

# 计算机网络聊天程序实验报告

姓名：徐文斌 学号：2010234

## 一.协议设计

为了便于实现群聊等功能，本次聊天程序实验我设计了一个适用于C/S模型的应用层协议。协议基于TCP传输层协议，具体协议内容如下所示。

### 1.消息类型

消息类型方面，协议中一共有四种类型的报文，分别为in类型、out类型、answer类型、message类型。为了提高报文的可解析性和可扩展性，这里我将上述四种报文设计为如下格式：

```
报文类型 [状态字符串]
字段名:字段内容
...
字段名:字段内容
```

其中报文头由标识报文类型的字符串和可选的状态字符串组成（当前只有在answer类型报文中才会用到状态字符串）。报文头后跟若干字段行，由字段名和字段具体内容组成，用于描述报文的一些具体信息，如发送报文的用户、时间、消息内容等。报文头与字段行以及字段行之间均通过换行隔开，这样便于我们解析和构造报文。如果想要扩展某一类型的报文，只需要在报文中新增相应的字段行即可。

### 2.消息语法和语义

- in类型的报文语法如下(xxx表示不确定)：

```
in
name:xxx
```

in类型报文为用户请求进入聊天室时的请求报文。in类型报文第一行为消息头，固定为"in"字符串，以表示消息的类别；in类型报文的第二行为name字段，其固定由字符串"name:"开头，后跟用户的名字，表示请求进入聊天室的用户的名字。

- out类型的报文语法如下(xxx表示不确定)：

```
out
name:xxx
```

out类型报文为用户离开聊天室时发送的报文。out类型报文第一行为消息头，固定为"out"字符串，以表示消息的类别；out类型报文的第二行为name字段，其固定由字符串"name:"开头，后跟用户的名字，表示离开聊天室的用户的名字。

- answer类型的报文语法如下(xxx表示不确定)：

```
answer xxx
```

answer类型报文为服务端为响应客户端发来的in类型报文而发回给客户端的报文。其只有一行，该行开始为一个"answer"字符串，用以表示报文的类型。"answer"后跟对响应结果，结果可以为"accept"或"nameconflict"，表示同意用户进入聊天室或者用户名和已有用户名冲突。

- message类型的报文语法如下(xxx表示不确定):

```
message
name:xxx
time:xxx
sendto:xxx
msg:xxx
```

message类型报文用于客户端发送聊天信息。message类型报文第一行为消息头，固定为"message"字符串，以表示消息的类别；报文的第二行为name字段，其固定由字符串"name:"开头，后跟发送消息的用户的名字；报文的第三行为time字段，其固定由字符串"time:"开头，后跟用户发送消息的时间；报文的第四行为sendto字段，其固定由字符串"sendto:"开头，后跟用户发送消息的对象，若属性为"all"，表示用户群发消息，若属性为某一用户名，表示用户向该用户名对应的用户私发消息；报文的最后一行为msg字段，其固定由字符串"msg:"开头，后跟用户发送的具体信息。

## 2.消息时序

这里根据一个客户端从与服务端建立TCP连接后到离开聊天室的过程来详细地介绍协议中消息的时序。

1. 进入聊天室：客户端与服务端建立TCP连接后，其首先应向服务端发送一个in类型报文，报文中含有用户名，以向服务端申请进入聊天室，然后开始等待来自服务端的answer报文。服务端接收到来自客户端的in报文之后，根据当前聊天室中的用户名情况，检查是否存在用户名冲突，若无用户名冲突，则服务端向发起请求的客户端发送报文"answer accept"，之后将该客户端发过来的in报文直接转发给其他所有在聊天室中的客户端，用以通知其余客户端新客户端的加入；若有用户名冲突，则服务端向发起请求的客户端发送报文"answer nameconflict"。发起请求的客户端接收到answer报文后，对报文进行解析，若为"answer accept"报文，则客户端进入到下一步与其他用户聊天的状态；若为"answer nameconflict"报文，则客户端会请求用户重新输入名字，并重复上述请求过程，直到接收到"answer accept"报文。
2. 与其他用户聊天：客户端接收到"answer accept"报文后，开始进入聊天室与其他用户聊天。在此期间，客户端可以向服务器发送message报文，以通过服务器将消息转发给其他客户端。当服务器收到message报文后，其会对报文进行解析，识别出报文中的sendto字段，若该字段的属性为"all"，则服务器会将该message报文直接转发给其余所有客户端；若该字段的属性为某一个用户的用户名，则服务器会将该message报文直接转发给相应的单个用户，以实现私聊功能。在此期间，一个客户端可能接收到三种类型的报文，分别为in类型、out类型、message类型，客户端程序会对它们进行解析，在控制台中打印出相应信息。
3. 离开聊天室：当客户端接收到用户的退出指令时，其会向服务器发送一个out报文，报文中含有该用户的用户名，以通知服务器该用户退出聊天室。当服务器收到一个out报文后，其会注销掉该用户占用的资源，并将来自该用户的out报文直接转发给聊天室中的所有用户，以通知他们该用户退出聊天室。

