计算机网络实验一

利用Socket，设计和编写一个聊天程序(聊天室)

姓名：刘伟 学号：2013029 专业：物联网工程

一、实验说明

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。

二、协议设计

实验使用的是Socket套接字，传输层协议采用TCP协议，数据以流式进行传输。

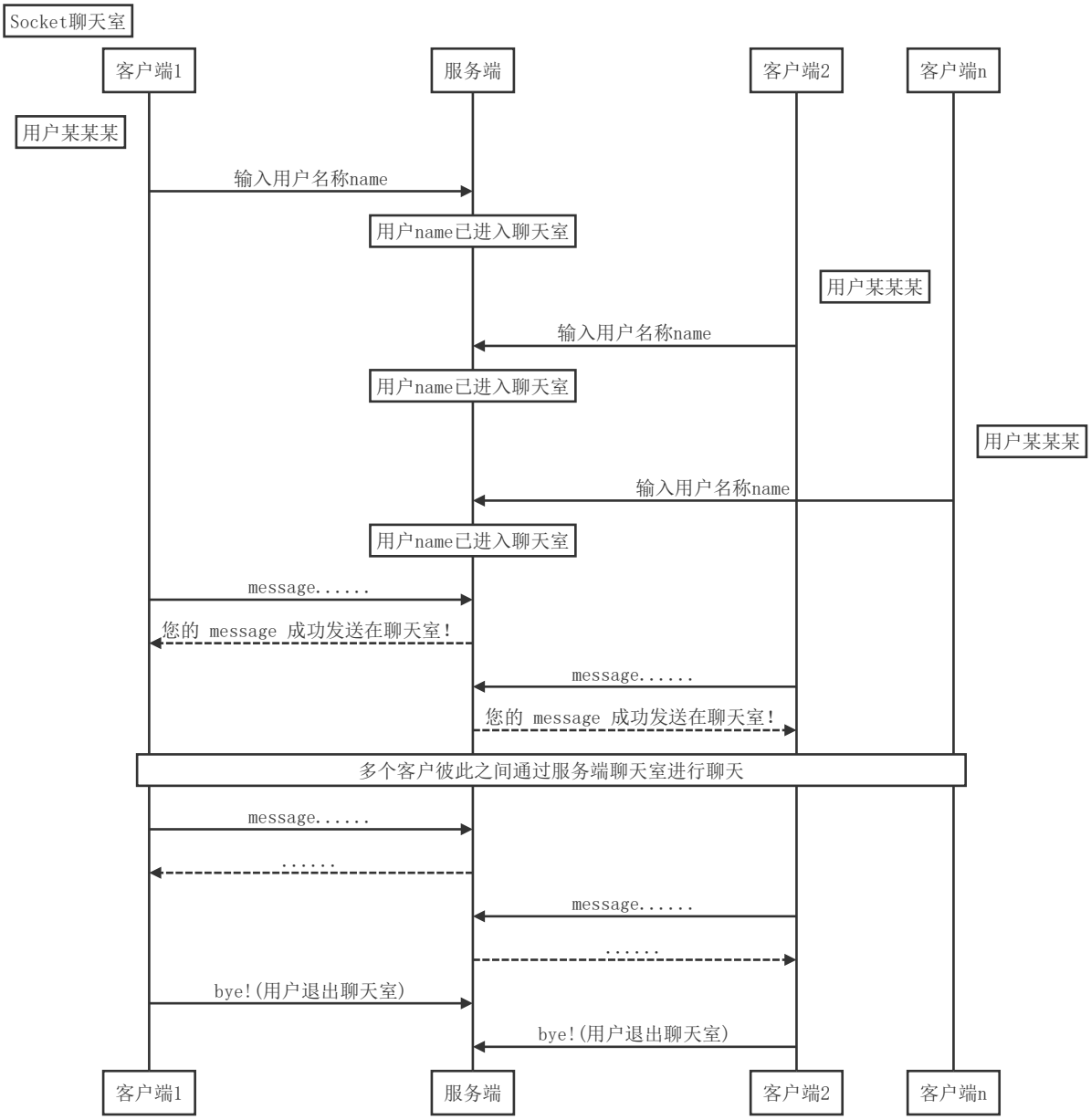
设计实现的聊天程序实现多人同步在线聊天的功能。本次实验的client与server端的PORT端口号已绑定，IP地址设定为127.0.0.1（也可进行修改成手动输入的模式）。

