

聊天程序设计与实现

2011269 王楠舟

实验目的

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。

可以进行协议设计拓展实现功能（如群聊、多线程等）。

实现基本功能

- Lab1_1版本中实现最基础的CS双端通话功能
- Lab1_2版本中使用一个Server端转发两个Client的消息，实现了一个双Client端通话功能
- Lab1_3版本中，使用在Server端利用多线程，实现多人聊天室功能，支持群发消息、私发消息的功能

由于报告篇幅有限，主要描述如何实现Lab1_3为主，因为三个版本之间大多数代码是由复用的。具体前两个版本的代码可以在我的GitHub上获取。

1_3支持的功能：

- 使用TCP作为传输层协议，使用流式套接字进行消息交互
- 实现了支持最多五十人的聊天室，随时可以加入新用户
- 用户可以拥有唯一的用户名
- 可以使用 @用户名 的格式私聊任意一名加入聊天室的用户
- 正常直接发送信息，所有人可见
- 可以直接读入一整行内容

*提醒：由于我的应用程序是控制台应用，若你想运行测试，请务必先退出你电脑上所有的文献阅读器和翻译器（甚至是WPS），否则会导致BUG，在最后**实验中遇到的问题及思考**小节中有解释原因。

实验环境

使用CLion IDE，使用的包如下：

```
#include <cstdio>
#include <winsock2.h>
#include <windows.h>
#include <iostream>
#include <thread>

#pragma comment(lib, "ws2_32.lib") //加载 ws2_32.dll
#pragma comment(lib, "winmm.lib")
```

注意，在CLion中编译网络程序代码需要你在CMakeLists中添加以下内容，否则不能正常编译：

```

set(CMAKE_EXE_LINKER_FLAGS "-static")
set(CMAKE_CXX_STANDARD 14)
link_libraries(ws2_32 wsock32)

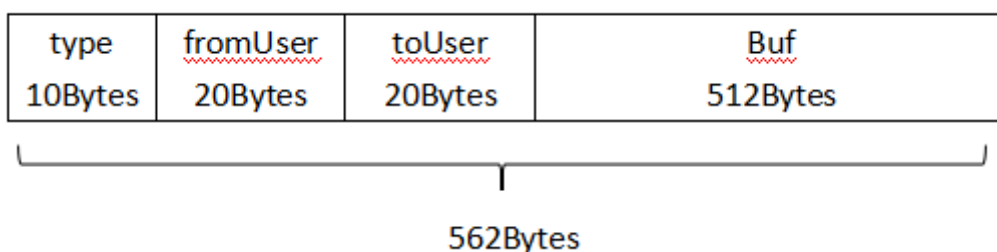
add_executable(server server.cpp)
add_executable(client client.cpp)

```

协议设计

报文类型与结构的设计

在我设计的聊天程序中，消息报文是 `char[]` 字符串类型，定义消息报文的具体结构如下所示：



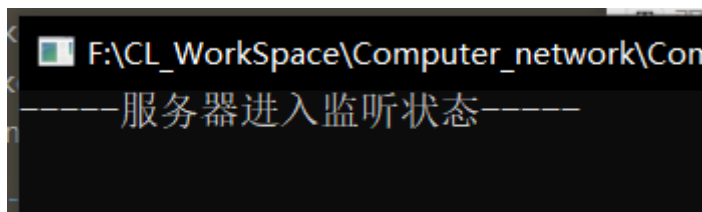
- **type**字段：指示消息的类型，支持三种基础消息类型：“SYS”（系统消息）、“PRI”（私聊消息）、“PUB”（公共消息）；
- **fromUser**字段：消息发送方的用户名；
- **toUser**字段：消息接收方的用户名，只用当消息类型为“PRI”时该字段有效；
- **Buf**字段：需要发送的具体消息

我们将**type**、**fromUser**、**toUser**这三个字段看成一个整体为**head**字段，即附加在需要传输的消息前的头消息。综上所述，最终报文的长度为562Bytes。

语义及时序

在这里针对程序运行时的输入输出语义及时序作出解释，暨介绍程序如何在命令行运行：

1. 在打包文件中解压出server.exe和client.exe，分别为服务器端和客户端可执行文件；
2. 首先运行server.exe，会弹出 服务器进入监听状态 的提示消息，看到此消息后说明服务器端已经准备好接收客户端SOCKET的连接了；