## 二.程序各模块功能

首先说明一下自定义的结构体类型Message。如下所示，可以看到Message结构体的定义和我们上边实现的报文格式是十分相似的。考虑到send函数和recv函数的消息缓冲区均是一个字符数组，对字符数组的操作有些繁琐。这里定义该结构体作为中转，当我们需要调用send函数的时候，首先构造一个Message类型的结构体变量，向变量中填入相应内容，然后调用construct\_message函数将该Message变量转化为一个字符串类型的变量，用于作为send

的参数。当我们调用过recv函数后，为了便于后续的处理，会调用parse\_message函数，将recv得到的字符数组转化为一个Message类型的变量，从而方便我们后续提取报文中的不同的字段并进行相应的处理。

```
struct Message {
    string type;
    string name;
    string time;
    string sendto;
    string msg;
    Message() {
        type = "message";
        name = time = sendto = msg = "";
    }
};
```

## 1.客户端模块

客户端主要的函数为：

- main函数：程序的入口点，负责创建socket，与服务器建立连接。之后调用try\_to\_enter\_room函数申请进入聊天室。然后创建监听线程，线程函数为recv\_thread。最后调用send\_msg\_to\_server函数，用于读取用户输入数据，实现客户端发送报文到服务器。
- try\_to\_enter\_room(SOCKET\*)函数：该函数主要实现申请进入聊天室的功能，即请求用户输入用户名，然后发送in类型报文给服务器，之后等待服务器发回answer类型报文，循环该过程直到服务器发回"answer accept"报文后函数执行结束。
- recv\_thread线程函数：该函数参数为socket指针。该函数通过循环不断监听来自服务器的报文。接收到一个报文后，调用parse\_message函数对得到的报文进行分析生成Message类型变量，之后根据报文的类型和内容在控制台中打印出不同的信息。
- send\_msg\_to\_server(SOCKET\*)函数：该函数通过循环不断得到来自用户的控制台输入，对用户的输入进行解析生成Message类型变量，之后调用construct\_message函数生成字符串数组形式的报文，并将报文发送给服务器。
- parse\_message(char\*)函数：由于从recv函数获得的数据以一整个字符串的形式存放在字符串数组中，可读性较差。定义该函数将传入的报文字符串转化为自定义的Message类型的结构体，从而便于对报文类型以及报文内容的获取。
- construct\_message(Message)函数：由于send函数发送数据需要将报文放入一个字符串数组中，直接构建字符串数组有些繁琐，我们先生成Message类型变量，再将该变量转为字符串数组的形式，该函数便负责将传入的结构体变量转化为字符串类型，用以为send函数准备参数。

## 2.服务端模块

- main函数：程序的入口点，负责创建监听socket，绑定相应的端口号后，通过while循环不断地接受来自客户端的连接。每接受一个连接就新创建一个线程用于获得该连接上来自客户端的报文，线程函数为recv\_thread。
- recv\_thread函数：该线程函数参数为socket指针。与客户端程序的recv\_thread函数类似，该函数通过循环不断监听来自客户端的报文。接收到一个报文后，调用parse\_message函数对得到的报文进行分析生成Message类型变量，之后根据报文的类型和内容打印不同的日志记录，并创建报文转发线程将报文转发给不同的用户，报文转发线程的线程函数为send\_to\_users。
- send\_to\_users函数：该线程函数的参数为一个自定义的结构体类型Wrap\_Data的变量指针。该函数可以根据传入参数的内容，来选择性地将报文转发给不同的用户。
- parse\_message(char\*)函数：由于从recv函数获得的数据以一整个字符串的形式存放在字符串数组中，可读性较差。定义该函数将传入的报文字符串转化为自定义的Message类型的结构体，从而便于对报文类型以及报文

