南開大學
Nankai University

网络空间安全学院

计算机网络实验报告

实验一：基于 TCP 的聊天程序设计

姓名：魏伯繁

学号：2011395

专业：信息安全

2022 年 10 月 22 日

目录

1 总体实验框架 **2**
1.1 实验要求 . . . 2
1.2 功能实现 . . . 2
1.3 实验环境 . . . 2
1.4 协议设计 . . . 3

2 流程概述 **4**

3 代码展示 **6**
3.1 设计思想 . . . 6
3.2 服务端 . . . 6
3.2.1 初始化 dll . . . 6
3.2.2 初始化 socket . . . 6
3.2.3 绑定套接字地址 . . . 7
3.2.4 进入监听状态 . . . 8
3.2.5 接收请求 . . . 8
3.2.6 基本判断函数 . . . 9
3.2.7 线程实现部分 . . . 10
3.3 客户端 . . . 14
3.3.1 初始化 . . . 14
3.3.2 接收线程 . . . 15
3.3.3 发送线程 . . . 16

4 效果展示 **17**

5 总结 **18**

1 总体实验框架

1.1 实验要求

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。

对聊天程序进行设计。给出模块划分说明、模块的功能和模块的流程图。

在 Windows 系统下，利用 C/C++ 对设计的程序进行实现。程序界面可以采用命令行方式，但需要给出使用方法。编写程序时，只能使用基本的 Socket 函数，不允许使用对 socket 封装后的类或架构。

可以进行的提高练习：其他协议设计及拓展功能（如群聊、多线程等），视协议功能性、复杂

1.2 功能实现

多线程聊天室-支持可自定义的人数进行聊天（通过修改源代码进行修改即可）

服务端可以自由决定使用的 ip 地址以及端口号（通过命令行输入的形式使用）同理，客户端也可以通过输入不同的想要连接的 ip 地址以及端口后进入不同的聊天室。

在聊天过程中，服务端负责日志记录并将其实时打印在命令行上，日志内容主要包括：聊天室的人数变化情况、用户的聊天内容、更改个人资料等操作。

在客户端，用户端可以通过服务端的转发得知聊天室内其他用户发送的聊天内容，同时也可以自主选择退出聊天室以及更改个人资料等操作。

为丰富客户的使用功能，实现了私聊功能，只需要发送格式为 sendto <receivername> <content> 的聊天内容即可完成私聊转发，转发内容对服务端可见，但是在客户端只有 receiver 可以看到转发的内容。

1.3 实验环境

本实验代码编写采用 VS2019 作为编译器，实验的本地环境为 WIN10 系统

具体引用的头文件、使用的动态链接库以及错误处理机制的使用如下使用的头文件以及动态链接库

```
1
2  int main() {
3      #define _WINSOCK_DEPRECATED_NO_WARNINGS
4      #define _CRT_SECURE_NO_WARNINGS 1
5      #include<iostream>
6      #include<string>
7      #include<WinSock2.h>
8      #include<WS2tcpip.h>
9      #include<thread>
10     #include<map>
11     using namespace std;
```

```
12      #pragma comment(lib,"ws2_32.lib")
13  }
```

1.4 协议设计

首先，每个用户发送的消息不能超过使用 100 个字节表示，多出的部分会自动被舍弃。其次，每个用户都拥有无限次更改自己用户名的权利，其初始值我们使用函数

to_string(cs).data()

对其进行随机初始化。

在进行信息传递转发时，由于服务端中保存了两个 map，分别保存 socket 与其用户名间的映射，所以在 server 段进行日志输出以及消息转发时会将用户的昵称以及消息发出的时间进行一并展示，其在服务端具体格式为：

在 xxxx 年 xx 月 xx 日 xx 时 xx 分 xx 秒 xxx 说：xxxxx

消息将以这种形式进行传递

与此同时，在客户端，为了准确的实现更改昵称以及退出等功能的实现，在客户端定义了关键字，即一旦用户按照输入输入了指定字符，就会触发相应的操作：
我们规定，输入形如：

”changename”+” ”+username

将被系统自动转译为更改用户名的操作。
同理，输入:

“quit”

将被系统自动转译为退出操作，客户端会主动退出，服务端也会接收到消息并且不再给对应的客户端转发消息，该位置将自动空缺。

根据定义的的协议设计在服务端的输出为：

图 1.1: 日志输出效果图

综上所述：协议规定，传递的信息将通过 char* 进行保存，一次性传输的信息不能超过 100 字节（包括关键字及关键字描述信息）。服务端在拿到客户端的信息后，会首先检查信息头是否包括关键字，如果包含关键字，将进行特殊处理。如果不包含关键字，则打包接受到的信息转发给所有聊天室内除发送者的用户。时间的输出由单独的函数处理完成，在各个客户端和服务端做分别处理，保证了消息时间的准确性以及低延迟性。