3. 随后运行client.exe，需要我们输入服务器的IP地址（若在本机测试输入 127.0.0.1 即可），随后SOCKET成功与服务器建立连接，此时需要我们继续输入我们的用户名（正如我在报文结构上提到的，这里要求你的用户名小于20字节，并且服务器中没有用户使用相同的用户名），输入正确用户名后客户端成功加入聊天室；

```
F:\CL_WorkSpace\Computer_network\Computer_Network\Lab1_3\cmake-build
[NOT CONNECTED] 请输入聊天服务器的地址
127. 0. 0. 1
[CONNECTED] SOCKET连接成功
[CONNECTED] 请输入您的用户名
WNZ
[CONNECTED] 已连接到服务器
[2022/10/20 20:36:23.149] 发送成功
[系统通知] 新用户了加入聊天室, 用户名为: WNZ
```

```
F:\CL_WorkSpace\Computer_network\Computer_Network\Lab1_3\cmake-build
-----服务器进入监听状态-----
新客户端连接成功, 它的编号是: 0 用户名为: WNZ IP地址: 127. 0. 0. 1
```

4. 随后加入聊天室的用户消息都会在服务器中以日志形式记录;
5. 当用户输入 '@', 客户端会进一步提示用户输入想要私聊发送消息的接收方, 用户手动输入接收方用户名后, 再输入私聊的内容, 最后只有双方用户能看见私聊消息, 其他加入聊天室的用户则接收不到;
6. 当用户直接输入 "quit", 客户端结束服务, 直接退出, 服务器会将用户退出的消息广播给其他用户。

在这里我们并不详细地描述程序如何进行用户交互, 在作者介绍完实现原理、代码框架后会展示更完全的聊天程序的交互界面功能。

语法

- 正常输入一行消息后按回车发送消息;
- 需要使用私聊功能时, 先输入 '@' 按下回车, 再输入对方用户的用户名, 按回车, 最后输入你想发送的消息;
- 输入 "quit" 为特殊命令, 直接关闭客户端。

程序的实现

技术框架

首先, 本次实验是基于流式SOCKET搭建的多人聊天室, 所以底层使用的是TCP协议, 在我们编程时也必须使用 `SOCK_STREAM` 来声明我们的SOCKET;

其次, 为了让客户端可以自由地发送、接收消息, 显然单线程不能完成我们的工作, 所以我们在客户端新开两个线程分别负责处理消息的接收和发送;

再者, 在服务器端, 我们同样使用多线程的技术来管理与用户连接的SOCKET, 并实现线程函数用于实现客户端消息转发、广播的功能, 我们为每一个连接到服务器的用户新开一个线程管理其SOCKET。

启动并建立连接过程的代码

由于在启动并建立连接阶段, Server端和Client端代码绝大多数都是可以复用的, 所以我们以Server端代码为例进行介绍。

报文结合的宏定义

正如我们在 **报文类型与结构的设计** 小节中设计的报文格式, 我们在C++代码中对报文结构的长度进行宏定义, 方便后续编程。

```
#define BUF_SIZE 512
#define NAME_SIZE 20
#define TYPE_SIZE 10
#define HEAD_SIZE 2*NAME_SIZE+TYPE_SIZE
#define MESSAGE_SIZE HEAD_SIZE+BUF_SIZE
```

端口号和IP地址的宏定义

由于Server端需要使用 `bind` 函数绑定IP地址和端口号，为了简化实现，我们直接给出端口号和IP地址宏定义，方便后续编程使用。

```
#define PORT 7878
#define ADDRsrv "127.0.0.1"
```

一些全局变量的声明

```
//Server端
static map<string,int>userNameMap;//用于记录用户名和对应的用户编号
static SOCKET sockConnects[MAXUSER];//用于维护所有与Client端连接的SOCKET的一个数组

//Client端
static char userName[NAME_SIZE];//用户名
```

初始化SOCKET库

Server端和Client启动的第一步，都是初始化加载SOCKET库。首先，使用 `MAKEDWORD(2, 2)` 来确定使用的版本为2.2版本，如果 `WSAStartup` 返回值不是0，说明加载DLL失败，程序直接退出。

```
WSADATA wsaData;
if (WSAStartup(MAKEDWORD(2, 2), &wsaData) != 0) {
    //加载失败
    cout << "加载DLL失败" << endl;
    return -1;
}
```

