Lab1: Socket聊天程序

李思凡 2010239

一、消息处理方式

消息类型

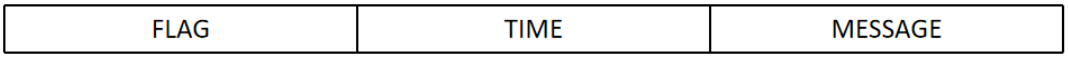
该聊天程序的消息类型共有5种，为**NAME，MODE，EXIT，NORM，LOSE**。它们各自的含义如下：

1. NAME：指示该客户端对应的用户名称的消息
2. MODE：指示该客户端选择的聊天模式的消息（私聊或群聊）
3. EXIT：指示该客户端要退出的消息
4. NORM：指示该客户端发送给对方的聊天内容的消息
5. LOSE：服务器发送给客户端，指示该客户端想要私聊的对象未在线的消息

消息语法

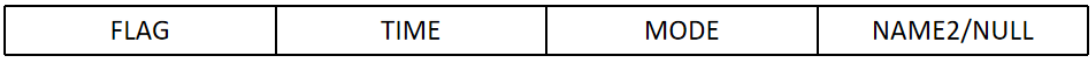
各类型消息对应的语法如下，总长度为1024位。

对于**NAME, EXIT, NORM, LOSE**消息，其格式如下：



其中，FLAG表示消息的类型，占4位，为“NAME”，“EXIT”，“NORM”，“LOSE”；TIME表示发送该消息的时间，占8位；MESSAGE表示发送的消息的具体内容，占1012位。

对于**MODE**消息，其格式如下：



其中，FLAG表示消息类型，为“MODE”；TIME表示发送该消息的时间，占8位；MODE表示是私信还是群聊，占1位；对于私信，NAME2表示私信的对象，对于群聊，该段为空。

消息语义

对于所有消息，FLAG代表该消息所属类型；TIME表示发送时间，为“hh:mm:ss”格式。

对于NAME，其MESSAGE段表示的是客户端的用户名称，长度不超过20；

对于EXIT，其MEESAGE字段为空；

对于LOSE，其MESSAGE字段为服务端返回给客户端的信息，表示其私信对象未在线；

对于NORM，其MESSAGE字段由<发送端名称>+<[私信/群聊]：>+<输入内容>组成；

对于MODE消息，其MODE字段为a或b，a表示私信模式，b表示群聊模式；对于私信模式，NAME2为其私信对象的名称，长度不超过20，对于群聊模式，该段为空。

时序

1. 在客户端首次登入时，会提示输入名称，此时输入名字，将包装成NAME消息发送给服务器端。服务器收到后将建立该客户端Socket和该客户端名字的连接，提示加入成功。
2. 正确输入PRIVATE后，将提示输入私信对象的名称。客户端将根据是否处于与该对象的私信状态、判断是否进行状态切换。若已经处于与该对象的私信状态，则提示并不做处理；否则将其包装成MODE消息发送给服务器端，服务器收到后将保存该客户端通信状态为私信，并建立该客户端名称和私信对象名称的联系，提示切换成功。若未正确输入PRIVATE，则提示重新输入。
3. 正确输入PUBLIC后，客户端将根据是否处于群聊状态、判断是否进行状态切换。若已经处于群聊状态，则提示并不做处理；否则将其包装成MODE消息发送给服务器端，服务器收到后将保存该客户端通信状态为群聊，提示切换成功。若未正确输入PUBLIC，则提示重新输入。
4. 正确输入EXIT后，客户端将提示是否确认退出，此时输入y表示确认退出，将其包装为EXIT消息发送给服务器端，服务器端收到后将置空该客户端的Socket信息，并关闭该Socket，客户端退出；若输入n表示取消退出，将不会退出。
5. 输入其他内容则表示为正常的聊天信息，将其包装成NORM消息发送给服务器端。服务器端收到后，将查看该客户端的通信模式。若为私信模式，将判断其私信对象是否在线，若不在线，则包装成LOSE信息返回给该客户端，若在线，则将该消息转发给对应的客户端。若为群聊信息，则将该消息转发给在线的客户端。

