南開大學
Nankai University

南　开　大　学

网　络　空　间　安　全　学　院

计算机网络实验报告

利用 Socket 编写一个聊天程序

2111460　张洋

年级：2021 级

专业：信息安全

指导教师：吴英

2023 年 10 月 18 日

目录

一、 前期准备 **1**
（一） 实验要求 1
（二） 功能实现 1
（三） 利用 TCP 服务的应用程序编写步骤 1

二、 协议设计 **2**
（一） 原理 2
（二） 设计 2

三、 代码分析 **4**
（一） 服务器端 4
1. main 函数 4
2. process 线程函数 6
（二） 客户端 12
1. main 函数 12
2. receive 线程函数 14
3. my_send 线程函数 14

四、 结果展示 **15**

五、 总结和心得 **17**
（一） 协议设计 17
（二） 代码分析 18
1. 服务器端 18
2. 客户端 18
（三） 可改进的地方 18

一、 前期准备

(一) 实验要求

- 给出聊天协议的完整说明
- 利用 C 或 C++ 语言，使用基本的 Socket 函数完成程序。
- 使用流式套接字、采用多线程（或多进程）方式完成程序；
- 程序应该有基本的对话界面，但可以不是图形界面。程序应该有正常的退出方式。
- 完成的程序应该支持多人聊天，支持英文和中文聊天；
- 编写的程序应该结构清晰，具有较好的可读性；
- 在实验中观察是否有数据丢失，提交可执行文件、程序源码和实验报告。

(二) 功能实现

- 实现多线程聊天室，客户端可连续发送或接收多条信息，服务端把客户端发送的信息同步给其他用户。
- 服务端使用固定的 ip 地址以及端口号。
- 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。
- 客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。
- 实现了私聊功能，转发内容对服务端可见，在客户端只有接收方可以看到转发的内容。

(三) 利用 TCP 服务的应用程序编写步骤

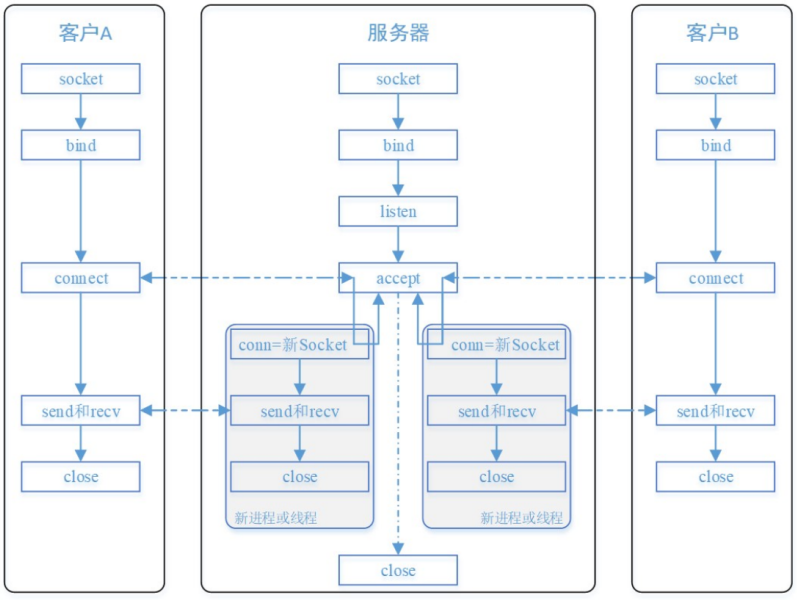


图 1: TCP 服务应用程序

二、 协议设计

(一) 原理

协议定义了通信的方式和规则，而 Socket 套接字则用于在编程中实现这些协议，以便应用程序能够进行网络通信。不同的协议适用于不同类型的应用和通信需求。

1. Socket 套接字

   Socket 是一种编程接口，用于实现网络通信。它提供了一种通用的机制，使不同计算机上的程序能够通过网络进行数据交换。Socket 可以用于不同的编程语言，如 C/C++, Python, Java 等。

   在 Socket 编程中，套接字是一种端点，它可以与其他套接字建立连接，以进行数据的发送和接收。套接字通常分为客户端套接字和服务器套接字，它们协同工作以建立通信连接。

