
南开大学

Nankai University

# 南　　开　　大　　学

## 计　算　机　学　院

**计算机网络实验报告**

---

**编程作业 1—聊天小程序**

---

姓名：高祎珂

年级：2020 级

专业：计算机科学与技术

2022 年 10 月 22 日

摘要

**关键字：TCP/IP 客户端 服务器端**

# 目录

# 一、 简述

本次实验使用 TCP/IP 协议，和流式 socket 使用了简单的聊天程序，聊天信息带有时间标签。

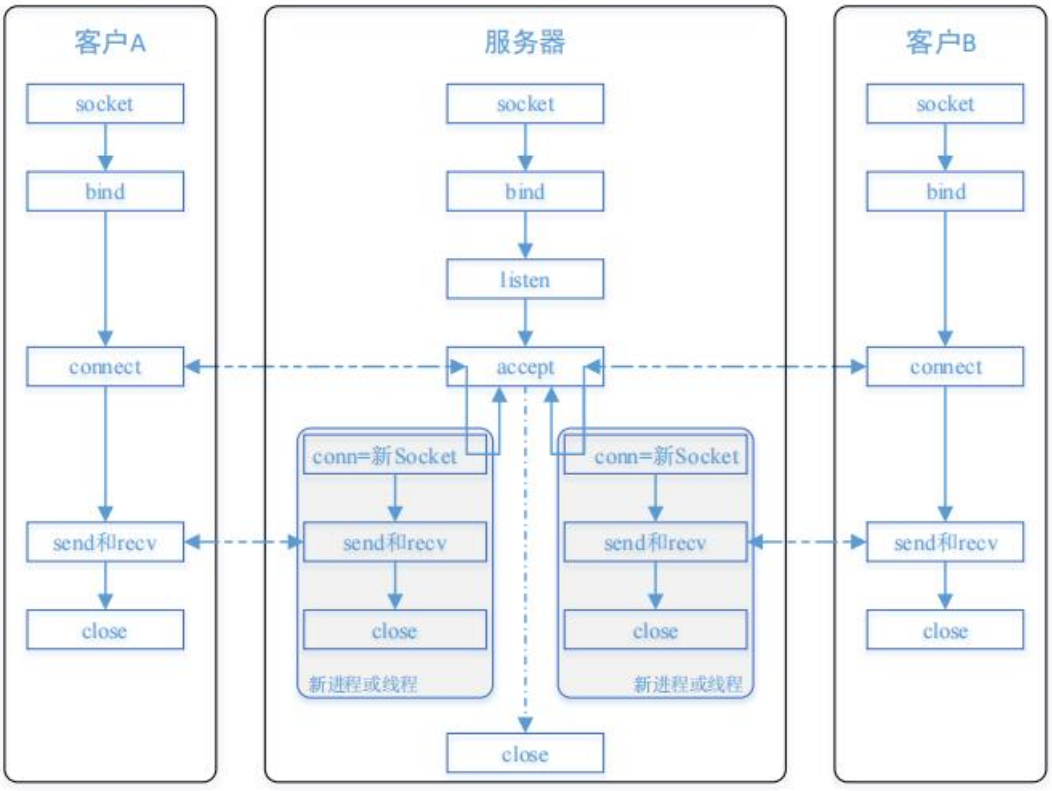
实验环境

- Windows 10
- IDE Visual Studio2022

代码的 github 链接这里

# 二、 协议设计

整个设计基于 TCP 协议，设计了客户端和服务器端，进行信息交互，客户端 client 发送消息给 server 服务器端，再有服务器 server 进行转发，将消息发送给其他客户端。实验设计模块图如下所示




消息类型又分为群聊消息和私聊消息，默认情况下为群聊消息，服务器端把消息广播给所有用户，如果按照特定格式输入，会被认为是私聊消息，服务器端只把该消息转发给指定的人。

# 三、 代码展示

## (一) 服务器端

为了与多个客户端进行交互，实现群聊模式，这里使用了多线程

服务器端主代码

```
SOCKET AcceptSocket = accept(ListenSocket, (SOCKADDR*)&ClientAddr, &len);
        if (AcceptSocket == INVALID_SOCKET) {
            cout << "接收客户端Socket失败" << endl;
            closesocket(ListenSocket);
            WSACleanup();
            return 1;
        }
        else {
            // 创建线程，并且调用函数传入与client通讯的套接字
            HANDLE hThread = CreateThread(NULL, 0, lpStartFun, (LPVOID)
                AcceptSocket, 0, NULL);
            CloseHandle(hThread); // 关闭对线程的引用
        }
```