二、各模块功能

客户端

客户端接受消息

客户端的一个线程用来接受消息。收到消息后，根据消息的格式切分出消息标识头、时间和具体内容。客户端能收到的消息有LOSE和NORM两种类型，收到后进行打印。

```
DWORD WINAPI handleRequest(LPVOID lparam) {
    SOCKET clientSocket = (SOCKET)(LPVOID)lparam;
    int result = 0;
    char recvBuf[1024];
    while (1) {
        result = recv(clientSocket, recvBuf, MSIZE, 0);
        if (result == SOCKET_ERROR) {
            printLog(ERRO, "Receive Error! ");
            break;
        }
        string strBuf = recvBuf;
        string recvFlag = strBuf.substr(0, 4); //获取消息头
        string recvTime = strBuf.substr(4, 8); //获取时间信息
        string recvMess = strBuf.substr(12, strBuf.size() - 12); //获取消息内容
        if (recvFlag.compare("LOSE")==0) {
            cout << recvTime << " ";
            printLog(RECV, recvMess);
        }
        if (recvFlag.compare("NORM")==0) {
            cout << recvTime << " ";
            printLog(RECV, recvMess);
        }
    }
    return 0;
}
```

客户端发送消息

客户端的另一个线程用来发送消息。根据输入的内容对消息的类型进行判断，包装成相应的类型后发送给服务器端。例如，发送私信模式的消息，部分代码如下所示。

```
else if (!strcmp(getMessage, "PRIVATE")) {
            if (mode == 'a') {
                memset(getInput, 0, GSIZE);
                printLog(GETI, "Input his or her name, no more than 20
characters: ");
                if (!strcmp(getInput, name2)) {
                    printLog(INFO, "You are already at PRIVATE mode with this
one now!");
                }
                else {
                    char temp[NSIZE];
                    strcpy(temp, name2);
                    memset(name2, 0, NSIZE);
                    strcpy(name2, getInput);
                    memset(message, 0, MSIZE);
                    strcat(message, "MODE");
                    char tmp[10];
                    time_t t = time(NULL); //获取当前时间
                    tm* curr_tm = localtime(&t); //使用t来填充curr_tm结构
                    int hh = curr_tm->tm_hour;
                    int mm = curr_tm->tm_min;
                    int ss = curr_tm->tm_sec;
                    sprintf(tmp, "%02d:%02d:%02d", hh, mm, ss);
                    strcat(message, tmp);
                    strcat(message, "a");
                    strcat(message, name2);
                    result = send(clientSocket, message, MSIZE, 0);
                    if (result == SOCKET_ERROR)
                    {
                        printLog(ERRO, "Choose Fail! ");
                        strcpy(name2, temp);
                        continue;
                    }
                    memset(cstr, 0, MSIZE);
                    strcpy(cstr, name);
                    strcat(cstr, ": choose to chat with ");
                    strcat(cstr, name2);
                    printLog(SUCC, cstr);
                }
            }
            else {
                result = changeToPrivate(clientSocket);
                if (result == -1) {
                    memset(name2, 0, NSIZE);
                    continue;
                }
            }
        }
```

客户端打印日志

客户端根据不同类型的信息和消息，在终端输出相应的日志。例如错误采用ERRO，成功连接采用SUCC，发送采用SEND，接受采用RECV等。其中调用printTime()函数，会获取当前的时间，并按照格式打印。

```
void printLog(LOG log, string str) {
    switch (log) {
    case INFO:
        printTime();
        cout << "[ INFO ] " << str << endl;
        break;
    case NAME:
        printTime();
        cout << "[ NAME ] " << str ;
        cin.getline(name, sizeof(name));
        break;
    case SUCC:
        printTime();
        cout << "[ SUCC ] " << str << endl;
        break;
    case SEND:
        printTime();
        cout << "[ SEND ] " << str << endl;
        break;
    case RECV:
        cout << "[ RECV ] " << str << endl;
        break;
    case ERRO:
        printTime();
        cout << "[ ERRO ] " << str << endl;
        break;
    case GETI:
        printTime();
        cout << "[ GETI ]" << str;
        cin.getline(getInput, sizeof(getInput));
        break;
    }
}
```