实验设计的聊天协议为：

- 客户端初始启动进入聊天室（连接上服务端），首先需要输入`用户名`，用户名的长度限定为`20`个字符之内
- 服务端成功接收到`用户名`后，会在聊天窗口输出`欢迎消息`
- 完成用户名输入，成功进入聊天室后，多个用户便可在聊天室进行多人聊天
- 用户输入的报文message包括俩个数据段，从高位到低位，第一个数据段表示发送/接受时间，第二个数据段表示发送的信息内容。这数据段之间用`\n`换行符以及标识符**{Time：，用户：}**进行连接。报文结构如下所示:

	标识符	内容
第一段：时间	Time：	实时时间
第二段：内容	用户：	用户输入信息内容

- 服务端收到用户端发来的信息message后，首先会将内容打印到多人聊天窗口上。同时，服务端（即聊天室）会返回给发消息的客户端响应消息：“您的 **message** 成功发送在聊天室！”。
- 用户们在聊天退出时需要输入的信息内容为：`bye!`，服务器聊天室在收到这个message后，会认为用户想要退出聊天室，彼此之间便会切断连接。
- 实验编程采用的是多线程的方法，故不同的用户端之间的消息发送是不相干的，可以随时的发送给服务端。



三、各模块功能

关于多线程与聊天室设计的说明：

- 聊天室

  实验最初设计的为客户-服务端二者之间的双人聊天，但是在设计的时候觉得该程序的实际意义并不是很大。故对其进行优化设计，尝试设计多人聊天室的功能程序。
  多人聊天室：多个用户借助同一服务器将各自像发送的信息在聊天室窗口打印出来，实现多人自由通信。

- 多线程

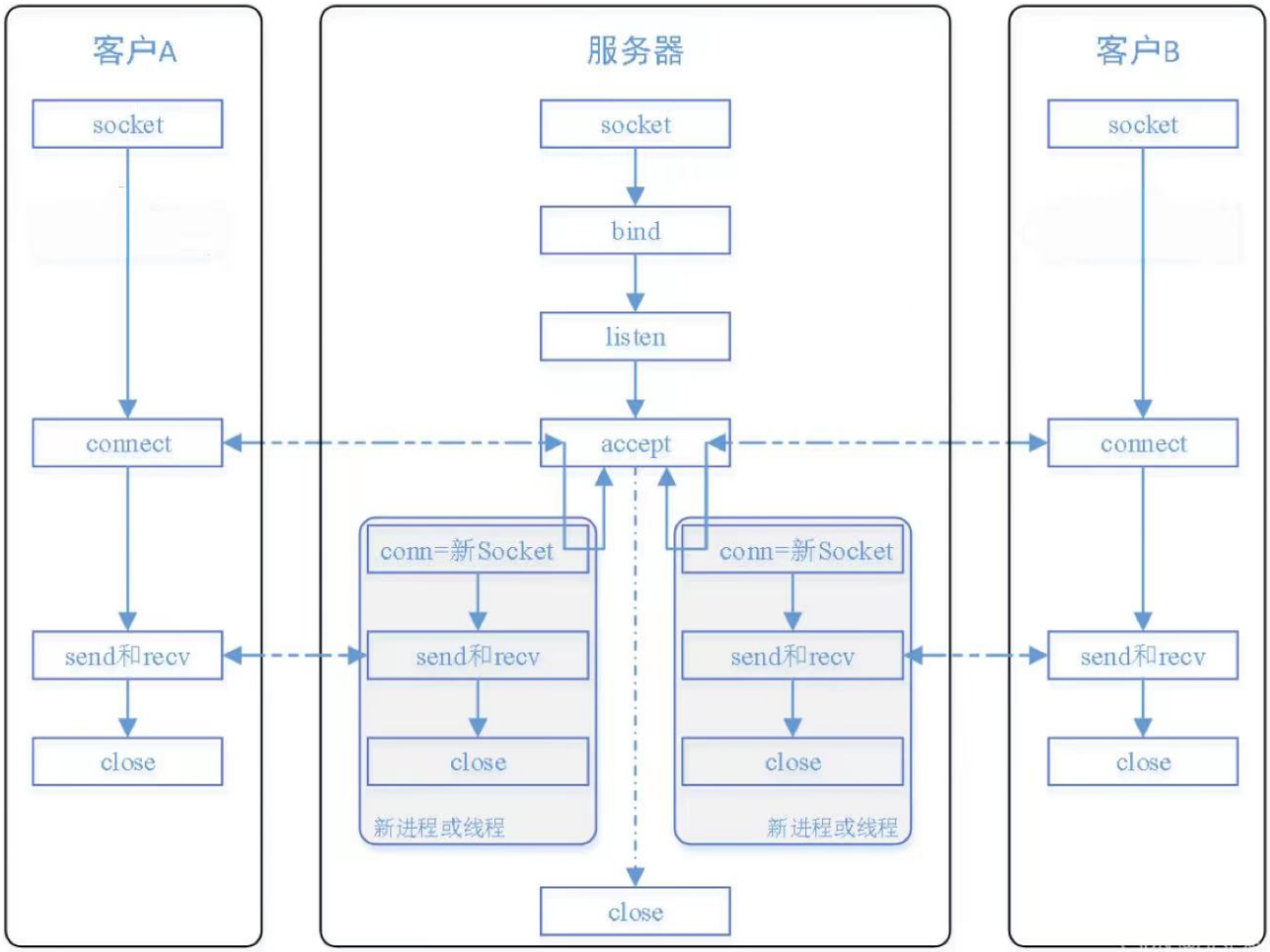
  在实验设计的时候，由于设计的是一款多人同时在线聊天的功能。这就要求多个用户彼此之间的进程不能相互干扰。在实验设计之处有考虑过用数组之类的数据结构进行保存客户端socket。尝试之后，该程序对于内存的要求可能会由于人数过多而不够实际、消耗可能较大。最终决定采用多线程的方式，彼此之间互不干扰。

```
HANDLE hThread = CreateThread(NULL, 0, ClientThread, &thread_client, 0, NULL);

DWORD WINAPI ClientThread(LPVOID ThreadParameter)
{
    threadsocket* clientT = (threadsocket*)ThreadParameter;
    SOCKET client = (SOCKET)(clientT->client);
    char* name = clientT->name;
    char buffer[1024];

    while (1)
    {
        ......
        iret = recv(client, buffer, sizeof(buffer), 0);
        ......
        strcpy(buffer, "您的 message 成功发送在聊天室！");
        ......
        iret = send(client, buffer, sizeof(buffer), 0); // 向客户端发送响应结果
        ......
    }
    return 0;
}
```