   套接字提供了一种双向通信的能力，使程序能够在网络上发送和接收数据。它们可以实现各种不同类型的通信，包括 TCP（传输控制协议）和 UDP（用户数据报协议）。

2. 协议

   协议是一组规则和约定，用于在计算机网络中实现通信。它定义了数据的格式、传输方法、错误处理和其他通信细节，以确保不同计算机或设备之间可以有效地进行通信。

   在网络通信中，常见的协议包括：

   TCP（传输控制协议）：TCP 是一种面向连接的协议，提供可靠的、有序的数据传输。它用于确保数据在传输过程中不会丢失、损坏或乱序。

   UDP（用户数据报协议）：UDP 是一种无连接的协议，它更轻量级，但不提供与 TCP 相同的可靠性。UDP 适用于需要更低延迟和更快速数据传输的应用。

   HTTP（超文本传输协议）：HTTP 用于在 Web 上传输超文本文档，是 Web 应用的基础。

   FTP（文件传输协议）：FTP 用于在网络上传输文件。

   SMTP（简单邮件传输协议）：SMTP 用于电子邮件传输。

(二) 设计

协议的设计是基于文本消息的通信，其中客户端和服务器通过文本消息进行交互。以下是关于这个协议的设计要点：

1. 客户端连接初始化：
   - 客户端通过套接字连接到服务器的特定 IP 地址和端口。
   - 客户端连接成功后，服务器向客户端发送一个独一无二的用户名，以便客户端识别自己。
2. 消息格式：
   - 客户端和服务器之间的通信基于文本消息，每个消息是一个字符串。
   - 服务器和客户端解析消息以执行不同的操作。
3. 命令格式：
   - change name to <new name>：用于更改客户端的用户名。

- send to <username> <message>：用于向特定用户名的用户发送私人消息。
- quit：用于退出聊天室。
- 普通文本消息：未附带特定命令的消息将被视为广播消息，发送给聊天室中的所有用户。

4. 消息广播：
   - 服务器负责将广播消息发送给聊天室中的所有客户端。
   - 服务器会在每条消息前添加时间戳，以及发送者的用户名（如果已更改）。
5. 更改用户名：
   - 客户端可以通过发送命令 change name to <new name> 来更改其在聊天室中的显示用户名。
   - 服务器会记录新用户名，并在广播消息时使用它。
6. 私聊功能：
   - 客户端可以通过发送命令 send to <username> <message> 来向特定用户名的其他用户发送私人消息。
   - 服务器会查找指定用户名并将消息发送给目标用户。
   - 目标用户会收到私人消息，而其他用户不会看到。
7. 退出协议：
   - 客户端可以发送 quit 命令通知服务器退出聊天室。服务器会将该客户端从活动用户列表中移除。
8. 用户名管理：
   - 服务器通过两个容器来管理用户名和用户状态。
   - store_info 存储了客户端套接字和他们的状态，1 表示活动，0 表示离开。
   - name_change 存储了用户名和对应的套接字，以实现私聊功能。

协议允许客户端与服务器之间进行文本消息通信，支持更改用户名、私聊、广播消息和退出聊天室功能。协议的设计相对简单，但实现了基本的聊天室功能。实际的聊天协议可能更加复杂，包括消息格式、安全性、错误处理等更多的细节。

三、 代码分析

总体流程是服务器端首先开启，绑定 socket，建立监听，等待客户端连接。客户端再开启，绑定 socket，与服务器端建立连接，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。

(一) 服务器端

1. main 函数

在此函数中进行了 Winsock 的初始化、套接字创建、绑定、监听，然后不断接受客户端的连接请求，并创建一个线程用于处理每个客户端的通信。

- 初始化 WSADATA

```
WSADATA wsaData;
struct hostent* remoteHost = NULL;
char serverHostName[128] = {};

int r = WSAStartup(MAKEWORD(2, 2), &wsaData);
if (r != 0)
{
    printf("WSAStartup failed: %d\n", r);
    return 0;
}
printf("WSAStartup succeed!\n");
```

- 打印 hostname

```
r = gethostname(serverHostName, sizeof(serverHostName));
if (r == 0)
{
    printf("gethostname succeed: %s\n", serverHostName);
}
else
{
    printf("gethostname failed(errcode %d)\n", ::GetLastError());
}
cout << "---------Initializing,please wait...-----------" << endl;
```

- 初始化 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 << "Create socket failed." << endl;
    WSACleanup();//释放dll资源
    return 0;
}
else {
    cout << "Create socket succeed." << endl;
}
```

- 绑定 socket