服务器端

服务器接收客户端连接

使用accept对于相应的客户端接受连接，并将其SOCKET保存到数组中。

```
SOCKET sockConn = accept(serverSocket, (SOCKADDR*)&clientAddrIn,
&clientAddrLen); //返回用来与客户端通信的socket
        if (sockConn == INVALID_SOCKET)
        {
            printLog(ERRO, "Accept Fail! ");
            break;
        }
        cnum++;
        printLog(SUCC, "Accept Succeed! ");
```

```
        for (int i = 0; i < CSIZE; i++) { //将生成的与客户端通信的socket添加到数组中保
存
            if (clientArray[i] == NULL) {
                clientArray[i] = sockConn;
                clientMap[sockConn] = i;
                string str = to_string((long long)sockConn);
                str += " joins.";
                printLog(JOIN, str);
                break;
            }
        }
```

服务器接收和发送消息

为每个客户端建立一个线程，进行消息的接收和转发。

服务器接受消息时，会根据消息的类型解析不同的消息，并做相应的处理。例如对于客户端发来的NORM类型的消息，部分代码如下所示：

```
if (recvFlag.compare("NORM")==0) {
            string str1 = "Receive From ";
            str1 += to_string((long long)clientSocket);
            str1 += ": ";
            str1 += recvMess;
            std::cout << recvTime << " ";
            printLog(RECV, str1);
            if (modeMap[nameMap[clientSocket]] == 'a') {
                string name2 = names[nameMap[clientSocket]];
                int flag = 0;
                for (int i = 0; i < NUM; i++) {
                    if (clientArray[i] != NULL &&
nameMap[clientArray[i]]==name2) {
                        char sendTo[CSIZE] = "NORM";
                        char tmp[10];
                        time_t t = time(NULL); //获取当前时间
                        tm* curr_tm = localtime(&t); //使用t来填充curr_tm结构
                        int hh = curr_tm->tm_hour;
                        int mm = curr_tm->tm_min;
                        int ss = curr_tm->tm_sec;
                        sprintf(tmp, "%02d:%02d:%02d", hh, mm, ss);
                        strcat(sendTo, tmp);
                        strcat(sendTo, recvMessC);
                        send(clientArray[i], sendTo, CSIZE, 0);
                        string str = "Send To ";
                        str += to_string((long long)clientArray[i]);
                        str += ": ";
                        str += recvMess;
                        printLog(SEND, str);
                        flag = 1;
                        break;
                    }
                }
                if (flag == 0) {
                    char sendBack[CSIZE] = "LOSE";
                    char tmp[10];
```

