计算机网络作业一

使用 socket 编程，实现多人聊天室

姓名：林语盈

学号：2012174

班级：计算机卓越班

目录

一、功能概述

实现了两个版本的聊天程序。版本一实现了多人聊天室功能包括加入聊天室、发送消息、退出聊天室，使用阻塞发送和接收，只能在客户端输入内容之后才能刷新当前获取的消息。版本二在版本一的基础上实现了实时接收消息的多人聊天室功能。

两个版本均分为 server 端与 client 端，server 端进行聊天室的总体控制，client 作为参与聊天的成员进入聊天室，并进行消息发送和接收。

二、版本一协议设计

对于 server 与 client 全部消息的接收和发送均为字符串类型，本程序中定于缓冲区最大为 char[1024].

1.客户端

对于客户端，所有接收消息的操作都会调用 recv_until_q 函数，该函数将不断阻塞接收来自 server 的信息并进行显示，直至收到字符串“q”则退出循环。

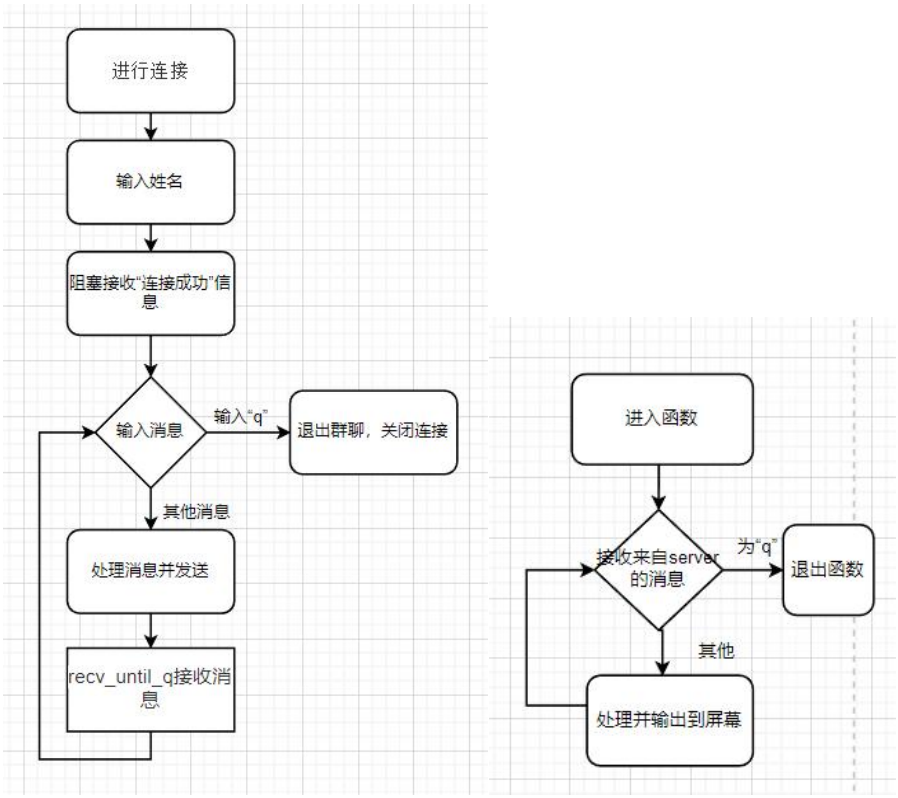
- 首先，进行连接，输入客户端自己的 name，并在本地保存，设置该姓名最长为 char[20]。
- 然后，客户端将阻塞接收来自 server 的“成功连接”信息与“q”，然后即进行聊天环节。
- 接下来，client 不断重复如下操作：

  1.从键盘读消息，消息中可以有空格、中英文、符号、数字，以回车('\n')结束，设定该消息最长为 char[900]。若消息为“q”，代表退出聊天，直接向 server 发送“q”，然后跳出循环并结束连接；若为其他，继续至 2 步。