创建SOCKET

第二步程序创建socket，指定地址类型为 `AF_INET`，并指定流式套接字，最后的参数 `0` 则是告诉系统自己选择合适的协议。（`PF_INET`和`AF_INET`应该没有区别，因为头文件里对`PF_INET`的宏定义就是直接用`AF_INET`）

```
SOCKET sockServer = socket(AF_INET, SOCK_STREAM, 0);
```

将本地地址绑定到SOCKET

第三步只需要服务器端实现，将服务器地址 `ADDRsrv` 和端口号 `PORT` 使用 `bind` 函数绑定到前一步创建好的服务器SOCKET上。*注意，`sin_family` 必须和上一步创建socket时指定的地址类型一致，否则不能正常运行。

```

SOCKADDR_IN addrSrv;
addrSrv.sin_family = AF_INET;
addrSrv.sin_port = htons(PORT);
addrSrv.sin_addr.s_un.s_addr = inet_addr(ADDRSRV);
bind(sockServer, (SOCKADDR *) &addrSrv, sizeof(SOCKADDR));

```

进入监听

第四步，服务器进入监听状态，监听客户端的连接。如果返回值是0说明服务器进入监听状态；如果不是，说明监听失败，程序直接结束。

```

if (listen(sockServer, 10) == 0) {
    cout << "服务器进入监听状态" << endl;
} else {
    cout << "监听失败" << endl;
    return -1;
}

```

用户端和客户端之间进行连接

第五步，首先由用户端使用 `connect` 函数发起连接请求。

```

//Client代码
if (connect(sockClient, (SOCKADDR *) &addrSrv, sizeof(SOCKADDR)) != 0) {
    //TCP连接失败
    cout << "连接失败" << endl;
    return -1;
}

```

在服务器端，首先，需要找到维护与客户端连接SOCKET的数组一个空闲位置的编号（在这里使用 `SOCKET_ERROR`代表空闲态），作为本次连接SOCKET的编号；接下来客户端调用 `accept` 函数与客户端进行连接，将 `accept` 函数返回的与客户端连接的SOCKET赋值给 `sockConnects[index]`。

```

while(1){
    //find the index
    int index=0;
    for(;index<MAXUSER;index++){
        if (sockConnects[index] == SOCKET_ERROR)
            break;
    }
    if(index==MAXUSER){
        cout<<"目前服务器已经到底最大连接人数"<<endl;
        continue;
    }

    SOCKADDR_IN addrClient;
    int lenAddr = sizeof(SOCKADDR);
    sockConnects[index] =accept(sockServer, (SOCKADDR *) &addrClient, &lenAddr));
    if (sockConnects[index] == SOCKET_ERROR) {
        cout << "连接失败" << endl;
        sockConnects[index]= SOCKET_ERROR;
        continue;
    }
}

```

```

    }
    //Recv the user name
    ...
    ...
}

```

随后，用户端必须给出唯一的用户名，如果服务器中已经有重复的用户名存在，则拒绝本次用户名的创建，要求用户重新输入；

```

//Client代码
//将用户名给客户端保存
while(1){
    cout << "[CONNECTED] 请输入您的用户名" << endl;
    cin >> userName;
    send(sockClient,userName,NAME_SIZE,0);

    char t[10];
    recv(sockClient,t,10,0);
    if(strcmp(t,"TRUE")==0){
        break;
    }
    cout<<"[CONNECTED] 该用户名已存在，请重新输入"<<endl;
}
cout<<"[CONNECTED] 已连接到服务器"<<endl;