模块示意图：



模块功能分析：

1. 服务端与客户端的socket建立：注意需要导入相关头文件

```
WSADATA wsaData;  // 存储WSAStartup函数调用返回的Windows Sockets数据
int err = WSAStartup(MAKEWORD(2, 2), &wsaData);
if (err != 0)
{
cout << "Startup 出错！" << endl;
WSACleanup();
return 0;
}
SOCKET server = socket(AF_INET, SOCK_STREAM, 0);  // 创建 流式套接字
if (server == -1)
{
cout << "socket 生成出错！" << endl;
WSACleanup();
return 0;
}

// 绑定端口及IP
......
```

2. 客户端connect()函数：对指定的服务端发起连接

```
int connect(int sockfd, struct sockaddr * serv_addr, int addrlen);
// connect函数用于将参数sockfd 的socket 连至参数serv_addr 指定的服务端，参数addrlen为
sockaddr的结构长度。
// 返回值：成功则返回0，失败返回-1
if (connect(client, (struct sockaddr*)&serveraddr, sizeof(serveraddr)) != 0)
{
    cout << "connect 连接出错！" << endl;
    WSACleanup();
    return 0;
}
```

3. 客户端send()&recv()函数和close()关闭

```
while (1)
{
    memset(buf, 0, sizeof(buf));
    cout << "请输入聊天message：";

    if (strcmp(message, "bye!") == 0) {
        ......   // 准备退出结束
    }

    // 对数据进行协议指定的格式进行包装
    // 发送数据
    if (send(client, buf, strlen(buf), 0) <= 0) {
        cout << "客户端数据发送失败！！！" << endl;
        break;
    }
```

```
    //接收客户端回传消息
    if (recv(client, buf, sizeof(buf), 0) <= 0)
    {
        cout << "服务端数据接收失败！！！" << endl;
        break;
    }
}

closesocket(client);  // 关闭client的socket进程
WSACleanup();  //清理相关缓存 终止Winsock.dll的使用
```