  2.在本地读取当前时间、姓名，处理好要发送的数据，如“小红(2022-10-19 23:11:11): 你好。\n”。

  3.向 server 端发送此条消息

  4.调用 recv_until_q 函数接收消息。



(a)版本一 client 端整体流程　　(b) recv_until_q 函数流程

图 1 版本一 client 端流程

2.服务端

对于服务端，采用多线程流式（为了保证数据的可靠性，采用 tcp 协议），对于每个客户端的 socket，创建新的线程进行处理。

- 首先，连接成功后会向客户端发送“成功连接”的信息和“q”。
- 接下来，server 会不断阻塞接收来自 client 的消息，每收到一条消息就将其转发给其他所有在线的 client，并向当前 client 发送“q”。这里其收到的信息“q”时，代表该客户端退出群聊，将退出循环；对于其他的消息，不会进行任何处理，而是直接转发。

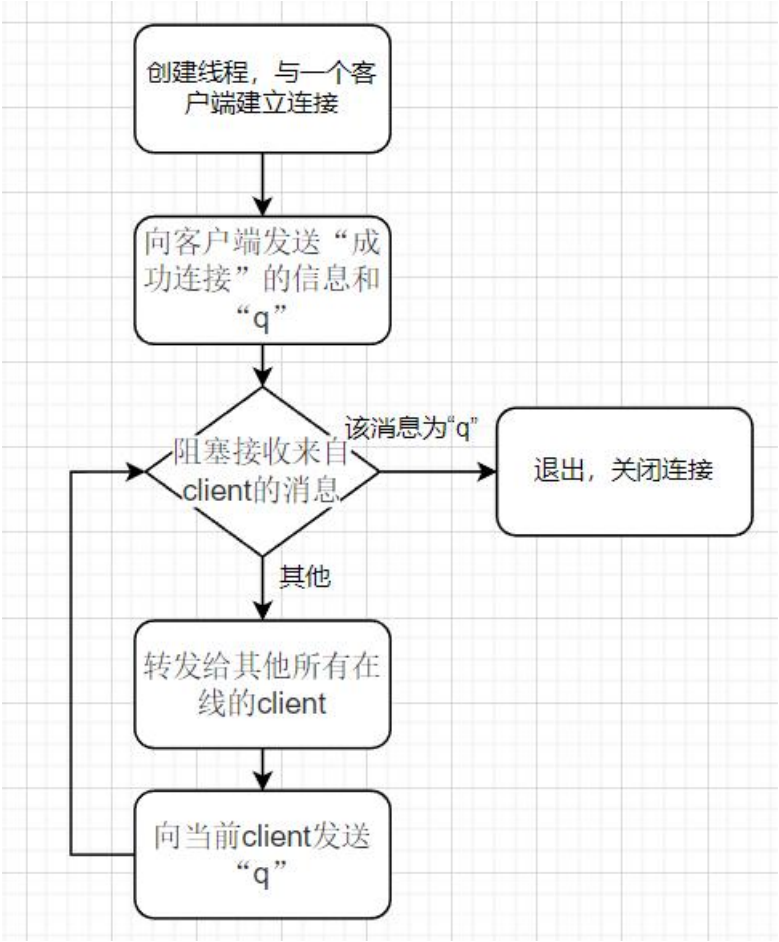


图 2 版本一 server 端流程

三、版本二协议设计

大部分与版本一类似，主要差别在于全部使用非阻塞接收发送，对于客户端，获取键盘中断并进行发送，故实现了实时的消息群发。

对于 server 与 client 全部消息的接收和发送，本程序中定于缓冲区最大为 char[1024].

1.客户端

对于客户端

- 首先，需要输入客户端自己的 name，并在本地保存，设置该姓名最长为 char[20]。
- 然后，客户端将非阻塞接收来自 server 的“成功连接”信息，然后即进行聊天环节。
- 接下来，client 不断重复非阻塞接收消息并显示，直至获取到键盘中断。获取到键盘中断时，从键盘读消息，消息中可以有空格、中英文、符号、数字，以回车('\n')结束，设定该消息最长为 char[900]。若消息为“q”，代表退出聊天，直接向 server 发送“q”，然后跳出循环并结束连接；若为其他，在本地读取当前时间、姓名，处理好要发送的数据，如“小红(2022-10-19 23:11:11): 你好。\n”，并向 server 端发送此条消息