```

在服务器端，使用 `map` 数据结构进行管理用户端和唯一的用户名之间的映射关系，如果是新的用户名则选择接收，并使用系统类型的报文，广播给全部用户通知他们有新的用户加入；如果是重复的用户名，则要求客户端重新 `send` 一个用户名，直到满足要求。

```

//Recv the user name
while(1) {
    char name[NAME_SIZE];
    string t("TRUE");
    recv(sockConnects[index], name, NAME_SIZE, 0);
    if(userNameMap.find(string(name))==userNameMap.end()){
        //新的用户名，选择接收
        userNameMap.insert(pair<string, int>(string(name), index));
        send(sockConnects[index],t.c_str(),10,0);
        cout << "新客户连接成功,它的编号是:" << index << "用户名为:" << name << " IP
地址:" << inet_ntoa(addrClient.sin_addr) << endl;

        //使用系统类型消息“SYS”，广播给全部用户有新用户加入
        char mess[MESSAGE_SIZE]="SYS";
        char b[BUF_SIZE]="新用户了加入聊天室,用户名为:";
        strcat(b,name);
        for(int j=0;j<BUF_SIZE;j++)
            mess[j+HEAD_SIZE]=b[j];
        transfer2AllUser(mess);
        break;
    }
    else{
        t="FALSE";
        send(sockConnects[index],t.c_str(),10,0);
    }
}

```

```
}
```

以上操作实现了客户端和服务端端的初始化已经建立SOCKET连接的过程，实际上服务器端的主线程并不会只执行一次连接。如果你细心观察，就能发现，服务器端主线程中等待连接、接收连接是在一个 `while(true)` 循环中的，所以这个线程总是在等待新的客户端与服务器连接。

既然建立好了连接，接下来就要处理如何进行消息的传输了，对于客户端来说，需要处理的是如何进行消息的接收和发送；对于服务器端，需要处理的是如果将一个用户的消息转发到指定用户或广播给全部用户，正如我们在技术框架所述的，我们使用多线程技术来实现，下面先给出线程的启动代码，再描述线程函数细节。

```
//Server端，为每一个连接SOCKET启动新的线程处理信息转发
CloseHandle(CreateThread(NULL, NULL, handlerTransfer,
(LPVOID)&sockConnects[index], 0, NULL));

//Client端，为接收和发送消息分别启动新的线程
HANDLE hthread[2];
hthread[0] = CreateThread(NULL, 0, handlerRec, (LPVOID) &sockClient, 0, NULL);
hthread[1] = CreateThread(NULL, 0, handlerSend, (LPVOID) &sockClient, 0, NULL);
WaitForMultipleObjects(2, hthread, TRUE, INFINITE);
CloseHandle(hthread[0]);
CloseHandle(hthread[1]);
```

Client端的线程函数

消息发送

用户端处理消息发送的线程函数如下所示，线程函数参数是一个 `SOCKET*` 类型，即与服务器建立连接的SOCKET的指针。由于我们希望允许用户无限次、在任意时刻发送消息，所以用一个 `while(true)` 循环监听用户是否有新的输入。

接下来接收到用户在命令行的输入后会先进行判断本次消息类型是“PRI”或“PUB”，然后按照不同消息类型的要求组织报文结构，最后通过 `send` 将报文发送给服务器，让服务器负责消息的转发。

```
DWORD WINAPI handlerSend(LPVOID lparam) {
    SOCKET *socket = (SOCKET *) lparam;
    //报文结构
    char type[TYPE_SIZE];
    char fromUser[NAME_SIZE], toUser[NAME_SIZE];
    char buf[BUF_SIZE];
    char message[MESSAGE_SIZE];
    char head[HEAD_SIZE];
    memset(head, 0, HEAD_SIZE);
    memset(type, 0, TYPE_SIZE);
    memset(fromUser, 0, NAME_SIZE);
    memset(toUser, 0, NAME_SIZE);
    memset(message, 0, MESSAGE_SIZE);
    memset(buf, 0, BUF_SIZE);

    strcpy(fromUser, userName);

    while (1) {
        //读入一整行，所以用cin.getline()
        cin.getline(buf, 512);
```

```

//设计head头信息
//如果输入为@, 则是向另一个用户私发信息
if (strcmp(buf, "@") == 0) {
    cout << "请输入你想私聊的用户名:";
    cin >> toUser;
    cin.ignore();
    //TYPE是"PRI"是私聊信息
    strcpy(type, "PRI");
    strcpy(head, type);
    for (int j = 0; j < NAME_SIZE; j++)
        head[j + TYPE_SIZE] = fromUser[j];
    for (int j = 0; j < NAME_SIZE; j++)
        head[j + TYPE_SIZE + NAME_SIZE] = toUser[j];

    cin.getline(buf, 512);
} else {
    //TYPE为"PUB"是群发信息, 不需要toUser
    strcpy(type, "PUB");
    strcpy(head, type);
    for (int j = 0; j < NAME_SIZE; j++)
        head[j + TYPE_SIZE] = fromUser[j];
}

//将HEAD和BUF内容移动到MESSAGE的对应位置
for (int j = 0; j < HEAD_SIZE; j++)
    message[j] = head[j];
for (int j = 0; j < BUF_SIZE; j++)
    message[j + HEAD_SIZE] = buf[j];

//发送至服务器
send(*socket, message, MESSAGE_SIZE, 0);
//如果输入quit, 直接退出聊天室
if (strcmp(buf, "quit") == 0) {
    printSysTime();
    cout << "[System Info]";
    cout << "您已退出聊天室" << endl;
    getchar();
    return 0;
}

printSysTime();
cout << "发送成功" << endl;

memset(head, 0, HEAD_SIZE);
memset(type, 0, TYPE_SIZE);
memset(toUser, 0, NAME_SIZE);
memset(message, 0, MESSAGE_SIZE);
memset(buf, 0, BUF_SIZE);
}
}