内容的获取。

- `construct_message(Message)`函数：由于`send`函数发送数据需要将报文放入一个字符数组中，直接构建字符数组有些繁琐，我们先生成`Message`类型变量，再将该变量转为字符数组的形式，该函数便负责将传入的结构体变量转化为字符串类型，用以为`send`函数准备参数。

下面解释一下服务器中`send_to_users`线程函数的实现。首先，上述提到了自定义的结构体`Wrap_Data`如下所示。其中`message`存储要转发给其他客户端的报文的内容，`client_from`为向服务器发送该报文的客户端id号，`client_to`为服务器要转发到的客户端的id号。若`client_to`为-1，表示是群发。

```
struct Wrap_Data {  
    string message;  
    int client_from;  
    int client_to;  
};
```

`send_to_users`线程函数具体代码如下所示。其首先获得传入的参数，如果`client_to`为-1，则遍历当前连接的所有客户端，向他们转发报文。如果`client_to`不为-1，则函数只向id为`client_to`的客户端转发报文。

```
DWORD WINAPI send_to_users(LPVOID lpParam) {  
    char send_buff[BUFFER_SIZE];  
    Wrap_Data wpd = *(Wrap_Data*)lpParam;  
    delete (Wrap_Data*)lpParam;  
    int client_from = wpd.client_from;  
    int client_to = wpd.client_to;  
    strcpy_s(send_buff, wpd.message.data());  
    for (auto it = used_ids.begin(); client_to == -1 && it != used_ids.end(); it++) {  
        if (client_from == *it || id2name[*it] == "") {  
            continue;  
        }  
        int send_byte = send(*id2socket[*it], send_buff, sizeof(send_buff), 0);  
    }  
    if (client_to != -1) {  
        int send_byte = send(*id2socket[client_to], send_buff, sizeof(send_buff), 0);  
    }  
    return 0;  
}
```

## 三.程序界面展示及运行说明

### 1.开启服务端

如下图所示，服务器首先创建`socket`。然后请求用户输入端口号，这里如果用户不输入端口号直接回车，服务器将会绑定默认端口号2333。绑定端口成功后服务器将开始监听`socket`。

```
F:\C++\Server\Debug\Server.exe X + v
2022-10-22 10:54:37: 创建socket成功
请输入服务器绑定的端口号(默认为2333):
2022-10-22 10:55:02: 服务器端口绑定成功
2022-10-22 10:55:02: 成功进入监听状态
|
```

## 2. 开启客户端1

如下图所示，开启客户端时，客户端首先创建socket。然后请求用户输入服务器的IP和端口号，这里如果用户不输入IP或端口号而是直接回车，则客户端将会默认与127.0.0.1:2333处的服务器建立连接。成功建立连接后，客户端会请求用户输入用户名，用户输入不与已有用户冲突的用户名后便可以成功进入聊天室。

```
F:\C++\Client\Debug\Client.exe X + v
创建socket成功
请输入服务器的IP4地址(默认为127.0.0.1):
请输入服务器的端口号(默认为2333):
连接服务器成功
请输入姓名: a
成功进入聊天室
a> |
```

下图为客户端1连接服务器时服务器打印出的信息。服务器会打印它所收到的报文以及它所发送出去的报文。可以看到，客户端1连接服务器时，服务器首先收到了客户端1的in类型请求报文，然后服务器经过判断允许客户端1进入聊天室，向客户端1发送answer报文。并向其他所有客户端发送in类型报文，以通知其他客户端客户端1的进入（当

然，当前只有客户端1，所以这里实际上并没有向任何客户端发送in类型报文）。

```
F:\C++\Server\Debug\Server.e  X  +  v
2022-10-22 10:54:37:创建socket成功
请输入服务器绑定的端口号(默认为2333):
2022-10-22 10:55:02:服务器端口绑定成功
2022-10-22 10:55:02:成功进入监听状态
2022-10-22 10:59:36:接收到来自a的报文:
in
name:a

2022-10-22 10:59:36:a加入聊天室

2022-10-22 10:59:36:向用户a发送报文:
answer accept

2022-10-22 10:59:36:向所有用户（除了a）发送报文:
in
name:a
```