4. 服务端bind()&listen()函数

```
int bind(int sockfd, const struct sockaddr *addr,socklen_t addrlen);
// 服务端把用于通信的地址和端口绑定到socket上。

if (bind(server, (struct sockaddr*)&addr, sizeof(addr)) != 0)
{
    cout << "bind 出错！" << endl;
    WSACleanup();
    return 0;
}

int listen(int sockfd, int backlog);
// listen函数把主动连接socket变为被动连接的socket，使得这个socket可以接受其它socket的连接请求，从而成为一个服务端的socket。

if (listen(server, 5) != 0)
{
    cout << "listen 监听出错！" << endl;
    WSACleanup();
    return 0;
}
```

5. 服务端accept()&线程创建

```
int accept(int sockfd,struct sockaddr *addr,socklen_t *addrlen);
/*
参数sockfd是已经被listen过的socket。
参数addr用于存放客户端的地址信息，用sockaddr结构体表达，如果不需要客户端的地址，可以填0。
参数addrlen用于存放addr参数的长度，如果addr为0，addrlen也填0。
accept函数等待客户端的连接，如果没有客户端连上来，它就一直等待，这种方式称之为阻塞。
accept等待到客户端的连接后，创建一个新的socket，函数返回值就是这个新的socket，服务端使用这个新的socket和客户端进行报文的收发。
*/
```

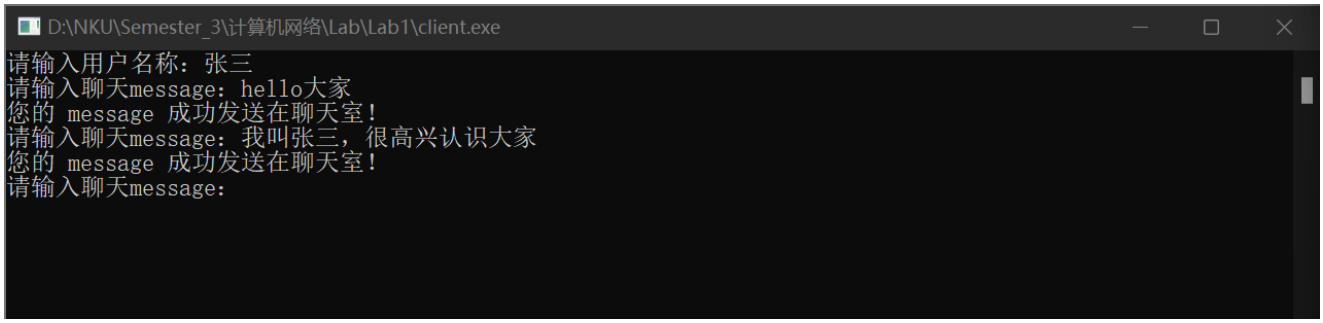
```
// 主线程循环接收客户端的连接
while (1)
{
    sockaddr_in addrClient;
    int socketlen = sizeof(sockaddr_in);
    // 接受成功返回与client通讯的Socket
```

```
    SOCKET client = accept(server, (SOCKADDR*)&addrClient, (socklen_t*)&socketlen);
    if (client != INVALID_SOCKET)
    {
        char name[10];
        memset(name, 0, sizeof(name));
        recv(client, name, sizeof(name), 0);
        cout << name << "已经进入聊天室！" << endl;
        // 创建线程，并且传入与client通讯的套接字
        threadsocket thread_client;
        thread_client.client = client;
        thread_client.name = name;
        // 创建线程
        HANDLE hThread = CreateThread(NULL, 0, ClientThread, &thread_client, 0, NULL);
        CloseHandle(hThread); // 关闭对线程的引用
    }
}
```

四、程序界面展示及运行说明

运行效果展示（示例：有3个用户同时在聊天室聊天）：

- 客户端运行展示：用户：张三、李四、王五进入聊天室进行聊天交流



- 客户端输入`bye!`代表用户退出聊天室

```
请输入聊天message: 很晚了，大家早点睡吧
您的 message 成功发送在聊天室!
请输入聊天message: bye!
您的 message 成功发送在聊天室!

D:\Project_C++\Net_Application\Debug\Net_Application.exe (进程 30492)已退出，代码为 0。
按任意键关闭此窗口. . .
```