```
sockaddr_in sa;
memset(&sa, 0, sizeof(sa));
sa.sin_family = PF_INET;
ipaddress = "127.0.0.1";
port = 8000; //超过范围 默认8000
sa.sin_addr.s_addr = inet_addr(ipaddress);  //对S_un.S_addr进行赋值
sa.sin_port = htons(port);  //端口
cout << "server IP: " << ipaddress << ":" << port << endl;

r = bind(s, (SOCKADDR*)&sa, sizeof(SOCKADDR));//执行绑定
if (r == 0) {
    cout << "bind succeeed." << endl;
}
else {
    closesocket(s);//释放socket资源
    WSACleanup();//释放dll资源
    cout << "bind failed." << endl;
    return 0;
}
```

- 进入监听状态

```
if (listen(s, 10) == 0) {//最大队列为10
    cout << "----------Initialized chat room successfully with up to 10
        people----------" << endl;
    cout << "listen secceed." << endl;
}
else {
    cout << "listen failed." << endl;
    closesocket(s);//释放socket资源
    WSACleanup();//释放dll资源
    return 0;
}
```

- 接受请求

  使用一个 while（1）结构不断地接受客户端的请求。

```
while (1) {
    int size = sizeof(sockaddr_in);
    sockaddr_in clientaddr;
    SOCKET cs;
    cs = accept(s, (SOCKADDR*)&clientaddr, &size);
    if (cs == INVALID_SOCKET) {
                cout << "Connection failed." << endl;
                closesocket(s);//释放socket资源
                WSACleanup();//释放dll资源
                return 0;
    }
    else{
                //HANDLE CreateThread(LPSECURITY_ATTRIBUTES
                    lpThreadAttributes, SIZE_T dwStackSize,
                //LPTHREAD_START_ROUTINE lpStartAddress, LPVOID lpParameter,
                    DWORD dwCreationFlags, LPDWORD lpThreadId)
                //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
                //LPVOID是一个没有类型的指针，可以将任意类型的指针赋值给
                    LPVOID类型的变量（一般作为参数传递），然后在使用的时候再
                    转换回来
                HANDLE cthread = CreateThread(NULL, 0, process, (LPVOID)cs,
                    0, NULL);
                CloseHandle(cthread);
    }
}
closesocket(s);//释放socket资源
WSACleanup();//释放dll资源
cout << "----------Chat Room Ended, See You Next Time.----------" << endl;
```

2. process 线程函数

在线程处理函数中，我们需要判断用户端传来的信息是不是带有关键字，如果带有关键字，需要对其进行特殊处理。

- 存储用户信息到容器中

  map<SOCKET, int>store_info 中存储了用户的 id 以及是否该用户是否在聊天室中。用户 id 用 to_string(cs).data() 进行随机初始化，store_info[user] = 1 代表在聊天室中，否则表示不在聊天室中。

```
if (store_info.find(cs) == store_info.end()) {
    store_info.insert(pair<SOCKET, int>(cs, 1));
}
else {
    store_info[cs] = 1;
}
char* name = new char[100];
```

```
strcpy(name, to_string(cs).data());
send(cs, (const char*)name, 100, 0);
outTime();
cout << endl;
cout << "----------New user " << to_string(cs).data() << " join in the chat
    successfully!----------" << endl;
cout << "----------Total number : " << ++number << "----------" << endl;
```

- 退出处理

  比较用户输入的单词是不是“quit”，如果是则退出。在服务器日志中打印该用户退出并打印聊天室中剩余的人数。

```
if (strcmp(recvBuff, "quit") == 0) {
    store_info[cs] = 0;
    if (name_change.find(to_string(cs).data()) == name_change.end()) {
        outTime();
        cout << endl;
        cout << "----------User " << to_string(cs).data() << " quits
            .-----------" << endl;
        cout << "----------Total number : " << --number << "----------" <<
            endl;
        break;
    }else {
        outTime();
        cout << endl;
        cout << "----------User " << name_change[to_string(cs).data()] << "
            quits.----------" << endl;
        cout << "----------Total number : " << --number << "----------" <<
            endl;
        break;
    }
}
```

- 更改名称处理

  比较用户输入的前三个单词是不是“change name to”，如果是则获取 newname。在 name_change 容器中记录用户 id 对应的新名字。

```
if (if_change_name(recvBuff)) {
    string newName = getNewName(recvBuff);
    if(strcmp(newName.data(), to_string(cs).data()) == 0){
                cout << "New name is same as old name.Not change." << endl;
    }
    else{
        name_change[to_string(cs).data()] = newName;
        outTime();
        cout << endl;
```

```
        cout << "----------User " << to_string(cs).data() << " change his/her
            nickname to: " << newName<<"------------" << endl;
        continue;
    }
}
```

判断前三个单词是不是“change name to”。