2 流程概述

本实验进行的 TCP 模型交互流程运用理论课程中讲述的流程进行处理，通过绘制流程图对流程进行直观说明：

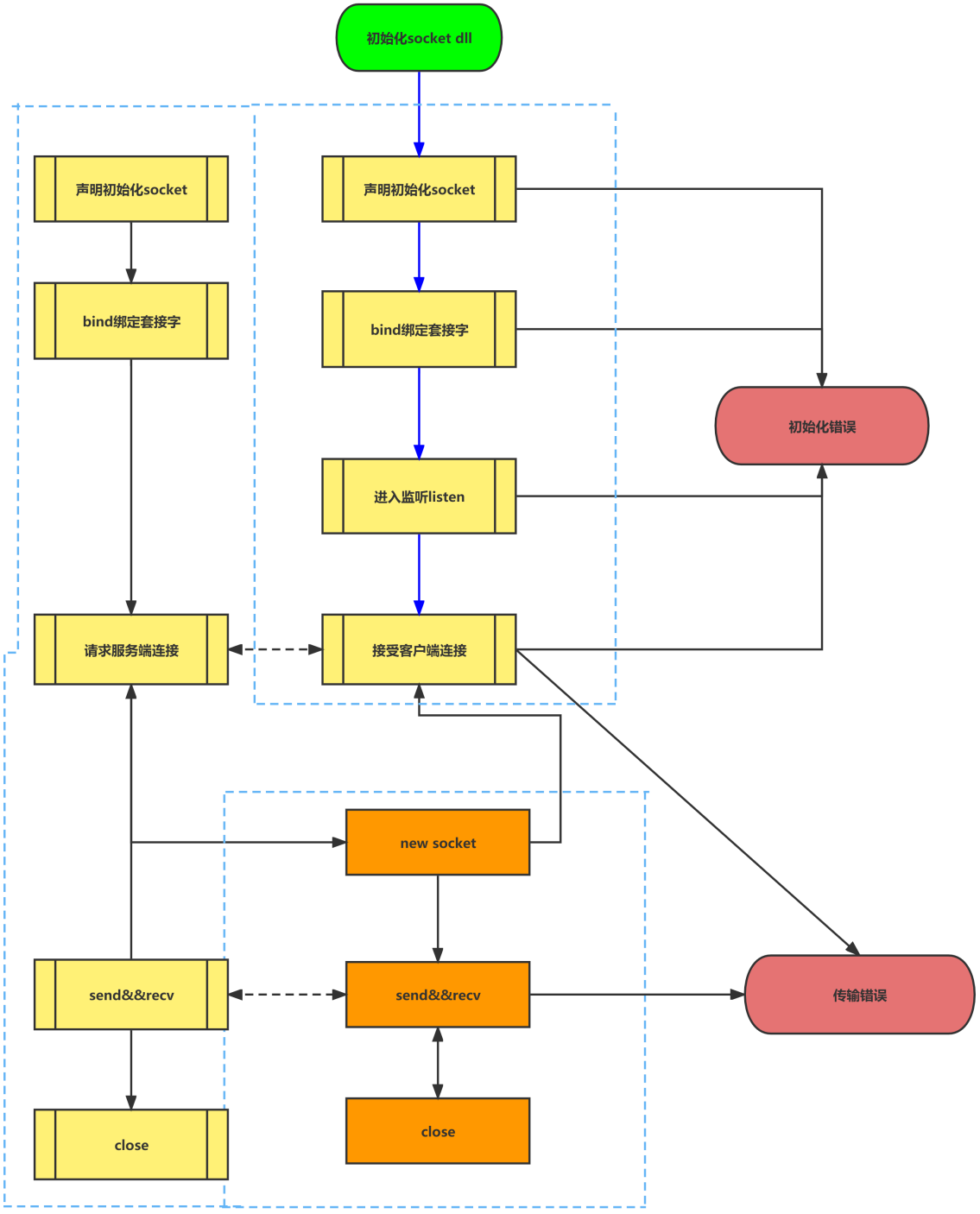

图 2.2: TCP 聊天室工作示意图

首先，我们需要在服务端初始化 socket.dll 并指定我们需要的版本。其次，根绝想要实现的下层协议初始化 socket，在本实验中，我们使用 IPV4 作为 ip 地址、使用流式 socket 以及 TCP 协议进行操作。接下来，我们会根据用户输入的 ip 地址以及端口号完成对 socket 的绑定。至此，聊天室初始化阶段完成