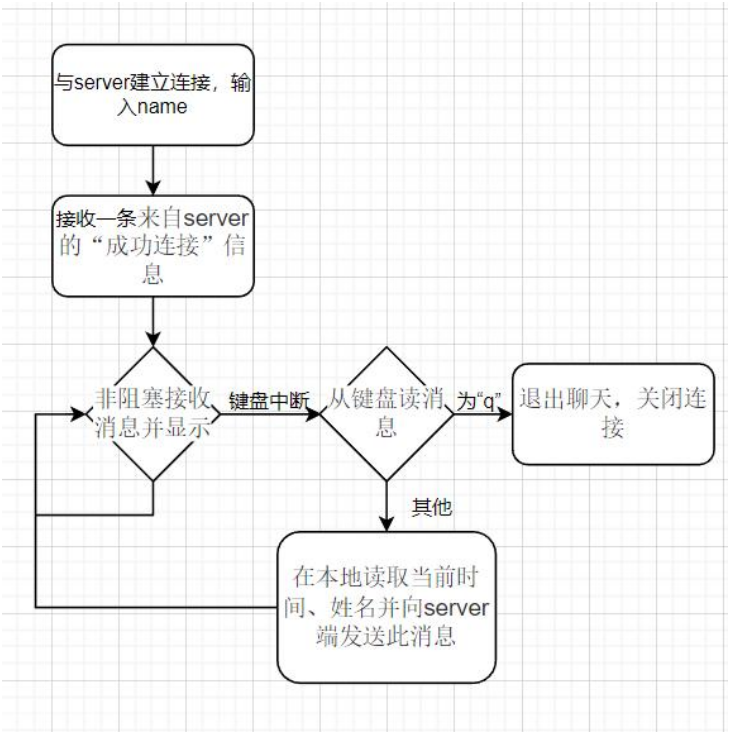


图 3 版本二 客户端流程

2.服务端

对于服务端，采用多线程流式（为了保证数据的可靠性，采用 tcp 协议），对于每个客户端的 socket，创建新的线程进行处理。

- 首先，连接成功后会向客户端发送“成功连接”的信息。
- 接下来，server 会不断非阻塞接收来自 client 的消息，每收到一条消息就将其转发给其他所有在线的 client。这里其收到的信息“q”时，代表该客户端退出群聊，将退出循环并断开连接；对于其他的消息，不会进行任何处理，而是直接转发。