线程执行函数为

线程函数

```
DWORD WINAPI lpStartFun(LPVOID lparam) {
    //传进来的是接收到返回的socket
    SOCKET ClientSocket = (SOCKET)(LPVOID)lparam;
    client_map[ClientSocket] = 1;       //该client在线
    //客户端输入呢，当成消息传过来，获取用户名
    char user_name[20];
    //strcpy_s(user_name, to_string(ClientSocket).data());
    //名字传给客户端
    //send(ClientSocket, user_name, 20, 0);
    recv(ClientSocket, user_name, 20, 0);
    name_map[ClientSocket] = string(user_name);

    SYSTEMTIME time;
    GetLocalTime(&time);
    cout << time.wHour << ":" << time.wMinute << ":" << time.wSecond << "  ";
    cout<< "用户: " << user_name << "加入聊天" << endl;

    int isquit = 1;//用户端是否退出
    int istoall = 1;//消息是否群发
    char recvMessage[MESSAGE_LEN];
    char sendMessage[MESSAGE_LEN];
    char speMessage[MESSAGE_LEN];//私聊消息

    int isRecv= recv(ClientSocket, recvMessage, MESSAGE_LEN, 0);
    while (isRecv != SOCKET_ERROR && isquit != 0) {
```

```
        //消息处理函数
    }
```

这里消息执行函数，要判断客户端传递出来的信息，是私聊信息还是群聊信息，私聊信息格式为“用户名 < 信息”, 群聊信息直接发送信息即可，服务器端会把此信息广播到其他所有客户端。这里为了记录用户名和客户端是否开始，使用了两个 map 结构, 以此来记录映射关系

两个映射 map

```
map<SOCKET, int> client_map;//map存储客户端状态，如果value=1，说明clinet在线
map<SOCKET, string> name_map;
```

首先根据以下代码判断是否为群聊消息，是则 istoall 值为 0，否则 istoall 值为 1

消息类型判断

```
for (int i = 0; i < MESSAGE_LEN; i++) {
                if (recvMessage[i] == '<') { //发现"<"即说明为私聊      消息类型是  目的用户<消息
                    istoall = 0;
                    break;
                }
            }
```

私聊消息

```
if (istoall == 0) {

                char talk_to[20] = "";          //私聊的用户名
                int j;
                for (j = 0; j < MESSAGE_LEN; j++) {
                    if (recvMessage[j] == '<') {
                        talk_to[j] = '\0';
                        for (int z = j + 1; z < MESSAGE_LEN; z++) {
                            speMessage[z - j - 1] = recvMessage[z];
                            //if (recvMessage[z] == '\0')break;
                        }
                        strcat_s(speMessage, "( from ");
                        strcat_s(speMessage, user_name);
                        strcat_s(speMessage, ")\0");
                        break;
                    }
                    else {
                        talk_to[j] = recvMessage[j];
                    }
                }
                cout << "私聊to" << talk_to << endl;
                //设置向客户端发送的消息  私聊消息  此时服务器端要记录
                for (auto it : client_map) {
                    //如果想要自定义用户名，这里也许再需要一个map，存储socket和用户名的映射
```

```
                    //cout << it.second<<endl;

                    if (name_map[it.first] == string(talk_to) && it.second == 1) {
                        //使用目的用户socket, 进行消息发送
                        cout << "string(name_map[it.first])" << name_map[it.first] << endl;
                        cout << "string(talk_to)" << string(talk_to) << endl;
                        int isSend = send(it.first, speMessage, MESSAGE_LEN, 0);
                        if (isSend == SOCKET_ERROR) {
                            cout << "发送失败" << endl;
                            it.second = 0;
                        }

                        else {
                            SYSTEMTIME time;
                            GetLocalTime(&time);
                            cout << time.wHour << ":" << time.wMinute << ":" << time.wSecond << "  [INFO]";
                            cout << user_name << "消息发送成功"  << endl;

                        }
                        break;
                    }
                    //私聊对象未上线
                    if (to_string(it.first) == string(talk_to) && it.second == 0) {
                        string disconnect = "该用户未上线";
                        send(ClientSocket, disconnect.data(), MESSAGE_LEN, 0);
                    }
                }
            }
```