这一步成功完成后，系统就进入了第二个阶段，也就是互相通信的阶段，服务端会进入 listen 状态并接受客户端的申请。一旦客户端申请成功，服务端就会分配一个线程进行交互通信，该线程使用

的交互函数是自定义的 process 函数。在这个函数中，服务端可以接受客户端的发送来的消息并进行打印分发。

在客户端中其处理顺序与服务端基本相似，具体内容包括初始化 socket.dll、初始 socket 以及绑定地址后发送连接请求并与服务端进行通信

3 代码展示

3.1 设计思想

程序运行总体流程是服务端首先开启，等待客户端连接，客户端发送的消息都将转呈给服务端处理，服务端将根据数据类型的不同（群发、私聊、功能函数）进行相应处理并对数据进行打包封装，传递给不同的用户或者调用服务端自身的功能函数相应用户请求。

服务端同时也包括日志记录的功能，他会将用户的请求一一记录并展示打印在命令行中。

3.2 服务端

3.2.1 初始化 dll

首先对 WSADATA 进行初始化化，指明我们需要的 dll 的版本，并进行错误检测

初始化 dll

```
WSADATA w;
int ans;
ans = WSAStartup(MAKEWORD(2, 2), &w);//初始化socket dll
if (ans == 0) {
  cout << "----------成功初始化WSADATA----------" << endl;
}
else {
  cout << "----------初始化WSADATA失败----------" << endl;
  cout << "----------详情可参看: " << WSAGetLastError()<<"----------" << endl;
  return 0;
}
```

3.2.2 初始化 socket

接下来，初始化 socket，并为其指定属性，在本次实验中将使用 ipv6 地址、使用 TCP 协议传输流数据。

初始化 socket

```
//----------启动一个socket----------
SOCKET s;
s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);//初始化
    IPV-4, 下层协议选择数据流, 根据要求选择TCP协议
if (s == INVALID_SOCKET) {
  cout << "----------socket创建失败----------" << endl;
  cout << "----------详情可参看: " << WSAGetLastError() << "----------" << endl;
```

```
    WSACleanup();//释放dll资源
    return 0;
  }
  else {
    cout << "-----------socket创建成功-----------" << endl;
  }
```

3.2.3 绑定套接字地址

随后，生成一个 socketaddress_in 用来绑定，服务端需要的 ip 地址和端口号，这两个数据需要用户输入。

绑定套接字地址

```
  //----------启动一个socket----------
  //绑定套接字，将一个指定地址绑定到指定的Socket
  sockaddr_in sa;
  memset(&sa, 0, sizeof(sa));
  sa.sin_family = PF_INET;
  cout << "----------请输入提供的ip号码----------" << endl;
  cout << "****如果想使用本地ip输入localhost即可****" << endl;
  cin >> ipaddress;
  cout << "----------请输入欲使用的端口号----------" << endl;
  cout << "****使用默认端口号1234则输入888888****" << endl;
  cin >> port;
  if ((string)ipaddress == "localhost") { strcpy(ipaddress, "127.0.0.1"); }
  if (port <= 0 || port > 65536) port = 1234;
  sa.sin_addr.s_addr = inet_addr(ipaddress);
      //查看源码，可以看到这个就是帮我们对S_un.S_addr进行赋值
  sa.sin_port = htons(port);  //端口
  cout << ipaddress << endl;
  cout << port << endl;

  ans = bind(s, (SOCKADDR*)&sa, sizeof(SOCKADDR));//执行绑定
  if (ans == 0) {
    cout << "**----------ip地址绑定成功----------**" << endl;
  }
  else {
    closesocket(s);//释放socket资源
    WSACleanup();//释放dll资源
    cout << WSAGetLastError();
    cout << "-----------ip地址绑定失败-----------" << endl;
    cout << "----------详情可参看: " << WSAGetLastError() << "-----------" << endl;
    return 0;
  }
```

3.2.4 进入监听状态

如果初始化一切顺利，服务端的 socket 初始化成功，准备进入监听状态了：

进入监听状态

```
//————————启动一个socket————————————
//————————开始进入监听状态————————————
//进入监听状态
if (listen(s, 10) == 0) {//最大队列为10
  cout << "----------初始化多人聊天室成功，最多可容纳10人----------" << endl;
  cout << "----------成功进入监听状态----------" << endl;
}
else {
  cout << "----------监听状态出错----------" << endl;
  cout << "----------详情可参看: " << WSAGetLastError() << "----------" << endl;
  closesocket(s);//释放socket资源
  WSACleanup();//释放dll资源
  return 0;
}
cout <<
    "SUCCESSFULLY!----------一切准备就绪，正在等待客户端连接----------SUCCESSFULLY!"
    << endl;
//————————开始进入监听状态————————————
```