```


消息接收

消息接收线程函数与消息发送线程很类似，不同点在于，当 `recv` 函数接收到消息后，消息接收线程必须先将消息报文中的 `type` 字段卸下来，判断本次消息的类型。

如果是 `type` 为 `SYS`，说明是系统发出的消息，如果具体消息内容为 `exit`，说明系统将该用户移出了聊天室，用户线程直接结束；如果 `type` 为 `PRI`，说明接收到的消息时某个用户向你发出的私聊信息，在消息的显示时需要加上（私聊）来提醒用户；如果 `type` 为 `PUB`，说明是公开的消息。

```
DWORD WINAPI handlerRec(LPVOID lparam) {
    SOCKET *socket = (SOCKET *) lparam;

    char type[TYPE_SIZE];
    char fromUser[NAME_SIZE];
    char buf[BUF_SIZE];
    char message[MESSAGE_SIZE];
    memset(type, 0, TYPE_SIZE);
    memset(fromUser, 0, NAME_SIZE);
    memset(message, 0, MESSAGE_SIZE);
    memset(buf, 0, BUF_SIZE);

    while (1) {
        recv(*socket, message, MESSAGE_SIZE, 0);
        for (int j = 0; j < TYPE_SIZE; j++)
            type[j] = message[j];

        for (int j = 0; j < BUF_SIZE; j++)
            buf[j] = message[j + HEAD_SIZE];
        if (strcmp(type, "SYS") == 0) {
            //系统发出的消息,结束对服务器服务,直接结束线程
            if (strcmp(buf, "exit") == 0) {
                return 0;
            }
            printf("[系统通知]%s\n", buf);
            continue;
        }
        //设置user名字字段
        for (int j=0;j<NAME_SIZE;j++)
            fromUser[j]=message[j+TYPE_SIZE];

        if (strcmp(buf, "quit") == 0) {
            printSysTime();
            printf("用户 [%s] 已经退出聊天室\n", fromUser);
            continue;
        }

        if (strcmp(type, "PRI") == 0) {
            //私聊信息显示
            printSysTime();
            printf("用户 [%s] (私聊): %s\n", fromUser, buf);
        }
        else {
            printSysTime();
            printf("用户 [%s]: %s\n", fromUser, buf);
        }
    }
}
```

```

        memset(message, 0, MESSAGE_SIZE);
        memset(type, 0, TYPE_SIZE);
        memset(fromUser, 0, NAME_SIZE);
        memset(buf, 0, BUF_SIZE);
    }
}

```

服务器端的线程函数

正如我上述说的，我希望对连接到服务器的每一个用户新开一个线程，线程函数负责处理用户消息的转发。所以函数参数自然就是服务器端和用户连接的 `SOCKET` 指针。由于服务器需要对报文做一些解析，所以我们同样在线程中声明了报文结构。

接下来需要判断该线程管理的 `SOCKET` 的编号，因为我们现在函数参数只是一个 `SOCKET` 指针，我们并不知道在该 `SOCKET` 的编号，所以我们通过遍历一次 `sockConnects[]` 找到 `SOCKET` 的编号。