群聊消息

```
strcpy_s(sendMessage, user_name); // data函数直接转换为char*
                strcat_s(sendMessage, ": ");
                strcat_s(sendMessage, recvMessage);

                //把消息转发给除了发消息的人
                for (auto it : client_map) {
                    if (it.first != ClientSocket && it.second == 1) {
                        int isSend = send(it.first, sendMessage, MESSAGE_LEN, 0);
                        if (isSend == SOCKET_ERROR)
                        {

```

```
                cout << "发送失败" << endl;
            }
            else {
                SYSTEMTIME time;
                GetLocalTime(&time);
                cout << time.wHour << ":" << time.wMinute << ":"
                    << time.wSecond << "  [INFO]";
                cout << user_name << "消息发送成功" << endl;

            }
        }
    }
```

## (二) 客户端

客户端建立好 socket，并且向服务器发送连接请求，用户名是由客户端输入，并把用户名传递给服务器端，由服务器存到 map 映射中，便于后续利用 socket 识别用户名

群聊消息

```
int isError=connect(ClientSocket, (struct sockaddr*)&ServerAddr, sizeof(
    ServerAddr));
if (isError == SOCKET_ERROR) {
    cout << "连接服务端失败" << endl;
    closesocket(ClientSocket);
    WSACleanup();
    return 1;
}
else {
    cout << "连接服务端成功" << endl;
}
//先接收send过来的用户名
//输入用户名，并且传给客户端
cout << "请输入用户名: ";
cin.getline(user_name, 20);
send(ClientSocket, user_name, 20, 0);

// 打印进入聊天的标志
cout << "              Welcome    User    " << user_name << endl;
cout << "*************************************************" << endl;
cout << "             Use quit command to quit" << endl;
cout << "*************************************************" << endl;

// 创建两个线程，一个接收线程，一个发送线程
HANDLE hThread[2];
hThread[0] = CreateThread(NULL, 0, Recv, (LPVOID)&ClientSocket, 0, NULL);
hThread[1] = CreateThread(NULL, 0, Send, (LPVOID)&ClientSocket, 0, NULL);
```

接收线程

```
SOCKET* recvSocket = (SOCKET*)lparam_socket;
    while (true)
    {
        char RecvMessage[MESSAGE_LEN];
        int isRecv = recv(*recvSocket, RecvMessage, MESSAGE_LEN, 0);
        if (isRecv > 0 && isquit == 1) {
            SYSTEMTIME time;
            GetLocalTime(&time);
            cout << time.wHour << ":" << time.wMinute << ":" << time.wSecond
                << "  ";
            cout << RecvMessage << endl;
        }
        else {
            //接受不到消息，关闭socket
            closesocket(*recvSocket);
            return 0;
        }
    }
```

发送线程

```
SOCKET* sendSocket = (SOCKET*)lparam_socket;
    while (true)
    {
        cout << "请输入你要发送的消息" << endl;
        char sendMessage[MESSAGE_LEN];
        cin.getline(sendMessage, MESSAGE_LEN);//getline可以识别空格，遇到换行
            自动结束
        if (string(sendMessage) == "quit") {
            isquit = 0;
            closesocket(*sendSocket);  //关闭socket，此时服务器端 isRecv就会
                出现错误，进而关闭通信
            cout << "退出群聊" << endl;
            WSACleanup();//调用WSACleanup函数来解除与Socket库的绑定并且释放
                Socket库所占用的系统资源
            return 0;
        }
        else {
            int isSend = send(*sendSocket, sendMessage, MESSAGE_LEN, 0);
            if (isSend == SOCKET_ERROR) {
                cout << "发送信息失败:" << WSAGetLastError() << endl;
                closesocket(*sendSocket);
                WSACleanup();//调用WSACleanup函数来解除与Socket库的绑定并且释
                    放Socket库所占用的系统资源
                return 0;
            }
        }
```

```
23
24      }
```