3.2.5 接收请求

使用一个 while（true）结构不断的接受客户端的请求。

接收请求

```
//————————处理成功接受的客户端，为他们分配资源————————
while (true) {
  int size = sizeof(sockaddr_in);
  sockaddr_in clientaddr;
  SOCKET cs;
  cs = accept(s, (SOCKADDR*)&clientaddr, &size);
  if (cs == INVALID_SOCKET) {
    cout << "----------连接客户端失败----------" << endl;
    closesocket(s);//释放socket资源
    WSACleanup();//释放dll资源
    cout << "----------详情可参看: " << WSAGetLastError() << "----------" << endl;
    return 0;
  }
  else {
    //线程属性、线程堆栈大小、线程执行函数、传入线程参数、创建线程参数、新线程ID好
    //LPVOID是一个没有类型的指针，也就是说你可以将任意类型的指针赋值给LPVOID类型的变量（一般作为
    HANDLE cthread = CreateThread(NULL, 0, process, (LPVOID)cs, 0, NULL);
    CloseHandle(cthread);
  }
```

```
20    }
21    //————————处理成功接受的客户端，为他们分配资源——————————
22    closesocket(s);//释放socket资源
23    WSACleanup();//释放dll资源
24    cout << "----------本次聊天室已结束，期待和大家下次会面~----------" << endl;
25  }
```

3.2.6 基本判断函数

主函数就到此为止了，接下来是一些判断函数，他们会在线程函数判断用户是否输入我们规定的关键字中被用到

基本判断函数

```
1   bool judge(char* c) {
2   if (c[0] == 'c' && c[1] == 'h' && c[2] == 'a' && c[3] == 'n' && c[4] == 'g' && c[5]
        == 'e' && c[6] == 'n' && c[7] == 'a' && c[8] == 'm' && c[9] == 'e') {
3     return true;
4   }return false;
5   }
6
7   bool judgesendto(char*c) {
8   if (c[0] == 's' && c[1] == 'e' && c[2] == 'n' && c[3] == 'd' && c[4] == 't' && c[5]
        == 'o') {
9     return true;
10  }return false;
11  }
12  string getNewName(char*c) {
13    string s = "";
14    for (int i = 11; i <= 99; i++) {
15      if (c[i] == '\0'||c[i]==' ' || c[i] == '/t') { return s; }
16      else {
17        s += c[i];
18      }
19    }
20  }
21  string toSend(char* c) {
22    string s = "";
23    for (int i = 7; i <= 99; i++) {
24      if (c[i] == '\0' || c[i] == ' ' || c[i] == '/t') { return s; }
25      else {
26        s += c[i];
27      }
28    }
29  }
30  char* getMessage(char*c) {
31    memset(message, 0, 100);
32    int j = 0;
33    int count = 0;
```

```
34    for (int i = 0; i <= 99; i++) {
35      if (count >= 2) {
36        message[j++] = c[i];
37      }
38      if (c[i] == '\0' || c[i] == '/t') {
39        return message;
40      }
41      else {
42        if (c[i] == ' ') { count++; }
43      }
44    }
45  }
46  }
```

3.2.7 线程实现部分

在线程处理函数中，我们需要首先判断他是不是带有合法关键字，如果带有合法关键字，需要对其进行特殊处理。

如果用户输入了 quit，则代表用户已经退出了聊天室，我们需要把他从 map 中剔除掉，后续不再给他转发消息

退出处理

```
1   SOCKET cs = (SOCKET)p;
2   if (mymap.find(cs) == mymap.end()) {
3     mymap.insert(pair<SOCKET, int>(cs, 1));
4   }
5   else {
6     mymap[cs] = 1;
7   }
8   char* name = new char[100];
9   strcpy(name, to_string(cs).data());
10  send(cs, (const char*)name, 100, 0);
11  cout << "****----------------------------------------------------------------****" <<
        endl;
12  outTime();
13  cout << "----------新的用户" << to_string(cs).data() <<
        "成功加入聊天!! ----------" << endl;
14
15  int rflag;
16  int sflag;
17  int flag = 1;
18  char* sendbuffer = new char[100];
19  char* receivebuffer = new char[100];
20  //需要server处也打印一下
21  do {
22    rflag = recv(cs, receivebuffer, 100, 0);
23    if (receivebuffer[0]=='q'&& receivebuffer[1] == 'u'&& receivebuffer[2] == 'i'&&
          receivebuffer[3] == 't'&&receivebuffer[4]=='\0') {
```

```
24      mymap[cs] = 0;
25      if (allname.find(to_string(cs).data()) == allname.end()) {
26        cout << "----------用户" << to_string(cs).data() << "退出了群聊-----------"
              << endl;
27        break;
28      }else {
29        cout << "----------用户" << allname[to_string(cs).data()] <<
              "退出了群聊----------" << endl;
30        break;
31      }
32    }
```