### 3.开启客户端2

下面我们开启客户端2，其过程和开启客户端1是类似的。这里简单略过。

```
F:\C++\Client\Debug\Client.e  X  +  v
创建socket成功
请输入服务器的IP4地址(默认为127.0.0.1):
请输入服务器的端口号(默认为2333):
连接服务器成功
请输入姓名: b
成功进入聊天室
b> |
```

服务器输出日志也与开启客户端1时一致。

```
2022-10-22 11:13:17:接收到来自b的报文:
in
name:b

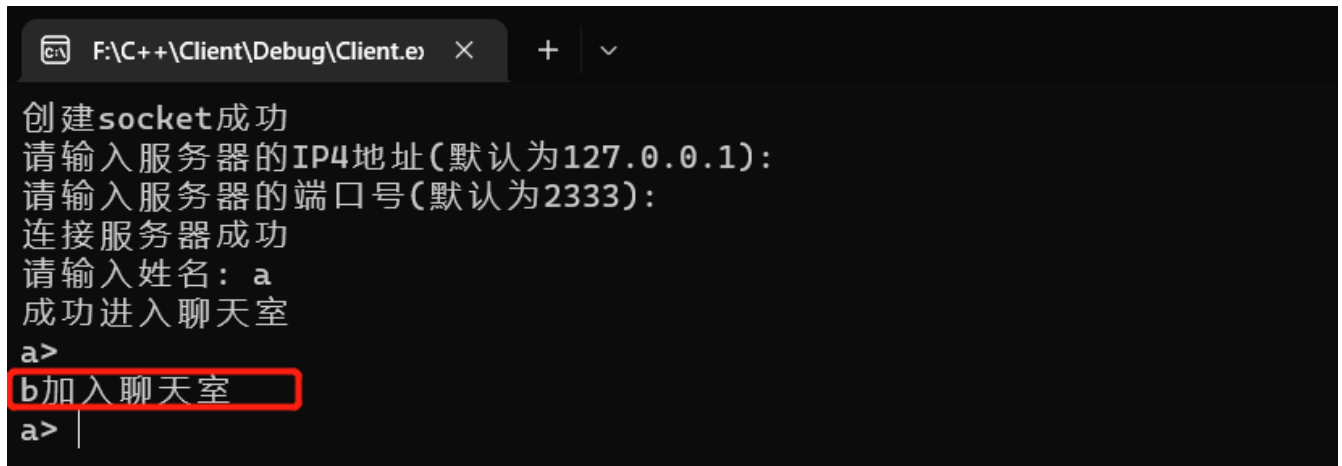
2022-10-22 11:13:17:b加入聊天室

2022-10-22 11:13:17:向用户b发送报文:
answer accept

2022-10-22 11:13:17:向所有用户（除了b）发送报文:
in
name:b
```

下图为开启客户端2时客户端1控制台打印出的内容，客户端1接收到来自服务器的in类型报文后，会在控制台打印语

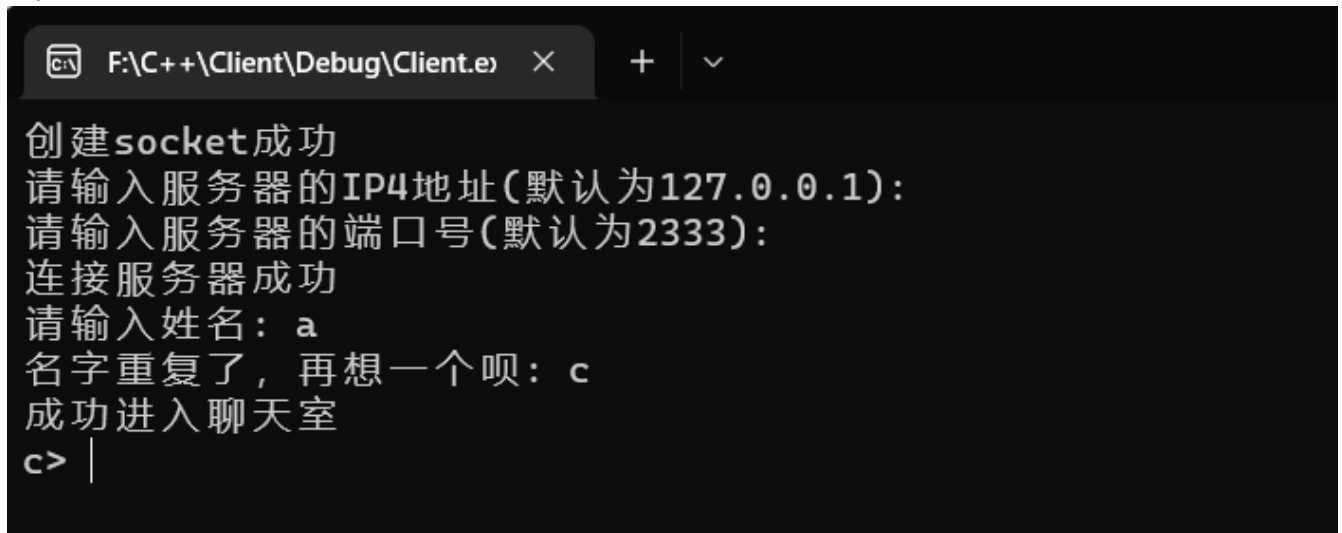
句以提示用户新用户的加入。



```
F:\C++\Client\Debug\Client.e
创建socket成功
请输入服务器的IP4地址(默认为127.0.0.1):
请输入服务器的端口号(默认为2333):
连接服务器成功
请输入姓名: a
成功进入聊天室
a>
b加入聊天室
a> |
```

