =)


D:\WorkSpace\computerNet\Lab2\ChatProgram\Debug\ChatProgram.exe
----------服务端上线啦----------
----------恭喜你创建成功闲聊室ヾ(≧▽≦*)o----------
不要着急~ 服务器正在召唤聊天的小伙伴(/▽＼)
【系统消息】找到一位一起聊天的小伙伴q(≧▽≦q)，跟他打个招呼吧!
【系统消息】客户端 1 上线啦!
2022-10-22 20:46:32 【客户端 1 (*^_^*)】:【新人入室】大家好呀~ 很高兴能和大家闲聊 =)
【系统消息】找到一位一起聊天的小伙伴q(≧▽≦q)，跟他打个招呼吧!
【系统消息】客户端 2 上线啦!
2022-10-22 20:46:46 【客户端 2 (*^_^*)】:【新人入室】大家好呀~ 很高兴能和大家闲聊 =)
【系统消息】找到一位一起聊天的小伙伴q(≧▽≦q)，跟他打个招呼吧!
【系统消息】客户端 3 上线啦!
2022-10-22 20:46:50 【客户端 3 (*^_^*)】:【新人入室】大家好呀~ 很高兴能和大家闲聊 =)

3. 在其中一个客户端发送消息大家好，观察服务端和剩下两个客户端的状态：



4. 在服务端发送消息`hello`，观察客户端状态：



5. 在其中一个客户端发送`exit`消息：


D:\WorkSpace\computerNet\Lab2\ChatProgram\Debug\ChatProgram.exe
----------服务端上线啦----------
----------恭喜你创建成功闲聊室ヾ(≧▽≦*)o----------
不要着急~ 服务器正在召唤聊天的小伙伴(/▽＼)
【系统消息】找到一位一起聊天的小伙伴q(≧▽≦q)，跟他打个招呼吧！
【系统消息】客户端 1 上线啦！
2022-10-22 20:46:32 【客户端 1 (*^_^*)】:【新人入室】大家好呀~ 很高兴能和大家闲聊 =)
【系统消息】找到一位一起聊天的小伙伴q(≧▽≦q)，跟他打个招呼吧！
【系统消息】客户端 2 上线啦！
2022-10-22 20:46:46 【客户端 2 (*^_^*)】:【新人入室】大家好呀~ 很高兴能和大家闲聊 =)
【系统消息】找到一位一起聊天的小伙伴q(≧▽≦q)，跟他打个招呼吧！
【系统消息】客户端 3 上线啦！
2022-10-22 20:46:50 【客户端 3 (*^_^*)】:【新人入室】大家好呀~ 很高兴能和大家闲聊 =)
2022-10-22 20:49:44 【客户端 1 (*^_^*)】:大家好
hello
2022-10-22 20:52:21 【服务器\^o^/】:hello
2022-10-22 20:54:16 【客户端 3 (*^_^*)】:exit
【系统消息】客户端 3 下线了，欢迎下次闲聊byebye~(≧▽≦)/~

6. 在服务端输入`exit`，观察客户端的状态：


D:\WorkSpace\computerNet\Lab2\ChatProgram\Debug\ChatProgram.exe
----------恭喜你创建成功闲聊室ヾ(≧▽≦*)o----------
不要着急~ 服务器正在召唤聊天的小伙伴(/▽＼)
【系统消息】找到一位一起聊天的小伙伴q(≧▽≦q)，跟他打个招呼吧！
【系统消息】客户端 1 上线啦！
2022-10-22 20:46:32 【客户端 1 (*^_^*)】:【新人入室】大家好呀~ 很高兴能和大家闲聊 =)
【系统消息】找到一位一起聊天的小伙伴q(≧▽≦q)，跟他打个招呼吧！
【系统消息】客户端 2 上线啦！
2022-10-22 20:46:46 【客户端 2 (*^_^*)】:【新人入室】大家好呀~ 很高兴能和大家闲聊 =)
【系统消息】找到一位一起聊天的小伙伴q(≧▽≦q)，跟他打个招呼吧！
【系统消息】客户端 3 上线啦！
2022-10-22 20:46:50 【客户端 3 (*^_^*)】:【新人入室】大家好呀~ 很高兴能和大家闲聊 =)
2022-10-22 20:49:44 【客户端 1 (*^_^*)】:大家好
hello
2022-10-22 20:52:21 【服务器\^o^/】:hello
2022-10-22 20:54:16 【客户端 3 (*^_^*)】:exit
【系统消息】客户端 3 下线了，欢迎下次闲聊byebye~(≧▽≦)/~
exit
2022-10-22 20:55:57 【服务器\^o^/】:exit
聊天室关闭，欢迎下次闲聊byebye~(≧▽≦)/~
服务端退出，程序将在3秒后退出！


Microsoft Visual Studio Debug Console
----------客户端上线啦----------
----------恭喜你成功进入闲聊室ヾ(≧▽≦*)o----------
成功进入闲聊室内，跟大家打个招呼吧o(*≧▽≦)ツ嗨
2022-10-22 20:46:32 【客户端我\ ^ o ^ / 】:【新人入室】大家好呀~ 很高兴能和大家闲聊 =)
2022-10-22 20:46:46 【客户端 2 (*^_^*)】:【新人入室】大家好呀~ 很高兴能和大家闲聊 =)
2022-10-22 20:46:50 【客户端 3 (*^_^*)】:【新人入室】大家好呀~ 很高兴能和大家闲聊 =)
大家好
2022-10-22 20:49:44 【客户端我\ ^ o ^ / 】:大家好
2022-10-22 20:52:21 【服务器(*^_^*)】:hello
2022-10-22 20:54:16 【客户端 3 (*^_^*)】:exit
【系统消息】客户端 3 下线了，欢迎下次闲聊byebye~(≧▽≦)/~
2022-10-22 20:55:57 【服务器(*^_^*)】:exit
【系统消息】聊天室关闭，欢迎下次闲聊byebye~(≧▽≦)/~
【系统消息】客户端退出，程序将在3秒后退出！
D:\WorkSpace\computerNet\Lab2\ChatProgram\Debug\ChatClientProgram.exe (process 37436) exited with code 100.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .

可以发现程序测试成功。