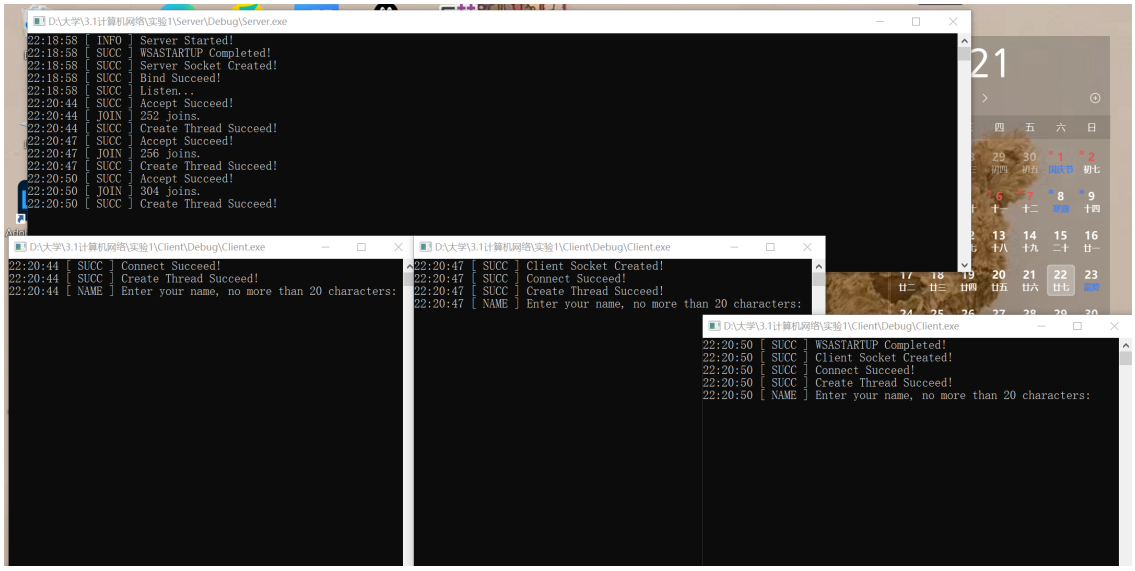
```
                    time_t t = time(NULL); //获取当前时间
                    tm* curr_tm = localtime(&t); //使用t来填充curr_tm结构
                    int hh = curr_tm->tm_hour;
                    int mm = curr_tm->tm_min;
                    int ss = curr_tm->tm_sec;
                    sprintf(tmp, "%02d:%02d:%02d", hh, mm, ss);
                    strcat(sendBack, tmp);
                    strcat(sendBack, name2.c_str());
                    strcat(sendBack, " Not Connected");
                    send(clientSocket, sendBack, CSIZE, 0);
                    string str2 = "Send To ";
                    str2 += to_string((long long)clientSocket);
                    str2 += ": ";
                    str2 += name2;
                    str2 += " Not Connected! ";
                    printLog(SEND, str2);
                }

            }
            else {
                for (int i = 0; i < NUM; i++) {
                    it_n = nameMap.find(clientArray[i]);
                    if (clientArray[i] != NULL && i != clientMap[clientSocket]
&& it_n!=nameMap.end()) {
                        char sendTo[CSIZE] = "NORM";
                        char tmp[10];
                        time_t t = time(NULL); //获取当前时间
                        tm* curr_tm = localtime(&t); //使用t来填充curr_tm结构
                        int hh = curr_tm->tm_hour;
                        int mm = curr_tm->tm_min;
                        int ss = curr_tm->tm_sec;
                        sprintf(tmp, "%02d:%02d:%02d", hh, mm, ss);
                        strcat(sendTo, tmp);
                        strcat(sendTo, recvMessC);
                        send(clientArray[i], sendTo, CSIZE, 0);
                        string str = "Send To ";
                        str += to_string((long long)clientArray[i]);
                        str += ": ";
                        str += recvMessC;
                        printLog(SEND, str);
                    }
                }
            }
        }
```