## 4.开启客户端3

这里我们体现一下，如果用户请求进入聊天室时输入的用户名和已进入聊天室的用户名冲突时的效果。如下图所示，我们开启客户端3，输入用户名时输入客户端1的用户名，客户端3发送in类型报文后将会接收到来自服务器的"answer nameconflict"报文，其将会提示用户输入的名字重复了，请求用户重新输入名字。若用户再次输入的名字不与已存在的用户名冲突，客户端3将接收到"answer accept"报文，提示该用户成功进入聊天室。否则将会循环该过程。



```
F:\C++\Client\Debug\Client.e
创建socket成功
请输入服务器的IP4地址(默认为127.0.0.1):
请输入服务器的端口号(默认为2333):
连接服务器成功
请输入姓名: a
名字重复了, 再想一个呗: c
成功进入聊天室
c> |
```

服务器日志如下，可以看到，服务器接收到客户端3的报文后，发现客户端3请求的用户名和已有用户名重复，则服务器会向客户端3发送"answer nameconflict"报文，以通知客户端3申请的结果。当客户端3请求用户重新输入用户名后，会再次向服务器发送in类型报文，这个时候服务器检测到客户端3申请的名字不与已有名字重复，会向客户端3发送"answer accept"报文，并向其他用户发送in类型报文以通知客户端3的进入。



```
2022-10-22 11:22:13:接收到来自a的报文:
in
name:a

2022-10-22 11:22:13:a名字重复拒绝进入聊天室

2022-10-22 11:22:13:向用户a发送报文:
answer nameconflict

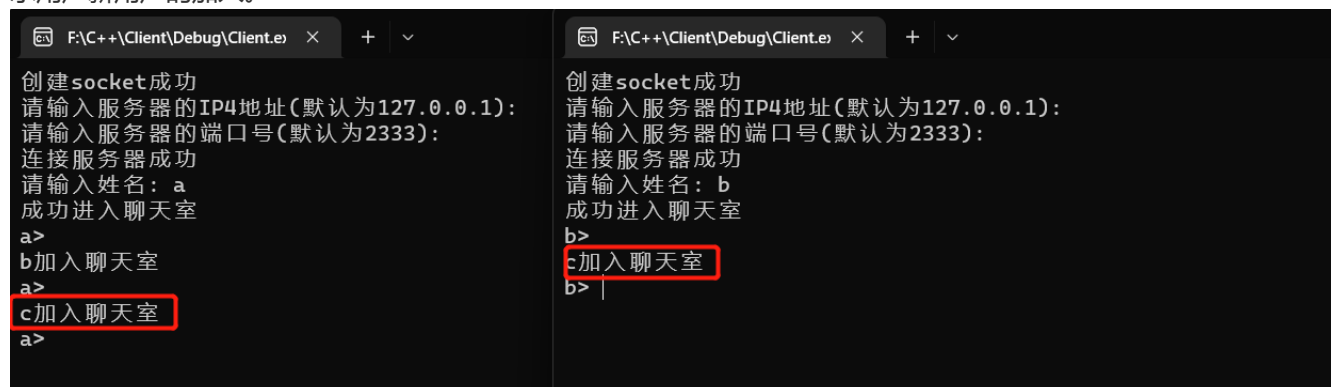
2022-10-22 11:23:57:接收到来自c的报文:
in
name:c

2022-10-22 11:23:57:c加入聊天室

2022-10-22 11:23:57:向用户c发送报文:
answer accept

2022-10-22 11:23:57:向所有用户（除了c）发送报文:
in
name:c
```