```
bool if_change_name(char* c) {
        char s[14];
        for(int i = 0 ; i < 14 ; i++){
                s[i] = c[i];
        }
        s[14] = '\0';
        if (strcmp(s, "change name to") == 0) {
                return true;
        }return false;
}
```

从输入的字符串中获取想要更改的用户名。

```
string getNewName(char*c) {
        int len = strlen(c);
        string s = "";
        for (int i = 15; i < len ; i++) {
                s += c[i];
        }
        return s;
}
```

- 私聊处理

  用户输入的前两个单词应该是“send to”，第三个单词是接受信息的客户端的用户名，然后输入对该客户端发送的信息。该功能的实现主要依靠在 store_info 容器和 name_change 容器中查找用户名。

```
if (if_private(recvBuff)) {
    bool stflag = false;
    string s = send_to(recvBuff);
    for (auto it : store_info) {
        //查找更改名字的容器
        if (name_change[to_string(it.first).data()] == s) {
            strcpy(sendBuff, "Private message from User ");
            if (name_change.find(to_string(cs).data()) == name_change.end())
                {
                strcat(sendBuff, to_string(cs).data());
            }
            else {
                strcat(sendBuff, name_change[to_string(cs).data()].c_str());
            }
```

```
14              strcat(sendBuff, "␣:␣");
15              const char* m = getMessage(recvBuff);
16              strcat(sendBuff, m);
17              outTime();
18              if (name_change.find(to_string(cs).data()) == name_change.end())
                    {
19                  cout << "User␣" << to_string(cs).data() << "␣:␣" << recvBuff
                        << endl;
20                  cout << "A␣message␣has␣been␣sent␣privately␣by␣User␣" <<
                        to_string(cs).data() << "␣to␣User␣" << name_change[
                        to_string(it.first).data()].c_str() << "␣:␣" << m << endl
                        ;
21              }
22              else {
23                  cout << "User␣" << name_change[to_string(cs).data()] << "␣:␣"
                         << recvBuff << endl;
24                  cout << "A␣message␣has␣been␣sent␣privately␣by␣User␣" <<
                        name_change[to_string(cs).data()] << "␣to␣User␣" <<
                        name_change[to_string(it.first).data()].c_str() << "␣:␣"
                        << m << endl;
25              }
26              send(it.first, sendBuff, 100, 0);
27              stflag = true;
28              break;
29          }
30          else if (to_string(it.first).data() == s) {
31              strcpy(sendBuff, "Private␣message␣from␣User␣");
32              if (name_change.find(to_string(cs).data()) == name_change.end())
                    {
33                  strcat(sendBuff, to_string(cs).data());
34              }
35              else {
36                  strcat(sendBuff, name_change[to_string(cs).data()].c_str());
37              }
38              strcat(sendBuff, "␣:␣");
39              const char *m = getMessage(recvBuff);
40              strcat(sendBuff, m);
41              outTime();
42              if (name_change.find(to_string(cs).data()) == name_change.end())
                    {
43                  cout << "User␣" << to_string(cs).data() << "␣:␣" << recvBuff
                        << endl;
44                  cout << "A␣message␣has␣been␣sent␣privately␣by␣User␣" <<
                        to_string(cs).data() << "␣to␣User␣" << to_string(it.first
                        ).data() << "␣:␣" << m << endl;
45              }
46              else {
47                  cout << "User␣" << name_change[to_string(cs).data()] << "␣:␣"
```

```
                    << recvBuff << endl;
48              cout << "A message has been sent privately by User " <<
                    name_change[to_string(cs).data()] << " to User " <<
                    to_string(it.first).data() << " : " << m << endl;
49          }
50          send(it.first, sendBuff, 100, 0);
51          stflag = true;
52          break;
53      }
54  }
55  if (stflag == false) {
56      strcpy(sendBuff, "Failed");
57      send(cs, sendBuff, 100, 0);
58      cout<<"A private chat failed to send."<<endl;
59  }
60  continue;
61 }
```

判断前两个单词是不是“send to”。

```
1  bool if_private(char*c) {
2          char s[7];
3          for(int i = 0 ; i < 7 ; i++){
4                  s[i] = c[i];
5          }
6          s[7] = '\0';
7          if (strcmp(s, "send to") == 0) {
8                  return true;
9          }return false;
10 }
```

获取接收方的用户名。

```
1  string send_to(char* c) {
2          int len = strlen(c);
3          string s = "";
4          for (int i = 8; i < len; i++) {
5                  if(c[i] == ' ') return s;
6                  else s += c[i];
7          }
8  }
```

获取想要发送的信息。