如果用户输入了 changename，则需要在 allname 中为该 socket 对应的名字进行改名操作。

更改名称处理

```
1   if (judge(receivebuffer)) {
2       cout << "****----------------------------------------------------------------****"
            << endl;
3       string newName = getNewName(receivebuffer);
4       allname[to_string(cs).data()] = newName;
5       cout << "----------用户" << to_string(cs).data() << "将昵称改为" <<
            newName<<"------------" << endl;
6       continue;
7     }
```

如果用书输入了 sendto，则代表着这条消息是单独发送给某个人的私聊，不能使用 for 循环进行循环转发给聊天似的每一个人，而是需要在 map 中查找特定的用户并将消息发送给他。

私聊处理

```
1   if (judgesendto(receivebuffer)) {
2       bool stflag = false;
3       string s = toSend(receivebuffer);
4       for (auto it : mymap) {
5         if (allname[to_string(it.first).data()] == s) {
6           strcpy(sendbuffer, "用户");
7           if (allname.find(to_string(cs).data()) == allname.end()) {
8             strcat(sendbuffer, to_string(cs).data());
9           }
10          else {
11            strcat(sendbuffer, allname[to_string(cs).data()].c_str());
12          }
13          strcat(sendbuffer, " 对你说: ");
14          //const char* m = getMessage(sendbuffer);
15          cout << message << endl;
16          strcat(sendbuffer, (const char *)receivebuffer);
17          cout <<
                "****----------------------------------------------------------****"
               << endl;
```

```
18            cout << "在";
19            outTime();
20            if (allname.find(to_string(cs).data()) == allname.end()) {
21              cout << "用户" << cs << "说: " << receivebuffer << endl;
22            }
23            else {
24              cout << "用户" << allname[to_string(cs).data()] << "说: " <<
                    receivebuffer << endl;
25            }
26            //cout << "————————这条记录只有" << s << "可以看到————————" << endl;
27            send(it.first, sendbuffer, 100, 0);
28            stflag = true;
29            break;
30          }
31          if (to_string(it.first).data() == s) {
32              strcpy(sendbuffer, "用户");
33              if (allname.find(to_string(cs).data()) == allname.end()) {
34                strcat(sendbuffer, to_string(cs).data());
35              }
36              else {
37                strcat(sendbuffer, allname[to_string(cs).data()].c_str());
38              }
39              strcat(sendbuffer, " 对你说: ");
40              //const char* m = getMessage(sendbuffer);
41              strcat(sendbuffer, (const char *)receivebuffer);
42              //strcat(sendbuffer, (const char*)receivebuffer);
43              cout <<
                    "****------------------------------------------------------------****"
                    << endl;
44              cout << "在";
45              outTime();
46              if (allname.find(to_string(cs).data()) == allname.end()) {
47                cout << "用户" << cs << "说: " << receivebuffer << endl;
48              }
49              else {
50                cout << "用户" << allname[to_string(cs).data()] << "说: " <<
                      receivebuffer << endl;
51              }
52              //cout << "————————这条记录只有" << to_string(it.first).data() <<
                    "可以看到————————" << endl;
53              send(it.first, sendbuffer, 100, 0);
54              stflag = true;
55              break;
56            }
57          }
58          if (stflag == false) {
59            strcpy(sendbuffer, "发送失败");
60            send(cs, sendbuffer, 100, 0);
61            cout<<"有一条私聊发送失败"<<endl;
```

```
62        }
63        continue;
64      }
```