做完以上准备工作后，进入 `while(true)` 循环为用户提供服务，因为服务器需要转发用户消息，第一步自然就是 `recv` 接收用户传到服务器的消息。首先需要判断用户传输上来的消息是否为 `quit`，如果接收到用户退出的信息则将改用户退出聊天室的消息广播给全部用户，然后向用户发送一条 `sys` 类型的消息，通知他的接收消息线程结束服务，然后在客户端维护的 `userNameMap` 中移除该用户的用户名，关闭连接的 `SOCKET`，最后结束线程，释放资源。

如果是正常的信息，服务器则将通过 `type` 字段来判断消息的类型，如果是 "PRI"，则是一条私聊信息，客户端在 `userNameMap` 中找到与 `toUser` 对应的编号，然后调用 `transfe2OneUser(toUser.index)` 将这条信息转发给该用户，其他用户都无法接收该消息，实现私聊功能；

如果是 "PUB"，说明是聊天室里的公开消息，服务器直接调用 `transfer2AllUser()` 将这条消息广播给全部用户，实现聊天室的群聊功能。

```

DWORD WINAPI handlerTransfer(LPVOID lparam) {
    SOCKET *socket = (SOCKET *) lparam;

    char type[TYPE_SIZE];
    char fromUser[NAME_SIZE], toUser[NAME_SIZE];
    char buf[BUF_SIZE];
    char message[MESSAGE_SIZE];
    char head[HEAD_SIZE];
    memset(head, 0, HEAD_SIZE);
    memset(type, 0, TYPE_SIZE);
    memset(fromUser, 0, NAME_SIZE);
    memset(toUser, 0, NAME_SIZE);
    memset(message, 0, MESSAGE_SIZE);
    memset(buf, 0, BUF_SIZE);

    //找编号，其实后面发现可以直接把函数参数改成编号，只是不方便改了，时间不够
    int i = 0;
    for(int j=0; j<MAXUSER; j++, i++)
        if(&sockConnects[j]==socket)
            break;

    while (1) {
        //接收用户发出的消息
        recv(*socket, message, MESSAGE_SIZE, 0);
        //报文结构拆解

```

```

for(int j=0;j<BUF_SIZE;j++)
    buf[j]=message[HEAD_SIZE+j];
for(int j=0;j<TYPE_SIZE;j++)
    type[j]=message[j];
for(int j=0;j<NAME_SIZE;j++){
    fromUser[j]=message[j+TYPE_SIZE];
    toUser[j]=message[j+TYPE_SIZE+NAME_SIZE];
}

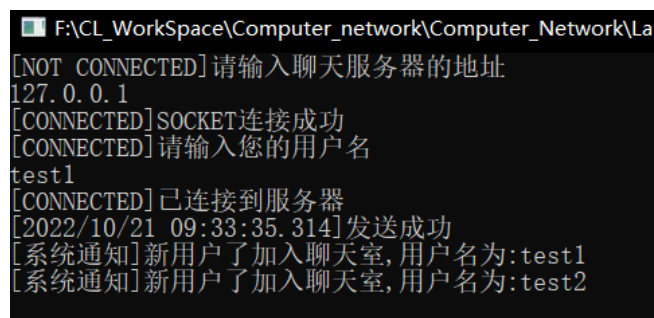
if (strcmp(buf, "quit") == 0) {
    printSysTime();
    printf("用户[%s]已经退出聊天室\n", fromUser);
    transfer2AllUser(message);
    //通知客户端下线
    char temMessage[MESSAGE_SIZE]="SYS";
    char temBuf[BUF_SIZE]="exit";
    for(int j=0;j<MESSAGE_SIZE;j++)
        temMessage[j+HEAD_SIZE]=temBuf[j];

    userNameMap.erase(string(fromUser));
    send(sockConnects[i],temMessage,MESSAGE_SIZE,0);
    //释放SOCKET
    closesocket(sockConnects[i]);
    return 0;
}
if (strlen(buf) > 0) {
    if (strcmp(type, "PRI") == 0) {
        printSysTime();
        printf("用户[%s]私聊[%s]:%s\n", fromUser, toUser, buf);
        //向特定用户转发私聊消息
        transfer2OneUser(userNameMap[string(toUser)],message);
    } else {
        printSysTime();
        printf("[用户%s]:%s\n", fromUser, buf);
        transfer2AllUser(message);
    }
}
memset(message, 0, BUF_SIZE);
}
}