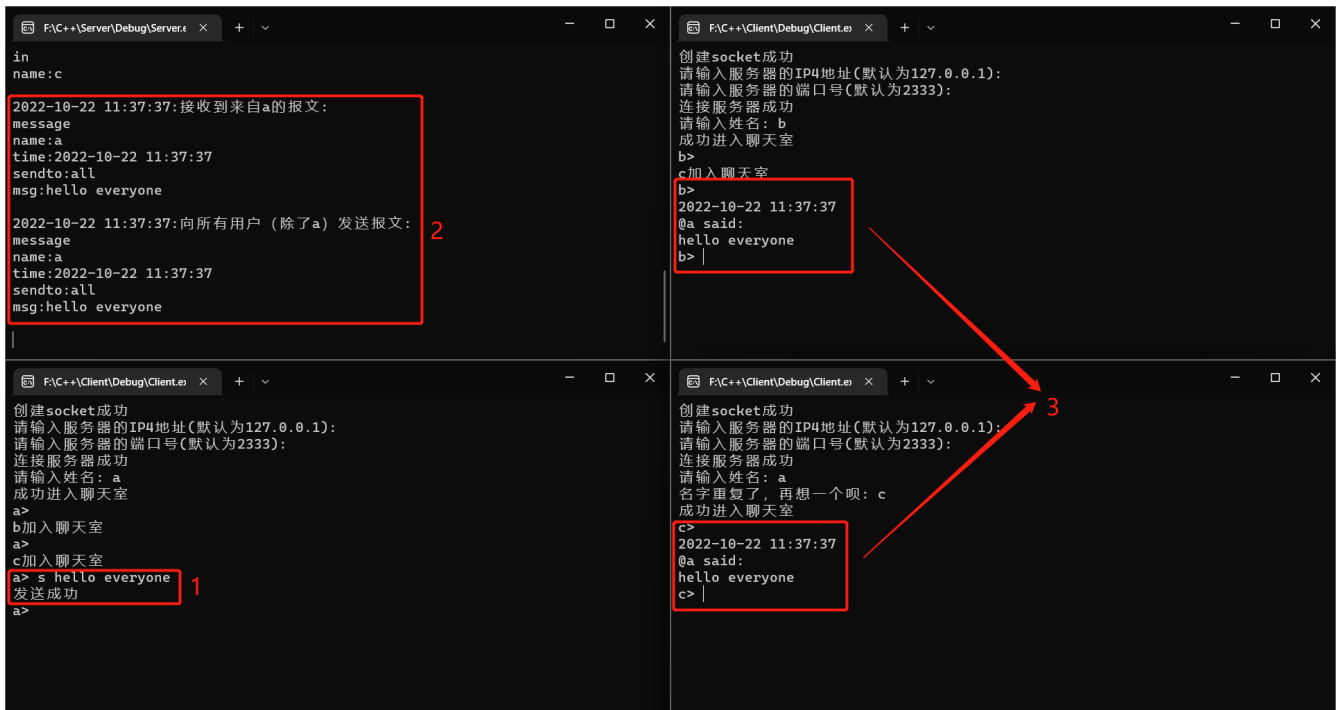
可以看到，客户端3成功进入聊天室的时候，客户端1和客户端2接收到相应的报文后，会在控制台打印相关信息以提示用户新用户的加入。



## 5.客户端进行群聊

首先说明一下群聊信息的发送格式，格式为"s msg"，s（send的简写）表示用户要群发消息，msg是消息的具体内容，两者用单个空格隔开。如下图所示，用户a在控制台中输入"s hello everyone"，客户端得到输入后会解析该字符串，生成对应的message报文发送给服务器。可以看到，左上角服务器成功接受了来自用户a的报文，并将其转发给除了a以外的所有用户。其他的客户端接收到message报文后，会对其进行解析，并打印相应的消息，如图片右侧所示，客户端2和客户端3接收到报文后在控制台打印出了用户a发送的消息。