```
1  char* getMessage(char*c) {
2          int len = strlen(c);
3          memset(message, 0, 100);
4          int j = 0;
5          int count = 0;
6          for (int i = 0; i <= len; i++) {
7                  if (count >= 3) {
```

```
                message[j++] = c[i];
            }
            if (c[i] == '\0' || c[i] == '/t') {
                return message;
            }
            else {
                if (c[i] == ' ') { count++; }
            }
        }
}
```

- 群发处理

  首先查找用户名在服务器端输出日志，然后把信息转发到其余客户端。

```
strcpy(sendBuff, "User ");
if (name_change.find(to_string(cs).data()) == name_change.end()) {
    strcat(sendBuff, to_string(cs).data());
}
else {
    strcat(sendBuff, name_change[to_string(cs).data()].c_str());
}
strcat(sendBuff," : ");
strcat(sendBuff, (const char*)recvBuff);
outTime();
if (name_change.find(to_string(cs).data()) == name_change.end()) {
    cout << "User " << to_string(cs).data() << " : " << recvBuff << endl;
}
else {
    cout << "User " << name_change[to_string(cs).data()] << " : " << recvBuff
        << endl;
}
//socket类型的数组
vector<SOCKET> temp;
//遍历store_info中的元素，即所有用户
for (auto it : store_info) {
    //找到不是当前客户端的用户并且运行状态为1
    if (it.first != cs && it.second == 1) {
        int a = send(it.first, sendBuff, 100, 0);
        if (a == SOCKET_ERROR) {
            cout << "Send to " << it.first << "failed." << endl;
            cout << "Move User " << it.first << " away." << endl;
            temp.push_back(cs);
        }
    }
}
for (int i = 0; i < temp.size(); i++) {
    store_info.erase(temp[i]);
}
```

(二) 客户端

1. main 函数

主函数中主要实现的功能如下。客户端的初始化部分和服务端基本相同，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。

- 初始化 Winsock。
- 创建一个套接字用于连接到服务器。
- 连接到服务器。
- 接收从服务器分配的用户名。
- 创建两线程，一个用于接收消息（receive 函数），另一个用于发送消息（my_send 函数）。
- 等待两个线程的完成，并释放相关资源。

```
int main() {
        // Declare and initialize variables
        WSADATA wsaData;
        struct hostent* remoteHost = NULL;
        char serverHostName[128] = {};

        // Initialize Winsock
        int r = WSAStartup(MAKEWORD(2, 2), &wsaData);
        if (r != 0)
        {
                printf("WSAStartup failed: %d\n", r);
                return 0;
        }
        printf("WSAStartup succeed!\n");
        //gethostname
        r = gethostname(serverHostName, sizeof(serverHostName));
        if (r == 0)
        {
                printf("gethostname succeed: %s\n", serverHostName);
        }
        else
        {
                printf("gethostname failed(errcode %d)\n", ::GetLastError());
        }
        cout << "---------Initializing,please wait...-----------" << endl;
        //socket
        SOCKET s;
        s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
        if (s == INVALID_SOCKET) {
                cout << "Create socket failed." << endl;
                WSACleanup();
```

```
32                  return 0;
33          }
34          else {
35                  cout << "Create socket succeed." << endl;
36          }
37
38          sockaddr_in sa;
39          ipaddress = "127.0.0.1";
40          port = 8000;
41          sa.sin_family = AF_INET;
42          sa.sin_addr.s_addr = inet_addr(ipaddress);
43          sa.sin_port = htons(port);
44          cout << "Server IP: " << ipaddress << ":" << port << endl;
45          r = connect(s, (SOCKADDR*)&sa, sizeof(SOCKADDR));
46          if (r == 0) {
47                  cout << "Connection succeed." << endl;
48          }
49          else {
50                  cout << "Connection failed." << endl;
51                  closesocket(s);
52                  WSACleanup();
53                  return 0;
54          }
55
56          recv(s, name, 100, 0);
57          cout << "----------Chat begins.Here's your username: " << name <<"
                ----------" << endl;
58          cout << "Here are some tips:" << endl;
59          cout << "Input 'change name to <new name>' to change your name" <<
                endl;
60          cout << "Input 'send to <username> <message>' to send message to
                someone privately." << endl;
61          cout << "Input 'quit' to quit this chat room.";
62          cout << "Input message directly to send message to all people in this
                chat room." << endl;
63          cout << "That's all! Have a good day!" << endl;
64          HANDLE t[2];
65          t[0] = CreateThread(NULL, 0, receive, (LPVOID)&s, 0, NULL);
66          t[1] = CreateThread(NULL, 0, my_send, (LPVOID)&s, 0, NULL);
67          WaitForMultipleObjects(2, t, TRUE, INFINITE);
68          CloseHandle(t[1]); CloseHandle(t[0]);
69          closesocket(s);
70          WSACleanup();
71  }
```

2. receive 线程函数

用于接收来自服务器的消息。它在循环中持续接收消息，如果接收到消息且 flag 为真，它会在控制台上打印消息，否则关闭套接字。