```

程序展示

1. 启动服务器与客户端，客户端需要输入服务器IP地址，然后输入用户名；



```

F:\CL_WorkSpace\Computer_network\Computer_Network\La
[NOT CONNECTED] 请输入聊天服务器的地址
127.0.0.1
[CONNECTED] SOCKET连接成功
[CONNECTED] 请输入您的用户名
test1
[CONNECTED] 已连接到服务器
[2022/10/21 09:33:35.314] 发送成功
[系统通知] 新用户了加入聊天室, 用户名为:test1
[系统通知] 新用户了加入聊天室, 用户名为:test2

```

下图展示的是输入重复用户名的情况，服务器拒绝该用户名并要求用户重新输入；

```
F:\CL_WorkSpace\T 双人聊天程序设计与实现.md - Typora
-----服务器进入监听状态-----
新客户端连接成功,它的编号是:0 用户名为:test1 IP地址:127.0.0.1
新客户端连接成功,它的编号是:1 用户名为:test2 IP地址:127.0.0.1
[NOT CONNECTED]请输入聊天服务器的地址
127.0.0.1
[CONNECTED]SOCKET连接成功
[CONNECTED]请输入您的用户名
test1
[CONNECTED]该用户名已存在. 请重新输入
[CONNECTED]请输入您的用户名
test2
[CONNECTED]已连接到服务器
[2022/10/21 09:34:25.310]发送成功
[系统通知]新用户加入了聊天室,用户名为:test2
```

2. 连接完成后用户可以在聊天室中自由发言;

```
[系统通知]新用户加入了聊天室,用户名为:test1
[系统通知]新用户加入了聊天室,用户名为:test2
你好
[2022/10/21 09:37:23.230]发送成功
[2022/10/21 09:37:23.232]用户[test1]:你好
[2022/10/21 09:37:31.563]用户[test2]:收到
[2022/10/21 09:37:38.788]用户[test2]:你也好 我今天很困

[2022/10/21 09:34:25.310]发送成功
[系统通知]新用户加入了聊天室,用户名为:test2
[2022/10/21 09:37:23.232]用户[test1]:你好
收到
[2022/10/21 09:37:31.562]发送成功
[2022/10/21 09:37:31.563]用户[test2]:收到
你也好 我今天很困
[2022/10/21 09:37:38.786]发送成功
[2022/10/21 09:37:38.789]用户[test2]:你也好 我今天很困
```

服务器端有一个日志记录聊天室中用户发送的全部消息;

```
F:\CL_WorkSpace\Computer_network\Computer_Network\Lab1_3\cmake-build-debug
-----服务器进入监听状态-----
新客户端连接成功,它的编号是:0 用户名为:test1 IP地址:127.0.0.1
新客户端连接成功,它的编号是:1 用户名为:test2 IP地址:127.0.0.1
[2022/10/21 09:37:23.230][用户test1]:你好
[2022/10/21 09:37:31.562][用户test2]:收到
[2022/10/21 09:37:38.786][用户test2]:你也好 我今天很困
```

3. 随后让第三个用户加入聊天室;

```
F:\CL_WorkSpace\Computer_network\Computer_Network\Lab1_3\cmake-build-debug
-----服务器进入监听状态-----
新客户端连接成功,它的编号是:0 用户名为:test1 IP地址:127.0.0.1
新客户端连接成功,它的编号是:1 用户名为:test2 IP地址:127.0.0.1
[2022/10/21 09:37:23.230][用户test1]:你好
[2022/10/21 09:37:31.562][用户test2]:收到
[2022/10/21 09:37:38.786][用户test2]:你也好 我今天很困
新客户端连接成功,它的编号是:2 用户名为:WNZ IP地址:127.0.0.1
你也好 我今天很困
[2022/10/21 09:37:38.786]发送成功
[2022/10/21 09:37:38.789]用户[test2]:你也好 我今天很困
[系统通知]新用户加入了聊天室,用户名为:WNZ
```