# 四、 结果演示


(a) 服务器端初始化 (b) 客户端初始化

图 1: 初始化

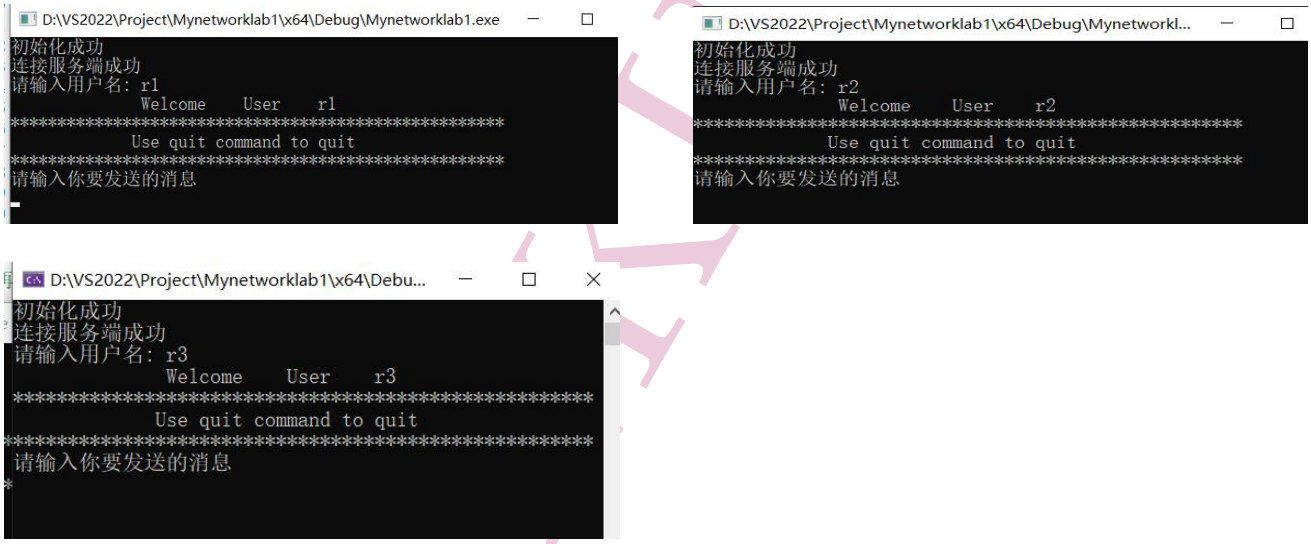
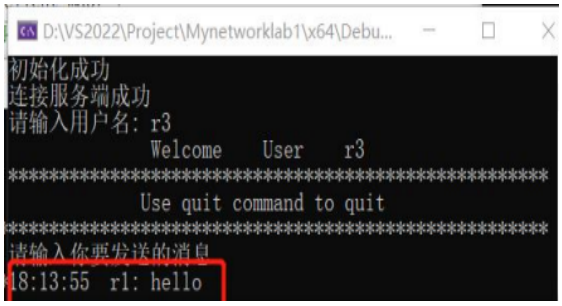