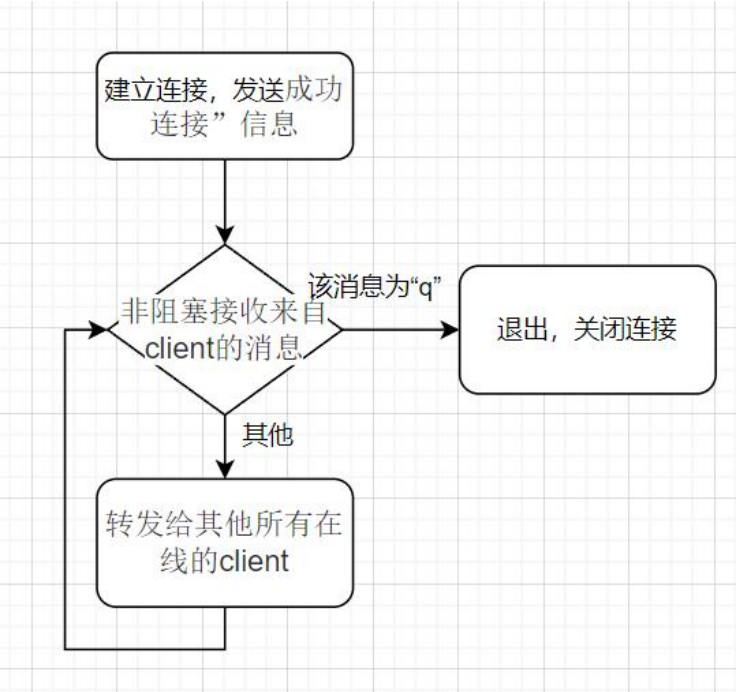


图 4 版本二 服务端流程

四、版本一核心代码解释

程序由 Client.cpp 与 server.cpp 两个文件组成，与第二节所述流程相同。

Client.cpp 结构大致如下：

```
...#define MAXBUFSIZE 1024 //接收或发送的 data 的最大长度为 sizeof(char[1024])
char name[20];
int sendMessage(SOCKET clientSocket) {
    //从键盘读入消息，并给 server 发送一条处理好的（带有当前姓名、时间、消息）信息。
    //返回 1 代表当前用户退出群聊，返回 0 代表当前用户正常发送完成本条消息。
...}

int recv_until_q(SOCKET clientSocket) {
    //不断接收来自 server 的消息并输出，直至接收到"q"，退出。
...}

int main()
{
...
    err = WSAStartup(versionRequired, &wsaData);//协议库的版本信息
...//创建 socket 并连接
    SOCKET clientSocket = socket(AF_INET, SOCK_STREAM, 0);
...
    SOCKADDR_IN clientsock_in;
    inet_pton(AF_INET, "127.0.0.1", &clientsock_in.sin_addr.S_un.S_addr);
...
    //开始连接
    int con_err = connect(clientSocket, (SOCKADDR*)&clientsock_in, sizeof(SOCKADDR));
    //接收连接确认消息
    recv_until_q(clientSocket);
    while (1) {
        //读入并发送消息
        int send_err = sendMessage(clientSocket);
        if (send_err == 1) break;
        //不断接收消息直至收到 q
        recv_until_q(clientSocket);
    }
    //关闭套接字
```

```
    closesocket(clientSocket);
    //关闭服务
    WSACleanup();
    return 0;
}
```

server.cpp 结构大致如下：

```
...
#define MAX_P_NUM 10
std::vector<SOCKET> ClientSockets;//用于记录全部 client 的 SOCKET
int ClientSocket_num = 0;
#define MAXBUFSIZE 1024 //接收或发送的 data 的最大长度为 sizeof(char[1024])

DWORD WINAPI handlerRequest(LPVOID lparam)//线程处理函数
{
    ...
//加入 socket
    if (ClientSocket_num > MAX_P_NUM) {
        printf("群聊人数已达上限\n");
        return 0;
    }
    SOCKET socket = (SOCKET)(LPVOID)lparam;
    ClientSockets.push_back(socket);
    ...

    //发送初始确认信息 与“q”
    ...
    send(socket, sendBuf, MAXBUFSIZE, 0);
    ....
    send(socket, sendBuf, MAXBUFSIZE, 0);


    while (1) {
        //接收
        ...
        recv(socket, receiveBuf, MAXBUFSIZE, 0);
        ...

        //关闭并断开连接
        if (strcmp(receiveBuf, "q") == 0) {
            ...
            closesocket(socket);
            ...
            break;
        }

        //发送此条 message 给所有 client
        ....
        for (...)
        {
            send(*it, receiveBuf, MAXBUFSIZE, 0);
            ..
        }

        //发送 q 使当前 client 停止接收
        memset(sendBuf, 0, MAXBUFSIZE * sizeof(char));
        strcpy_s(sendBuf, MAXBUFSIZE, "q");
        send(socket, sendBuf, MAXBUFSIZE, 0);
    }
    return 0;
}

int main()
{
    //创建套接字，socket 前的一些检查工作，包括服务的启动
    ...
    err = WSAStartup(myVersionRequest, &wsaData);
    ...
    SOCKET serSocket = socket(AF_INET, SOCK_STREAM, 0);//创建了可识别套接字
    //需要绑定的参数，主要是本地的 socket 的一些信息。
```

```
    SOCKADDR_IN addr;
    addr.sin_family = AF_INET;
    addr.sin_addr.S_un.S_addr = htonl(INADDR_ANY);//ip 地址
    addr.sin_port = htons(6000);//绑定端口

    bind(serSocket, (SOCKADDR*)&addr, sizeof(SOCKADDR));//绑定完成
    listen(serSocket, MAX_P_NUM);//其中第二个参数代表能够接收的最多的连接数

    SOCKADDR_IN clientsocket;
    int len = sizeof(SOCKADDR);
    while (1)
    {
        //第二次握手，通过 accept 来接受对方的套接字的信息
        SOCKET serConn = accept(serSocket, (SOCKADDR*)&clientsocket, &len);
        HANDLE hThread = CreateThread(NULL, NULL, handlerRequest, LPVOID(serConn), 0, NULL);
        CloseHandle(hThread);
    }

    closesocket(serSocket);//关闭
    WSACleanup();//释放资源的操作
    return 0;
}
```