```
DWORD WINAPI receive(LPVOID p) {
        int r;
        SOCKET* s = (SOCKET*)p;
        char* recvBuff = new char[100];
        while (true) {
                r = recv(*s, recvBuff, 100, 0);
                if (flag && r > 0) {
                        outTime();
                        cout << recvBuff << endl;
                }
                else {
                        closesocket(*s);
                        return 0;
                }
        }
}
```

3. my_send 线程函数

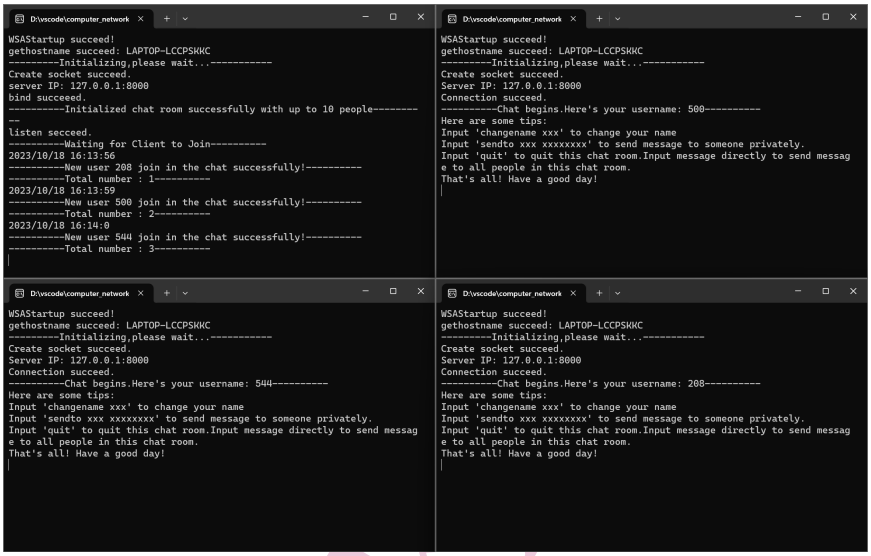
用于发送消息到服务器。它在循环中等待用户输入消息，将消息发送到服务器，如果用户输入”quit”，则发送退出消息，将 flag 设置为 0，并关闭套接字。

```
DWORD WINAPI my_send(LPVOID p) {
        int r;
        SOCKET* s = (SOCKET*)p;
        char* sendBuff = new char[100];
        while (true) {
                cin.getline(sendBuff, 100);
                if (string(sendBuff) == "quit") {
                        r = send(*s, sendBuff, 100, 0);
                        flag = 0;
                        closesocket(*s);
                        cout << "You quit! Goodbye!" << endl;
                        return 1;
                }
                r = send(*s, sendBuff, 100, 0);
                if (r == SOCKET_ERROR) {
                        cout << "Sending messages failed." << endl;
                        closesocket(*s);
                        WSACleanup();
                        return 0;
                }
                else {
                        outTime();
```