4. 用户 test1 和用户 test2 之间测试私聊功能;

```
[系统通知]新用户了加入聊天室,用户名为:WNZ
@
请输入你想私聊的用户名:test2
私聊的测试 ABCD
[2022/10/21 09:48:50.773]发送成功
[2022/10/21 09:49:06.596]用户[test2](私聊):收到私聊消息

[2022/10/21 09:48:17.159]用户[test2]:你也好 今天很困
[系统通知]新用户了加入聊天室,用户名为:WNZ
[2022/10/21 09:48:50.775]用户[test1](私聊):私聊的测试 ABCD
@
请输入你想私聊的用户名:test1
收到私聊消息
[2022/10/21 09:49:06.593]发送成功

F:\CL_WorkSpace\Computer_network\Computer_Network\Lab1_3
[NOT CONNECTED]请输入聊天服务器的地址
127.0.0.1
[CONNECTED]SOCKET连接成功
[CONNECTED]请输入您的用户名
WNZ
[CONNECTED]已连接到服务器
[2022/10/21 09:48:26.773]发送成功
[系统通知]新用户了加入聊天室,用户名为:WNZ

[2022/10/21 09:48:10.578][用户test2]:收到
[2022/10/21 09:48:17.157][用户test2]:你也好 今天很困
新客户连接成功,它的编号是:2用户名为:WNZ IP地址:127.0.0.1
[2022/10/21 09:48:50.773]用户[test1]私聊[test2]:私聊的测试 ABCD
[2022/10/21 09:49:06.593]用户[test2]私聊[test1]:收到私聊消息
```

可见,私聊功能实现了只在 test1 和 test2 之间的私聊通信, WNZ 用户接收不到任何信息;

5. 用户断开连接, 结束测试。

```
私聊的测试 ABCD
[2022/10/21 09:48:50.773]发送成功
[2022/10/21 09:49:06.596]用户[test2](私聊):收到私聊消息
[2022/10/21 09:52:16.311]用户[test2]已经退出聊天室
[2022/10/21 09:52:36.252]用户[WNZ]已经退出聊天室

[2022/10/21 09:49:06.593]用户[test2]私聊[test1]:收到私聊
[2022/10/21 09:52:16.309]用户[test2]已经退出聊天室
[2022/10/21 09:52:36.249]用户[WNZ]已经退出聊天室
[2022/10/21 09:53:24.977]用户[test1]已经退出聊天室
```

实验中遇到的问题及思考

- 一开始我对服务器的设计是,只使用一个线程来处理所有用户的消息转发,即在这个线程中重复遍历所有连接好的SOCKET判断他有没有recv成功,如果成功则负责转发消息,如果不成功则继续遍历直到有一个SOCKET收到消息。

之所以有这种想法是参考网上一种WSAEvent的做法,但我发现并不可行,因为recv会让线程进入阻塞态,即recv一定会等待到用户端SOCKET发送一个消息,如此我的程序就变成了按用户序号一人轮流发一条信息了,显然是不合理的,所以后面采取多线程,为每一个连接的SOCKET新建一个线程管理。

- `cin.getline()` 和 `cin>>` 混用导致的输入问题。其实这是一个老问题了，以前C++编程就遇到过，但是这里一时疏忽大意忘记了，因为 `cin>>` 不会读入 `\n`，导致 `cin>>` 输入完后字符缓存区还有一个 `\n`，而 `cin.getline()` 遇到 `\n` 就会结束读入，所以必须在 `cin>>` 后使用 `cin.ignore()` 清除这个回车符。
- 报文格式的错误问题。一开始我直接用了 `strcat` 拼接字符串，但是这种方法不能把字符串拼接成正确的报文结构，他会找到字符串结束的位置拼接下一个字符串，所以最后还是采取最笨的方法循环赋值，可能用 `string` 会方便一点。
- 多线程程序的全局变量的问题，注意要用 `static` 声明全局变量，不然每一个线程会复制一份自己的全局变量，如果发生全局变量上的重新赋值，就会出现数据不一致的问题。
- 史上最坑BUG。控制台程序一拖动就自动退出，原因是你的电脑开启了文献阅读器或翻译软件，鼠标左键按下拖动放开左键后，软件会自动加一个 `ctrl+c` 帮你复制文本，但是这个刚好是控制台的退出命令，所以懂了吧。