五、版本二核心代码解释

程序由 Client.cpp 与 server.cpp 两个文件组成，与第三节所述流程相同。

Client.cpp 结构大致如下：

```
int main()
{
     ...
     err = WSAStartup(versionRequired, &wsaData);//协议库的版本信息
     ...
    //定义 socket 并指定为非阻塞通信
    SOCKET clientSocket = socket(AF_INET, SOCK_STREAM, 0);
    u_long mode = 1;
     ioctlsocket(clientSocket, FIONBIO, &mode);
    //端口号，ip 地址等信息
    SOCKADDR_IN clientsock_in;
    inet_pton(AF_INET, "127.0.0.1",&clientsock_in.sin_addr.S_un.S_addr);
    clientsock_in.sin_family = AF_INET;
    clientsock_in.sin_port = htons(6000);
  ...
    scanf_s("%s", &name,50);
    //开始连接
    int con_err = connect(clientSocket, (SOCKADDR*)&clientsock_in, sizeof(SOCKADDR));
    //接收连接确认消息
    recv_one(clientSocket);

    while (1) {
        //读入并发送消息
        if (_kbhit()) {//获取键盘中断
            int send_err = sendMessage(clientSocket);
            if (send_err == 1) break;
        }
        //不断接收消息
       ...
        recv(clientSocket, receiveBuf, MAXBUFSIZE, 0);
        printf("%s", receiveBuf);
    }
    //关闭套接字
    closesocket(clientSocket);
    //关闭服务
     WSACleanup();
     ...
}
int sendMessage(SOCKET clientSocket) {
```

```
    //从键盘读入消息，并给 server 发送一条处理好的（带有当前姓名、时间、消息）信息。
    //返回 1 代表当前用户退出群聊，返回 0 代表当前用户正常发送完成本条消息。
     ...
}
int recv_one(SOCKET clientSocket) {
    //接收来自 server 的一条消息并退出。
    ...
}
```

server.cpp 结构大致如下：

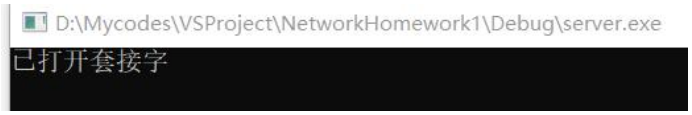
```
...
#define MAX_P_NUM 10
std::vector<SOCKET> ClientSockets;//用于记录全部 client 的 SOCKET
int ClientSocket_num = 0;
#define MAXBUFSIZE 1024 //接收或发送的 data 的最大长度为 sizeof(char[1024])

DWORD WINAPI handlerRequest(LPVOID lparam)//线程处理函数
{
    int ClientSocket_id = ClientSocket_num;
    ClientSocket_num++;
     //判断群聊人数是否已达上限
    if (ClientSocket_num > MAX_P_NUM) {
        printf("群聊人数已达上限\n");
        return 0;
     }
     //将 socket 加入 ClientSockets
    SOCKET socket = (SOCKET)(LPVOID)lparam;
    ClientSockets.push_back(socket);
    std::vector<SOCKET>::iterator socket_it = ClientSockets.end() - 1;

    //发送初始确认信息
    ...

    while (1) {
        //接收
        ...
        while (recv(socket, receiveBuf, MAXBUFSIZE, 0) <= 0);
        printf("收到来自%d: %s", ClientSocket_id, receiveBuf);

        //关闭并断开连接
        if (strcmp(receiveBuf, "q") == 0) {
            ...
            closesocket(socket);
            ...
            break;
        }
             //发送此条 message 给所有 client
        for (...)
        {
            send(*it, receiveBuf, MAXBUFSIZE, 0);
            ...
        }
     }
...
}

int main()
{
```

```
    //创建套接字，socket 前的一些检查工作，包括服务的启动
    ...
    err = WSAStartup(myVersionRequest, &wsaData);
    ...
    SOCKET serSocket = socket(AF_INET, SOCK_STREAM, 0);//创建了可识别套接字
    //需要绑定的参数，主要是本地的 socket 的一些信息。
    SOCKADDR_IN addr;
    addr.sin_family = AF_INET;
    addr.sin_addr.S_un.S_addr = htonl(INADDR_ANY);//ip 地址
    addr.sin_port = htons(6000);//绑定端口

    bind(serSocket, (SOCKADDR*)&addr, sizeof(SOCKADDR));//绑定完成
    listen(serSocket, MAX_P_NUM);//其中第二个参数代表能够接收的最多的连接数
...
    while (1)
    {
        //第二次握手，通过 accept 来接受对方的套接字的信息
        SOCKET serConn = accept(serSocket, (SOCKADDR*)&clientsocket, &len);
        //指定为非阻塞通信
        u_long mode = 1;
        ioctlsocket(serConn, FIONBIO, &mode);
       //创建线程并传入 socket
        HANDLE hThread = CreateThread(NULL, NULL, handlerRequest, LPVOID(serConn), 0, NULL);
        CloseHandle(hThread);
    }

    closesocket(serSocket);//关闭
    WSACleanup();//释放资源的操作
    return 0;
}
```