```
23                          cout << "Sending messages succeed." << endl;
24                  }
25          }
26  }
```

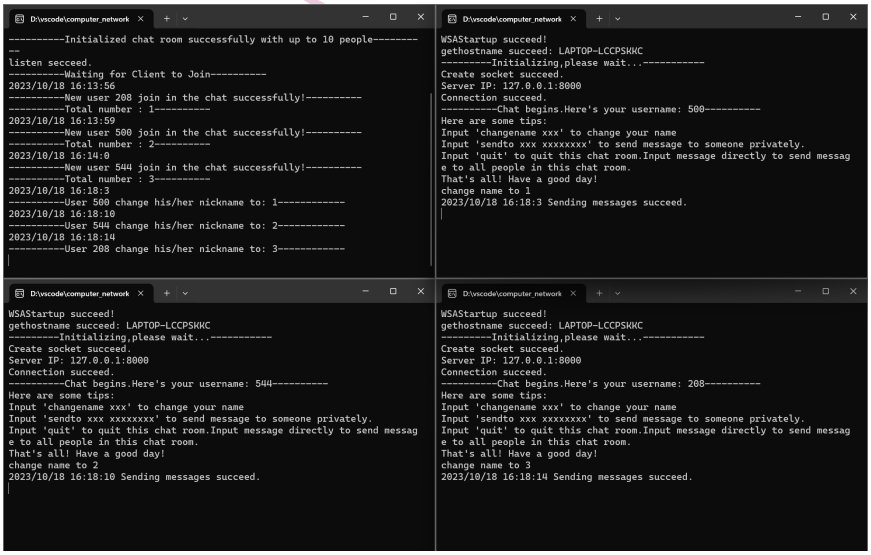
四、 结果展示

1. 运行 server 和三个 client



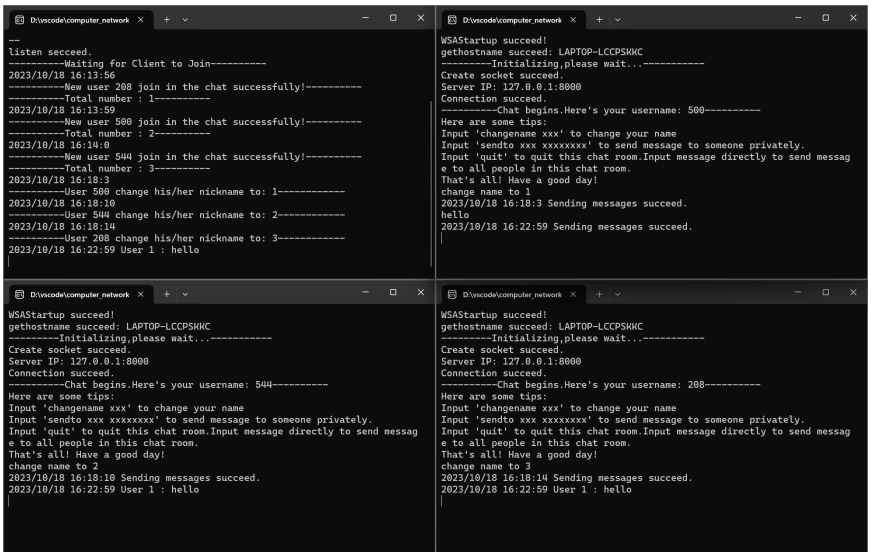
三个客户端均成功连接上服务端。

2. 为了方便观察信息，把三个客户端的名字使用“change name to 1”，“change name to 2”，“change name to 3”进行更改。



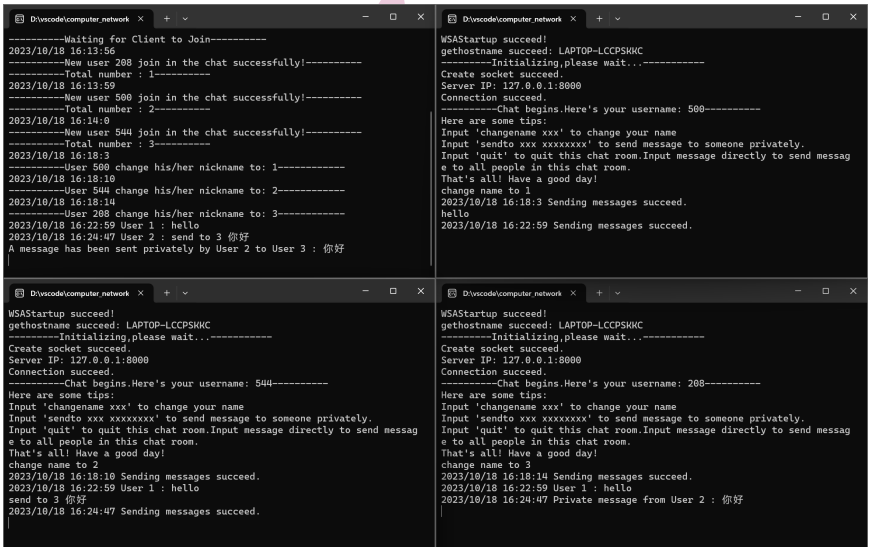
可以看到服务器端记录了三条信息，分别是三个客户端更改名字的信息。客户端输出信息表明更改名字成功。

3. 用户 1 向聊天室中发送“hello”。

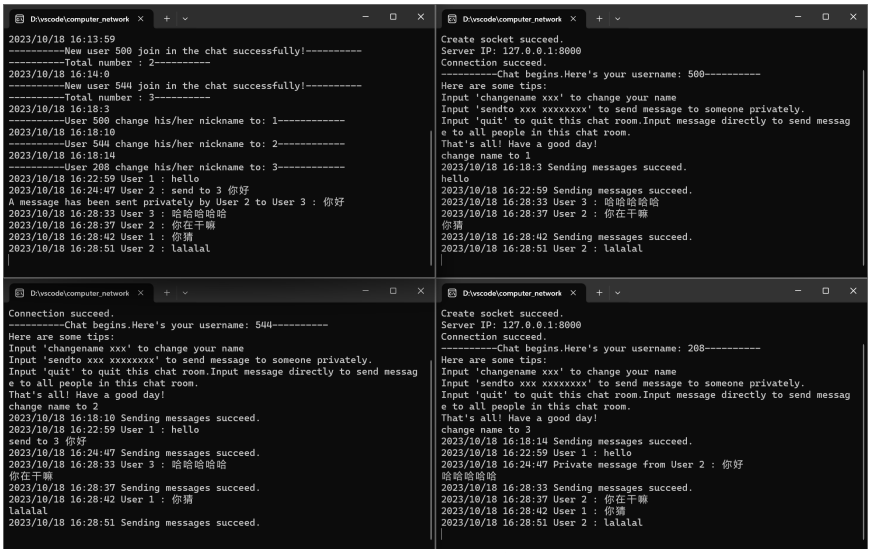


用户 1 发送消息后记录到日志中，服务器端向其他用户广播用户 1 发送的消息到聊天框里。

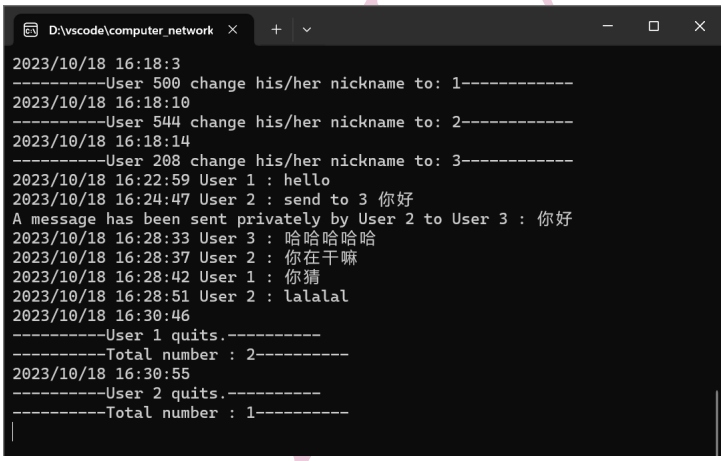
4. 用户 2 输入“send to 3 你好”，表明用户 2 向用户 3 发送了一条私聊信息。可以看到该行为记录在日志中，并且发送的信息只能转发给用户 3，用户 1 并没有收到用户 2 的信息。