如果消息中不包含任何关键字，那么就是默认为普通的群发消息，使用 for 循环遍历 map，并对其中每一个合法的 socket 进行转发

群发处理

```
1   if (rflag > 0) {
2
3       strcpy(sendbuffer, "用户");
4       if (allname.find(to_string(cs).data()) == allname.end()) {
5         strcat(sendbuffer, to_string(cs).data());
6       }
7       else {
8         strcat(sendbuffer, allname[to_string(cs).data()].c_str());
9       }
10      strcat(sendbuffer," 说: ");
11      strcat(sendbuffer, (const char*)receivebuffer);
12      cout << "****----------------------------------------------------------------****"
            << endl;
13      cout << "在";
14      outTime();
15      if (allname.find(to_string(cs).data()) == allname.end()) {
16        cout << "用户" << cs << "说: " << receivebuffer << endl;
17      }
18      else {
19        cout << "用户" << allname[to_string(cs).data()] << "说: " << receivebuffer <<
              endl;
20      }
21      vector<SOCKET> temp;
22      for (auto it : mymap) {
23        if (it.first != cs && it.second == 1) {
24          auto ans = send(it.first, sendbuffer, 100, 0);
25          if (ans == SOCKET_ERROR) {
26            cout << "----------向用户" << it.first << "发送失败, " << "错误码:  " <<
                  WSAGetLastError()<<"----------" << endl;
27            cout << "---------已经将用户" << it.first << "移除队列---------" << endl;
28            temp.push_back(cs);
29          }
30        }
31      }
32      for (int i = 0; i < temp.size(); i++) {
33        mymap.erase(temp[i]);
34      }
35    }
36    else {
37      flag = 0;
```

```
    }
  } while (rflag != SOCKET_ERROR && flag != 0);
  //mymap[cs] = 0;

  return 0;
```

3.3 客户端

3.3.1 初始化

客户端的初始化部分和服务端基本相同，在这里不再赘述，直接将代码贴出。这里和服务端最大的区别就是每一个客户端都需要两个线程，他们分别负责接收消息和发出消息，这样就不需要按顺序发送消息了。这两个线程的句柄被保存在了数组中

客户端初始化

```
//————————初始化WSADATA————————
WSADATA w;
SOCKET s;
int ans;
ans = WSAStartup(MAKEWORD(2, 2), &w);
if (ans == 0) {
  cout << "----------成功初始化WSADATA----------" << endl;
}
else {
  cout << "----------初始化WSADATA失败----------" << endl;
  cout << "----------具体错误可以参考: " << WSAGetLastError() << "---------" <<
      endl;
  return 0;
}
//————————初始化WSADATA————————
//————————初始化socket————————
s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
if (s == INVALID_SOCKET) {
  cout << "----------初始化socket错误---------" << endl;
  cout << "----------具体错误可以参考: " << WSAGetLastError()<<"---------" << endl;
  WSACleanup();
  return 0;
}
else {
  cout << "---------成功初始化socket----------" << endl;
}
//————————初始化socket————————
//————————进行连接尝试————————
sockaddr_in sa;
cout << "----------请输入想要连接的主机对应的ip地址--------" << endl;
cout << "****如果想使用本地ip输入localhost即可****" << endl;
cin >> ipaddress;
cout << "请输入想要连接的主机的端口号" << endl;
```

```
33  cout << "****使用默认端口号1234则输入888888****" << endl;
34  cin >> port;
35  if ((string)ipaddress == "localhost") { strcpy(ipaddress, "127.0.0.1"); }
36  if (port <= 0 || port > 65536) port = 1234;
37  sa.sin_family = AF_INET;
38  sa.sin_addr.s_addr = inet_addr(ipaddress);
39  sa.sin_port = htons(port);
40  ans = connect(s, (SOCKADDR*)&sa, sizeof(SOCKADDR));
41  if (ans == 0) {
42    cout << "---------成功连接对应主机! 准备开始聊天吧! ---------" << endl;
43  }
44  else {
45    cout << "---------连接主机失败, 请稍后再试---------" << endl;
46    cout << "----------具体原因可以参看:" << WSAGetLastError() <<"---------" << endl;
47    closesocket(s);
48    WSACleanup();
49    return 0;
50  }
51  //----------进行连接尝试----------
52  recv(s, name, 100, 0);//获取用户名
53  cout << "SUCCSSFULLY!----------成功进入聊天室, 您的用户名是----------" << name <<
        endl;
54  cout << "想要更换自己用户名吗? 输入changename
        yourname就可以啦! 主义不要忘记加上空格哦" << endl;
55  cout << "                    私聊可以使用sendto xxx xxxxxxxxxxx的形式~" << endl;
56  cout << "                        输入quit即可退出聊天室啦" << endl;
57  HANDLE t[2];
58  t[0] = CreateThread(NULL, 0, receive, (LPVOID)&s, 0, NULL);
59  t[1] = CreateThread(NULL, 0, mysend, (LPVOID)&s, 0, NULL);
60  WaitForMultipleObjects(2, t, TRUE, INFINITE);
61  CloseHandle(t[1]); CloseHandle(t[0]);
62  closesocket(s);
63  WSACleanup();
```