六、程序界面展示及运行说明

两个版本界面类似，仅以版本二进行展示。

首先，运行 server 端，输出 server 的 socket 的打开情况：



加入一个 client,并输入 name 并回车，服务端显示成功连接到 0 号 client，客户端显示成功连接到 server：

此时，client 正在实时接收其他聊天对象发送的消息，并且可随时从键盘输入消息，以回车结束（中间可包括空格、中英文、符号、数字，可以发送空消息）。注：从键盘输入消息时不会实时接收消息，该消息会在输入完成后被立即接收。

可以在 client 输出看到用户的消息以及发送时间，并且服务端可以显示输入与输出的消息。输入“q”表示退出群聊，一个人退出群聊时 server 会进行记录，不会影响其他用户的聊天。新用户可以随时进入群聊。

单人：

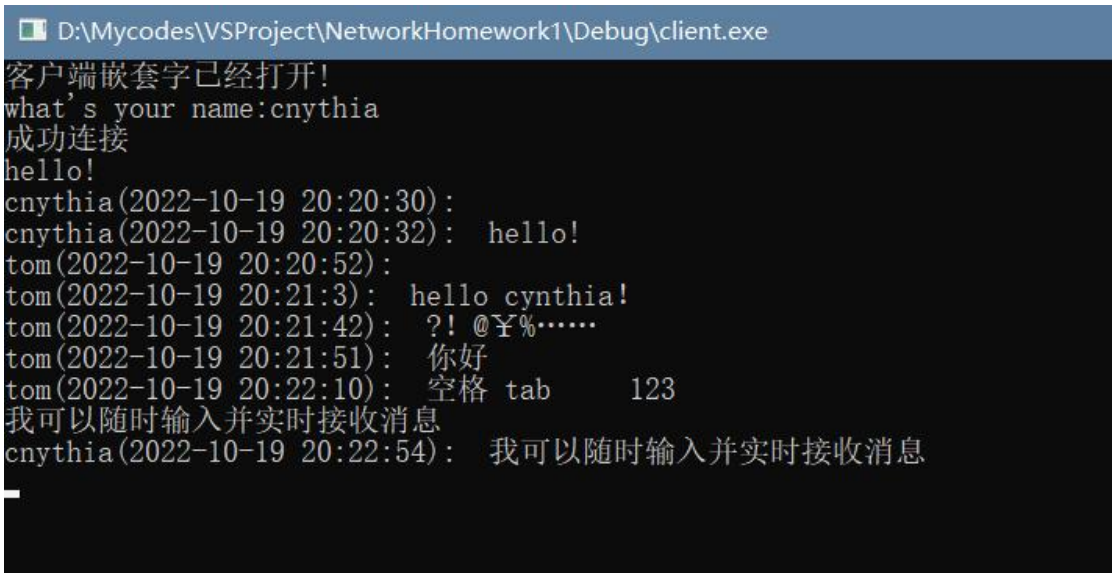
Server 端：

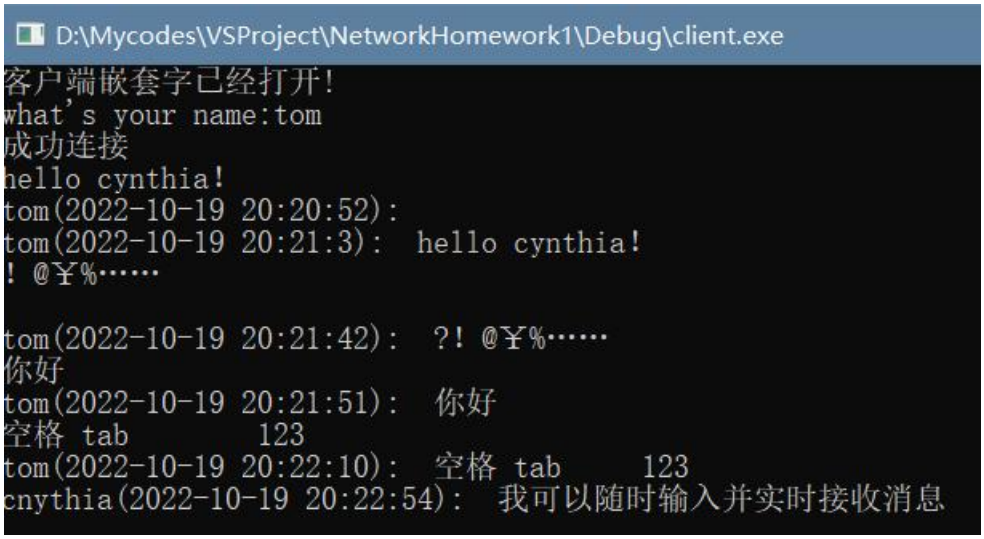
```
收到来自0: 小明123qwe(2022-10-19 20:13:32):  esdas 你好 234      d*……%) “: 》? ! @#¥%……&*