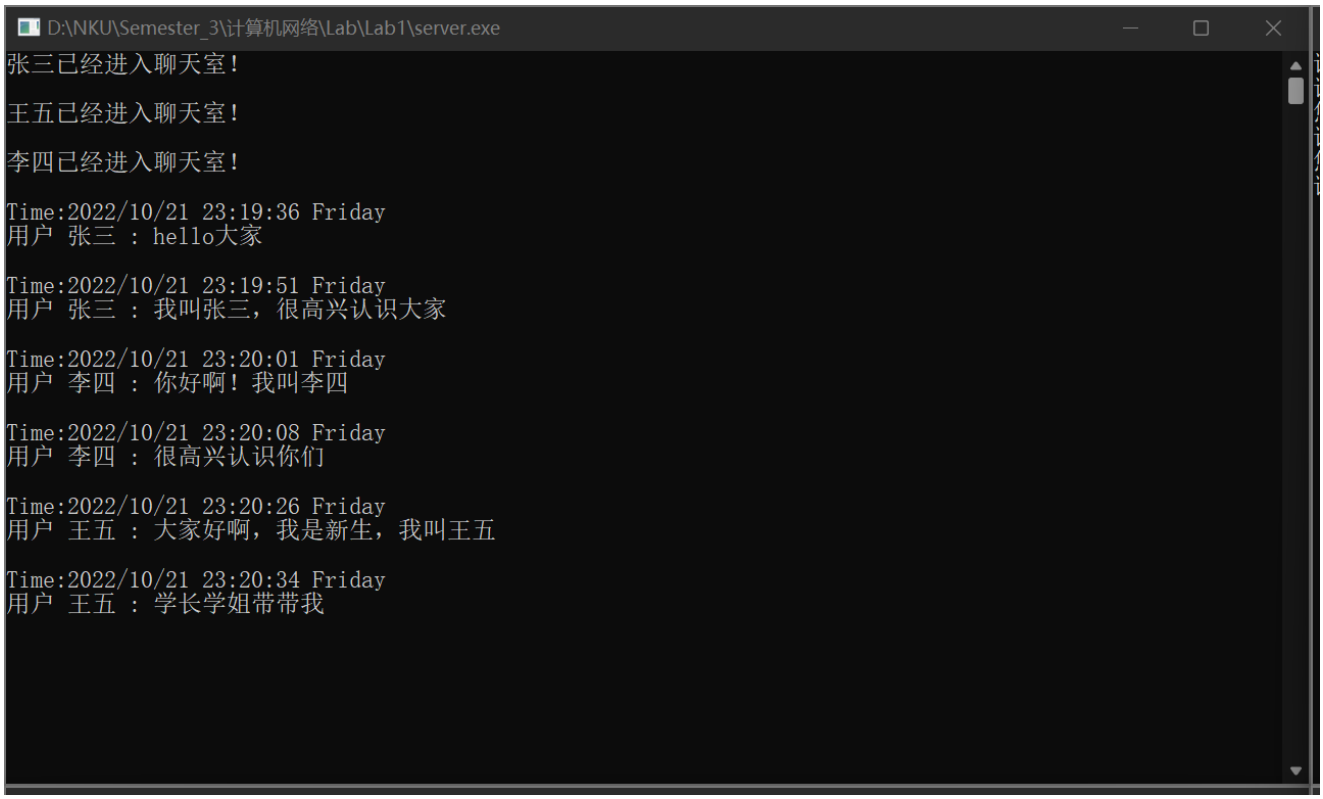
```
Time:2022/10/21 23:34:14 Friday
用户 李华 : 巴拉巴拉聊了一大多

Time:2022/10/21 23:34:27 Friday
用户 李华 : 很晚了，大家早点睡吧

Time:2022/10/21 23:34:31 Friday
用户 李华 : bye!

李华退出聊天室!
```