3.3.2 接收线程

接收线程

```
1  DWORD WINAPI receive(LPVOID p) {
2    int rflag;
3    SOCKET* s = (SOCKET*)p;
4    char* receivebuffer = new char[100];
5    while (true) {
6      rflag = recv(*s, receivebuffer, 100, 0);
7      if (flag && rflag > 0) {
8        cout << "****------------------------------------------------------------****"
           << endl;
9        cout << "在";
```

```
10        outTime();
11        cout << "收到消息: ";
12        cout << receivebuffer;
13        cout << "  说点什么吧! " << endl;
14        cout << "****----------------------------------------------------------------****"
              << endl;
15      }
16      else {
17        closesocket(*s);
18        return 0;
19      }
20    }
21  }
```

3.3.3 发送线程

发送线程

```
1   DWORD WINAPI mysend(LPVOID p) {
2     int sflag;
3     SOCKET* s = (SOCKET*)p;
4     char* sendbuffer = new char[100];
5     while (true) {
6       //cout << "说点什么吧~" << endl;
7       cin.getline(sendbuffer, 100);
8       if (string(sendbuffer) == "quit") {
9         sflag = send(*s, sendbuffer, 100, 0);
10        flag = 0;
11        closesocket(*s);
12        cout << endl << "----------即将退出聊天,期待下次见面--------" << endl;
13        return 1;
14      }
15      sflag = send(*s, sendbuffer, 100, 0);
16      if (sflag == SOCKET_ERROR) {
17        cout << "----------发送失败----------" << endl;
18        cout << "----------具体原因可以参看: " << WSAGetLastError()<<"----------" <<
              endl;
19        closesocket(*s);
20        WSACleanup();
21        return 0;
22      }
23      else {
24
25        outTime();
26        cout << "消息成功发送啦! ";
27        cout << "再说点什么吧!" << endl;
28        cout << "****----------------------------------------------------------------****"
              << endl;
```

```
29        }
30    }
31 }
```