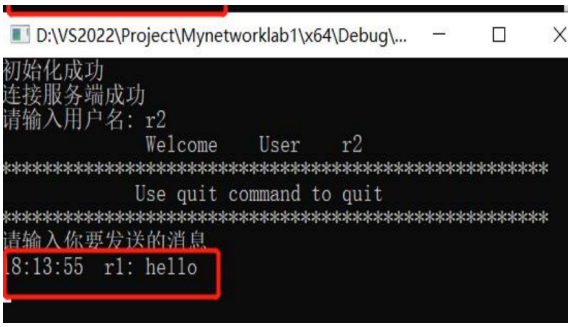

图2：输入用户名


图3：输入用户名后服务器端显示


(a) 用户r1发送hello



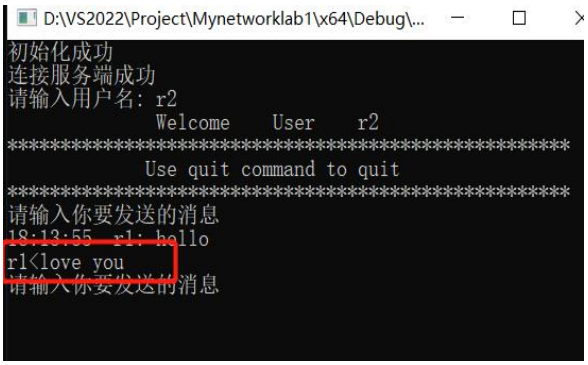

(b)用户r3接收到消息



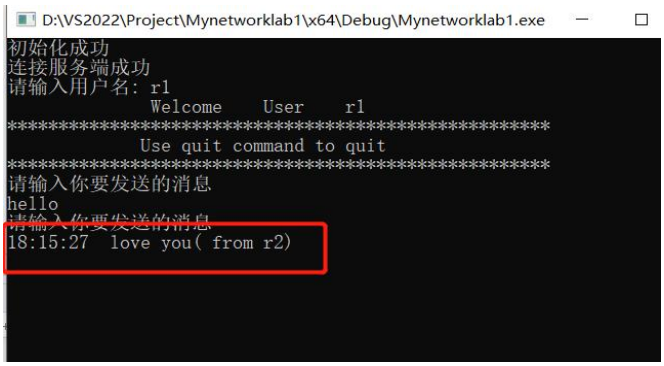

(c)用户r2接收到消息


(d)服务器端显示

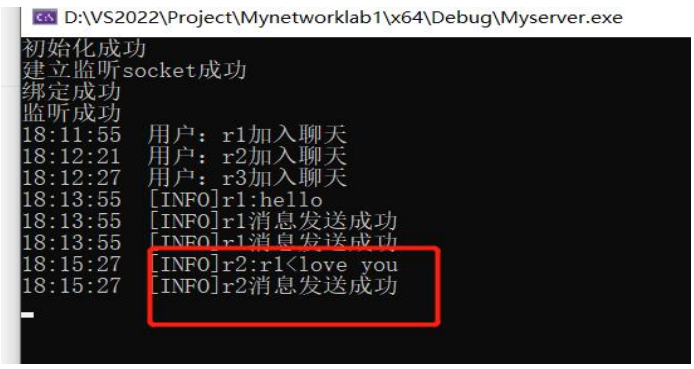
图4：用户r1发送群聊消息




(a)r2向r1发送私聊消息




(b)r1接收到消息




(c)服务器端显示

图5：私聊消息显示

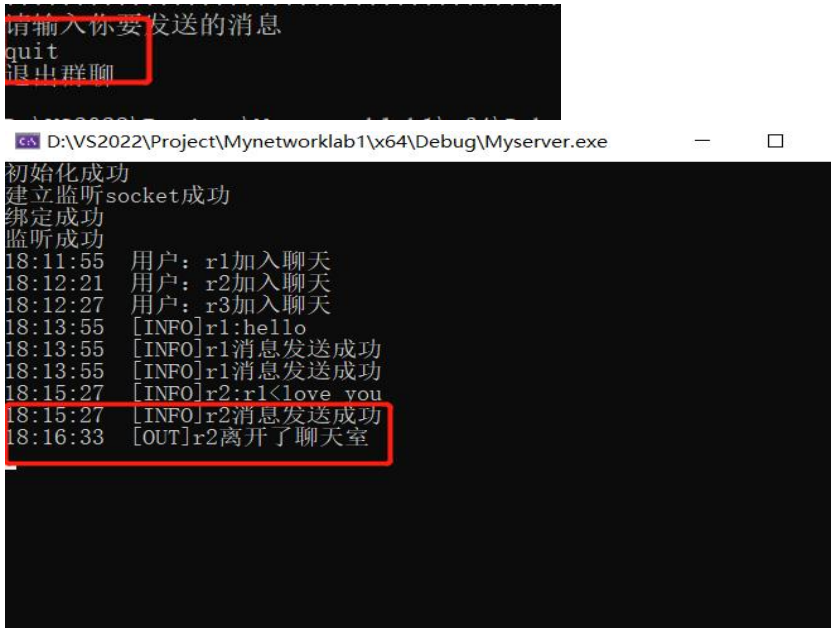

请输入你要发送的消息
quit
退出群聊
D:\VS2022\Project\Mynetworklab1\x64\Debug\Myserver.exe
初始化成功
建立监听socket成功
绑定成功
监听成功
18:11:55 用户：r1加入聊天
18:12:21 用户：r2加入聊天
18:12:27 用户：r3加入聊天
18:13:55 [INFO]r1:hello
18:13:55 [INFO]r1消息发送成功
18:13:55 [INFO]r1消息发送成功
18:15:27 [INFO]r2:r1<love you
18:15:27 [INFO]r2消息发送成功
18:16:33 [OUT]r2离开了聊天室

图6：退出客户端