- 服务端运行展示：三个用户进入聊天室，并在此展示消息



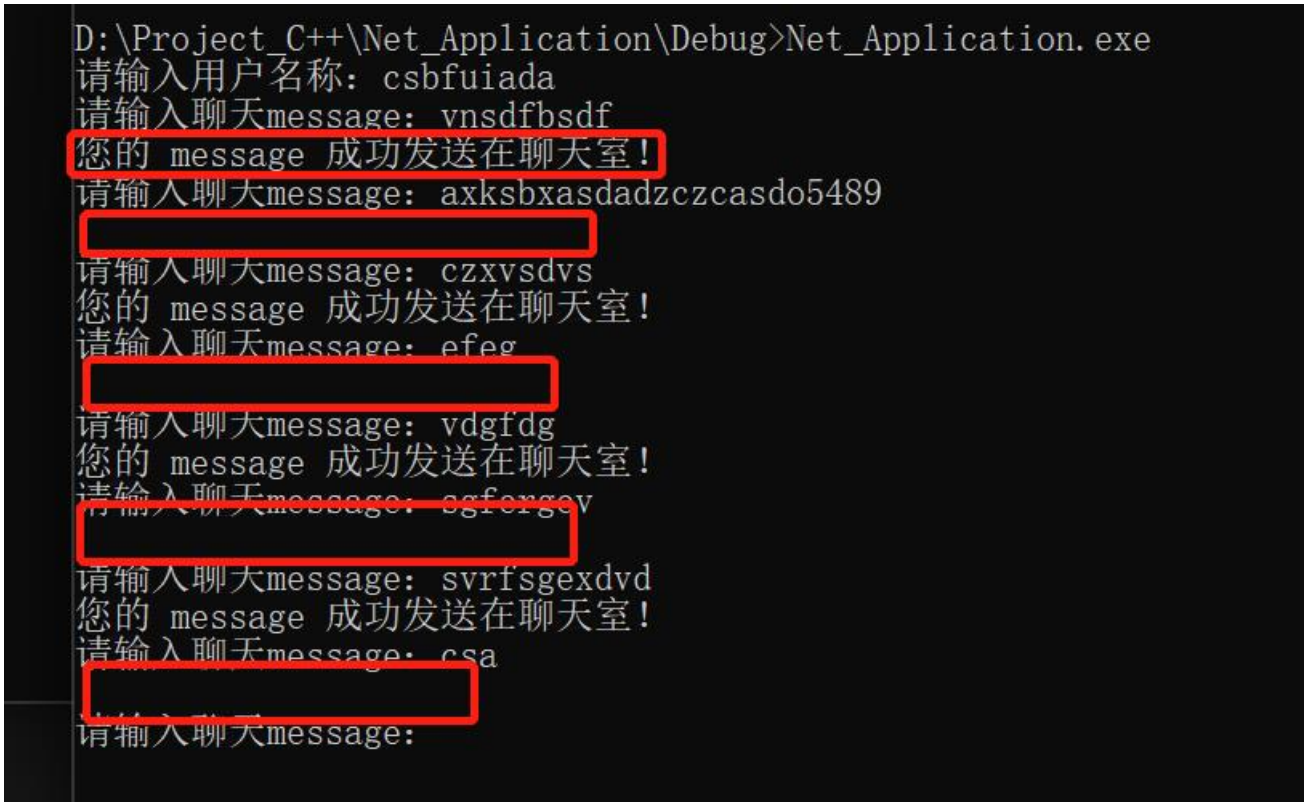
运行说明：

- 在本次实验中设计的客户-服务端由于均已经将IP与PORT定义好了，故俩个程序在启动后均没有指定输入IP与PORT。
- 首先启动服务端，否则客户端无法连接到服务端。
- 客户端执行启动，成功进入聊天室后，首先要输入用户名。告知聊天室是谁进入了聊天室。
- 聊天室（服务端）每次接收到一个客户端的连接之后，都会在聊天室的公共区域告知：“某某某进入了聊天室！”。
- 用户在各自的客户端输入想要发送的信息。这些信息在进行协议格式包装后，会被发送send到服务器，服务器进行输出交流。

- 当用户想要退出聊天室，断开连接的话，需要输入指定的message：`bye!`。信息在发送到服务端之后，聊天室会在公共区域告知：“某某某退出聊天室！”。

五、实验过程中遇到的问题及分析

服务端信息发送给客户端时，出现发送空值



错误代码：

```
while (1)
{
    // 客户端发送指定格式内容给服务端
    ......

    //接收客户端回传消息
    char M[512];
    memset(M, 0, sizeof(M));
    if (recv(client, M, sizeof(M), 0) <= 0)
    {
        cout << "服务端数据接收失败！！！" << endl;
        break;
    }
    cout << M << endl;
}
```

在错误代码中，客户端接收服务端收到信息时用的数组M用的是[512]大小的，将其修改为[1024]字节便可解决问题。

分析错误原因：由于服务端在传信息给客户端用的buffer是[1024]大小的，而客户端接收时用的是[512]的。导致了服务端第一次传回来的返回信息在前[512]个长度内，可以接收输出到。之后第二次接收到的会是上一次的后[512]个长度内的内容，这段内容没有内容。故客户端接收到的信息为空白。