## 6.客户端进行私聊

首先说明私聊信息的发送格式，格式为"st username msg", st (send to的简写)表示用户要私发消息，username是私发对象的用户名，msg是私发的具体消息内容，这三者用单个空格隔开。如下图所示，用户a在控制台输入"st b nihao", 表示向用户b发送消息"nihao", 客户端对输入进行解析后向服务器发送对应的message报文，服务器根据报文中的"sendto"字段将message只转发给用户b，用户b的客户端接收到报文后会在控制台打印出相应信息。可以看到，下图中只有用户b收到了"nihao", 用户c没有收到该消息。



## 7.客户端退出聊天室

最后是客户端退出聊天室功能的展示。用户只需在控制台输入"exit", 客户端得到输入后，在退出聊天室之前会向服务器发送out类型报文。服务器收到该报文后会注销该客户端的一系列资源，并将报文转发给其他客户端，以通知其他客户端该用户的离开。下图为用户a输入"exit"的退出过程。

```
F:\C++\Server\Debug\Server.exe
2022-10-22 11:51:47:向用户b发送报文:
message
name:a
time:2022-10-22 11:51:47
sendto:b
msg:nihao
2022-10-22 12:04:16:接收到来自a的报文:
out
name:a
2022-10-22 12:04:16:用户a离开聊天室
2022-10-22 12:04:16:向所有用户发送报文:
out
name:a

F:\C++\Client\Debug\Client.exe
请输入服务器的IP4地址(默认为127.0.0.1):
请输入服务器的端口号(默认为2333):
连接服务器成功
请输入姓名: b
成功进入聊天室
b>
c加入聊天室
b>
2022-10-22 11:37:37
@a said:
hello everyone
b>
2022-10-22 11:51:47
@a said to you:
nihao
b>
a离开聊天室
b>

F:\C++\Client\Debug\Client.exe
创建socket成功
请输入服务器的IP4地址(默认为127.0.0.1):
请输入服务器的端口号(默认为2333):
连接服务器成功
请输入姓名: a
成功进入聊天室
a>
b加入聊天室
a>
c加入聊天室
a> s hello everyone
发送成功
a> st b nihao
发送成功
a> exit
发送成功
ByeBye
请按任意键继续...

F:\C++\Client\Debug\Client.exe
创建socket成功
请输入服务器的IP4地址(默认为127.0.0.1):
请输入服务器的端口号(默认为2333):
连接服务器成功
请输入姓名: a
名字重复了,再想一个呗: c
成功进入聊天室
c>
2022-10-22 11:37:37
@a said:
hello everyone
c>
a离开聊天室
c>
```

## 四.实验中遇到的问题及分析

### 1.对send函数和recv函数的理解

send函数和recv函数是在网络通信程序中使用频度比较高,相对来说比较重要的两个函数。对它们行为的正确理解是实验中较为关键的一部分。网络编程中socket有阻塞模式和非阻塞模式,本实验中我使用阻塞模式的socket。下面是对阻塞模式下send函数和recv函数的学习与理解。

#### 何为阻塞模式

阻塞模式,就是当某个函数“执行成功的条件”当前不能满足时,该函数会阻塞当前执行线程,程序执行流在超时时间到达或“执行成功的条件”满足后恢复继续执行。

#### send函数和recv函数做了什么

- send函数本质上并不是往网络上发送数据,而是将应用层发送缓冲区的数据拷贝到内核缓冲区(即TCP窗口)中去,至于什么时候数据会从缓冲区中真正地发到网络中去要根据TCP/IP协议栈的行为来确定。
- recv函数本质上也并不是从网络上收取数据,而只是将内核缓冲区中的数据拷贝到应用程序的缓冲区中,拷贝完成以后会将内核缓冲区中该部分数据移除。

上述两条很好地体现了网络中的分层的思想,上层使用下层提供的服务,即send和recv这两个应用层调用的函数使用传输层提供的端到端传输服务。

#### send函数和recv函数阻塞模式下的行为

对于send函数来说,在阻塞模式下,如果接收方迟迟不从内核缓冲区中接收数据或其接收数据的速度较慢,而发送方一直调用send函数向其内核缓冲区中拷贝数据,将会导致接收方和发送方的内核缓冲区都被填满,此时如果发送方继续调用send函数,send函数将会阻塞当前的程序流。

对于recv函数来说,在阻塞模式下,如果接收方的内核缓冲区中没有数据,则其调用recv函数时,recv函数将会阻塞当前的程序流。