发送到0:  小明123qwe(2022-10-19 20:13:32):  esdas 你好 234      d*……%) “: 》? ! @#¥%……&*
```

Client 端：

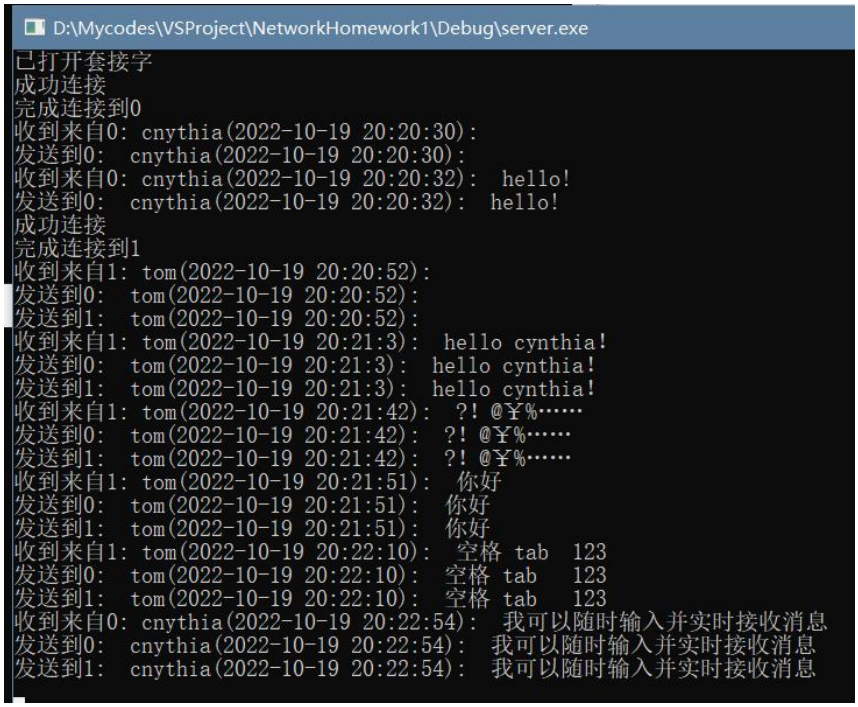
```
esdas 你好 234  d*……%) “: 》? ! @#¥%……&*
小明123qwe(2022-10-19 20:13:32):  esdas 你好 234        d*……%) “: 》? ! @#¥%……&*
```

多人聊天示例：



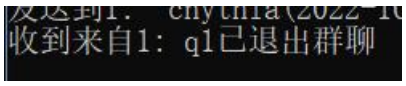


Server 记录：



Cynthia 用户输入“q”，退出群聊：

Server 端提示：



此时其他人可以继续正常聊天。

七、实验过程中遇到的问题及分析

实验中遇到的问题非常多，不再一一列举。比如应当注意发送和接收的缓冲区必须初始化、服务端的 socket 池应当注意赋值与释放的依赖关系与顺序、阻塞与非阻塞通信等。一个比较值得注意的问题是，若使用阻塞通信和接收，每次在等待输入时都无法实时获取其他用户发送的信息，这导致用户体验较差，于是由版本一产生了版本二，具体分析见上。

八、总结

本次实验实现了多人实时聊天程序，熟悉了 socket 编程，对整体的流程有了更深的理解，也为接下来的学习打下了基础。