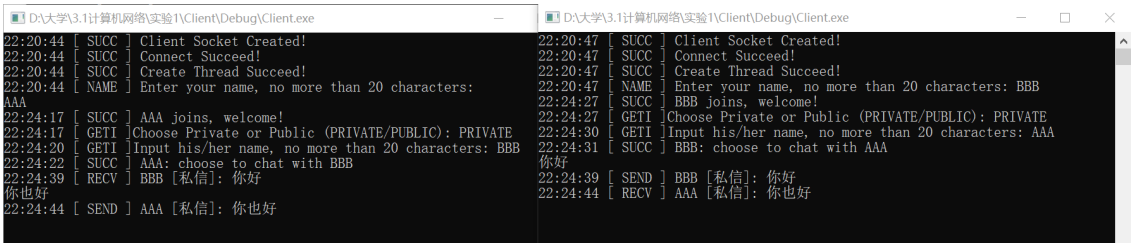
三、实现效果及说明

1. 首先启动服务器端，服务器进入监听状态

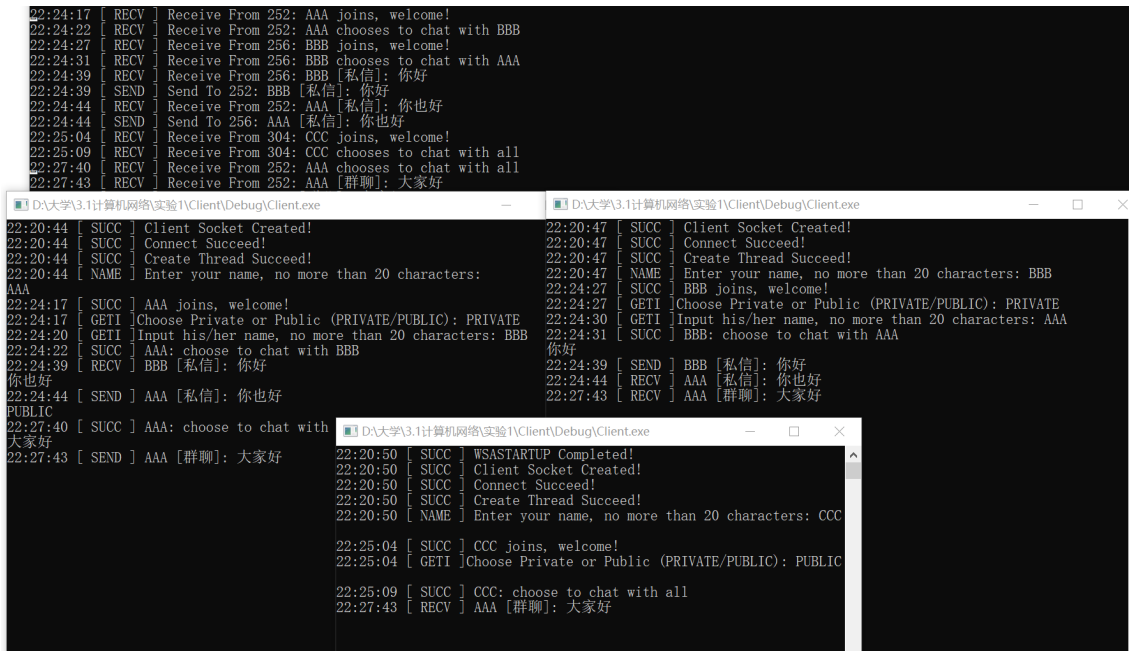
2. 以3个客户端做演示，点击客户端exe三次，显示3个客户端，服务器端显示它们加入。