4 效果展示

在这一部分，我们会对上面提及到的功能进行演示并提供相应截图

第一部分是展示普通的多人聊天效果图

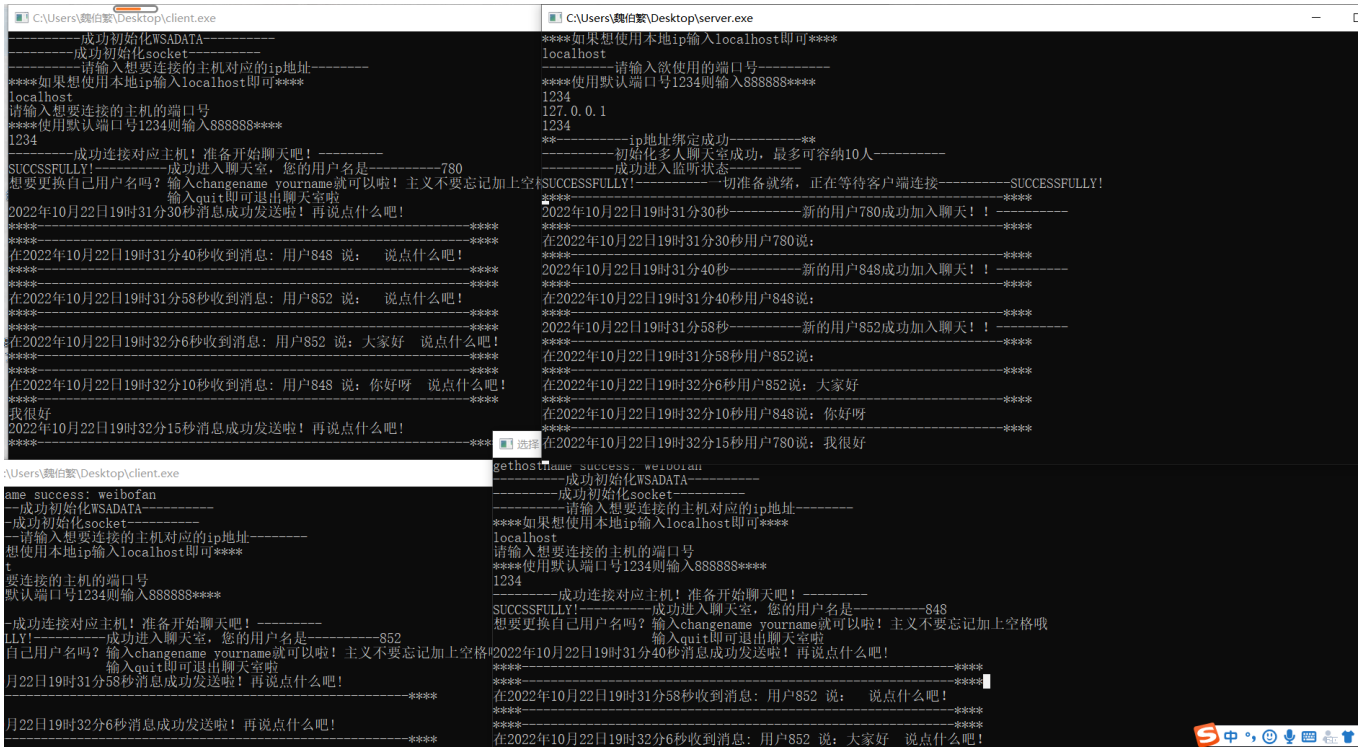

图 4.3: 多人多线程聊天室普通群聊

第二部分是用户更改名字以及改名之后的消息发送效果图

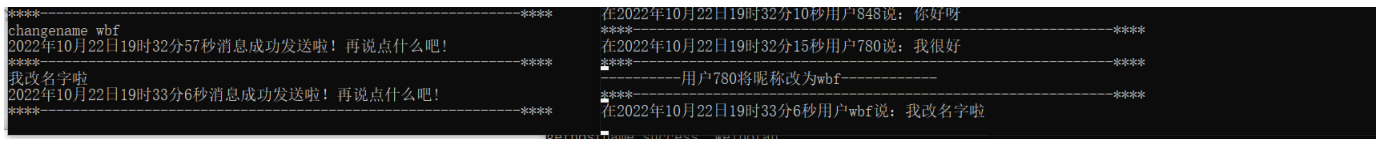

图 4.4: 用户更改用户名

第三部分是展示私聊发送消息

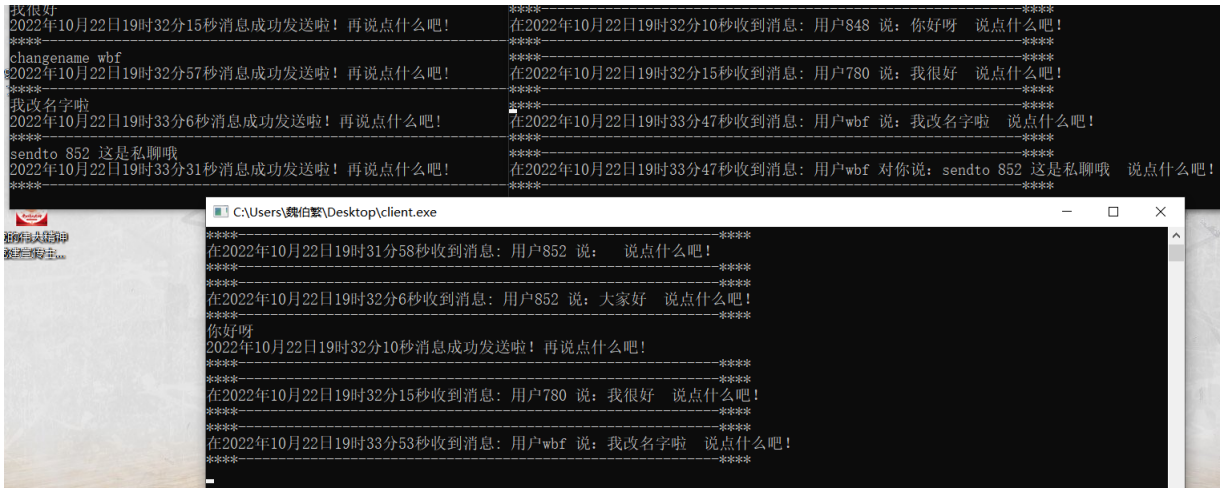

图 4.5: 私聊效果展示

最后一部分展示退出操作

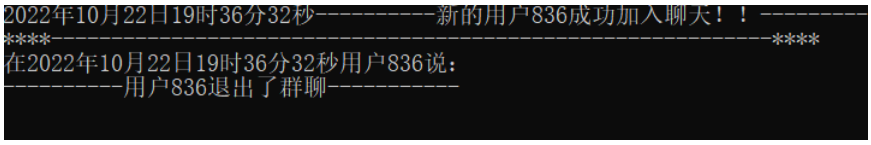

图 4.6: 退出效果展示

5 总结

通过本次实验，我了解掌握了计算机网络应用层编程的基本方法，并且自己动手实现了一个建议的聊天室。同时，在实现基本要求的基础上，我还动手尝试实现了多线程、更改用户名、私聊等附加操作，进一步理解了协议的基本内涵。