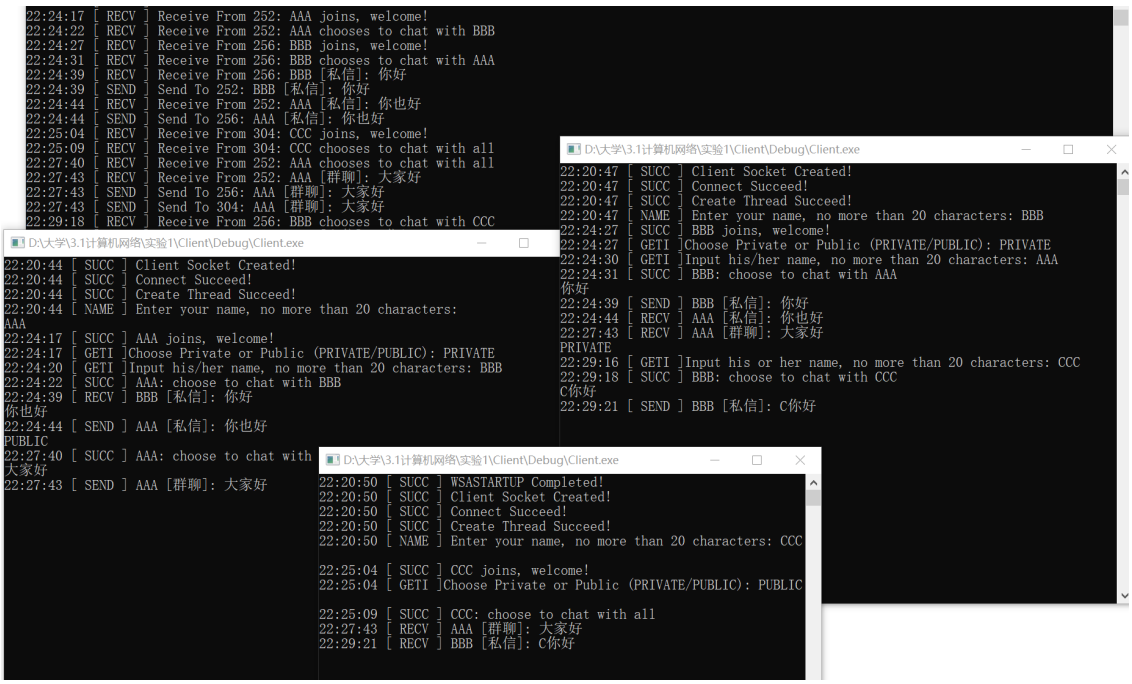
3. 演示私信，客户端1输入用户名AAA，选择私信模式，私信对象为BBB；客户端2输入用户名BBB，选择私信模式，私信对象为AAA。二者建立私聊，另一个客户端收不到这些私聊消息。



4. 演示群聊，客户端1输入PUBLIC，切换为群聊模式，此时输入消息大家好，另外两个客户端都能收到。

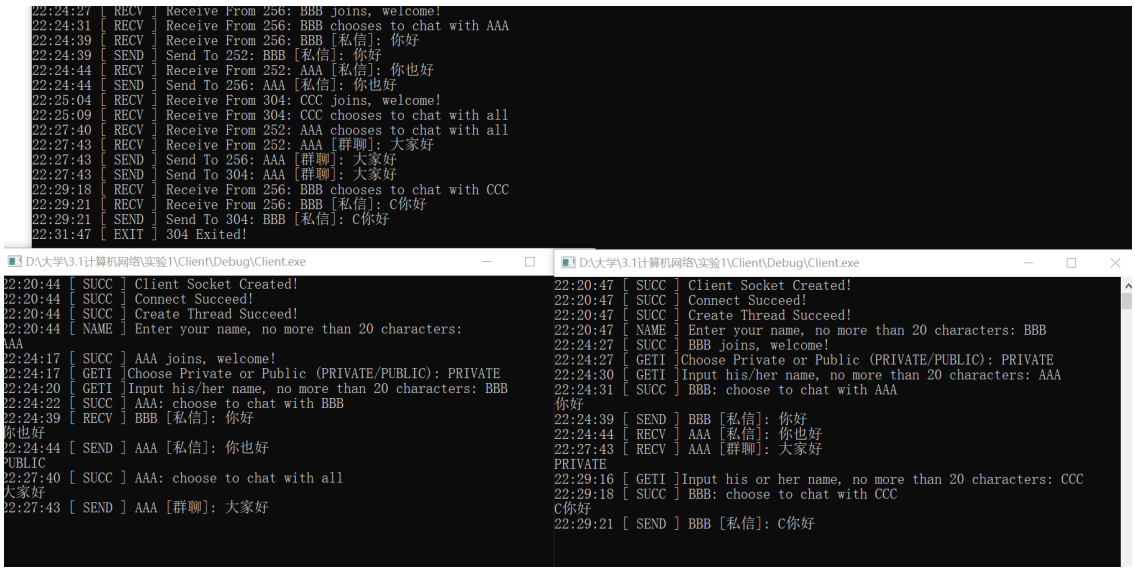


5. 演示切换，客户端BBB切换为和CCC私信，输入C你好，只有CCC收到消息。

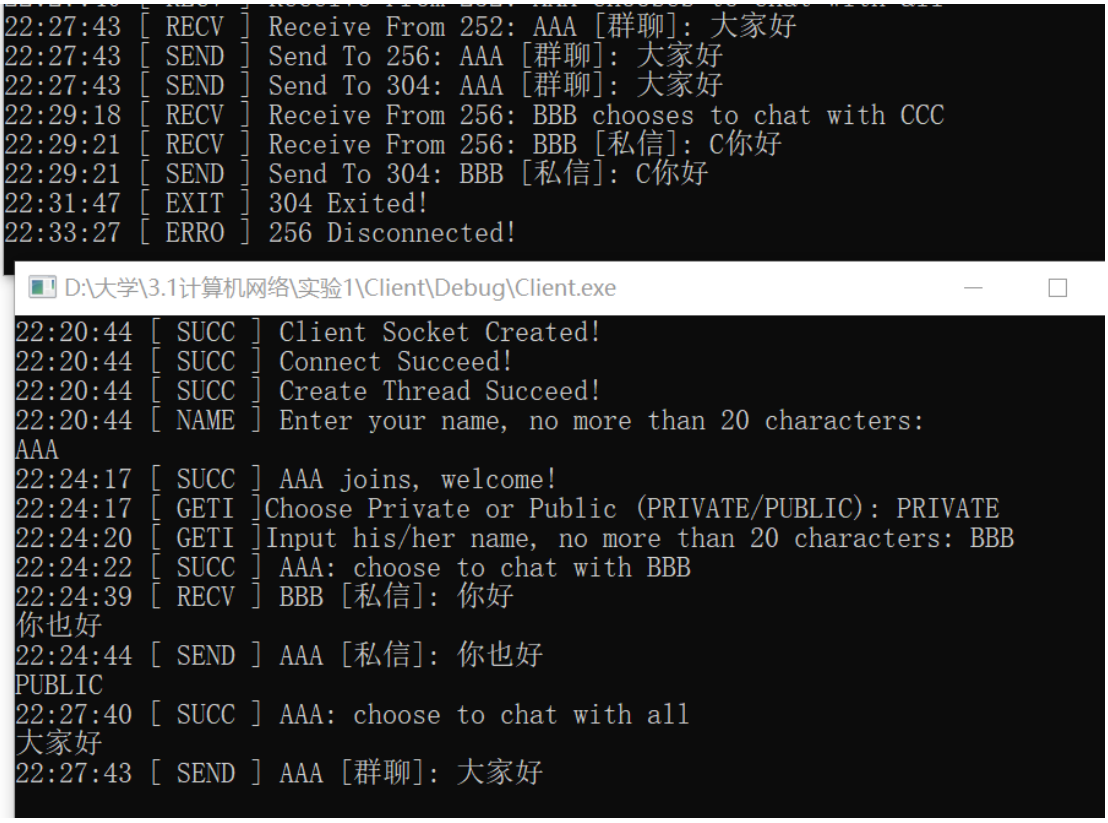


6. 演示退出，客户端C输入EXIT，进而输入y，实现退出。

服务器端收到，CCC退出



7. 直接关闭其中的客户端，显示Disconnect



四、实验过程中遇到的问题及分析

阻塞

如果不加入多线程，程序运行会遭遇阻塞的情况，两人聊天时，需要一方先发送再接受，而另一方先接收再发送，但无法使多人聊天正常进行。因此，客户端需要两个线程，一个处理接收，一个处理发送；服务器端为每个客户端创建线程，处理消息的接收和转发。

私信和群聊

私信和群聊的建立主要在服务器端。私信时，服务器需要保存两个客户端名称对应的SOCKET，将消息转发到对应的SOCKET；群聊则需要保存在线的所有客户端，将消息转发给它们。使用数组和数据结构map进行保存和关联。