5. 又随便输入了几条信息，可以发现聊天支持中英文。



6. 用户 1，2 输入“quit”退出



日志中记录了用户 1，2 退出的动作并实时显示剩余人数，客户端退出后自动关闭窗口。

可以看到我们成功实现了一个可以多人聊天的聊天室。同时，在实现基本要求的基础上，还增加了更改用户名、私聊等操作，进一步理解了协议的基本内涵。

五、 总结和心得

(一) 协议设计

在本次实验中，我设计了一种简单的文本消息通信协议，用于实现基本的聊天室功能。这个协议包括了以下关键要点:

1. 客户端和服务器之间通过文本消息进行通信，支持消息格式解析和处理。

2. 协议定义了不同的消息类型，包括更改用户名、发送私聊消息、退出聊天室等。

3. 服务器负责将广播消息发送给聊室中的所有客户端，并在消息前添加时间戳和发送者用户名。
4. 客户端可以更改其在聊天室中的显示用户名，并服务器会记录新用户名。
5. 实现了私聊功能，允许用户向特定用户名的其他用户发送私人消息。
6. 协议的设计使得客户端和服务器可以进行基本的聊天室交互，包括广播和私聊功能。

(二) 代码分析

在实验中，我编写了客户端和服务器端的程序，用于演示协议的工作原理。以下是代码的主要功能和关键点：

1. 服务器端

- 服务器端主要用于接受客户端的连接请求，然后创建线程用于处理每个客户端的通信。
- 服务器端能够解析并处理不同的消息类型，包括更改用户名、发送私聊消息、退出聊天室和广播消息。
- 服务器端通过存储用户信息和状态（存活或离开）的容器来管理用户。
- 服务器端还具有日志记录功能，能够记录用户的请求和聊天室的人数变化。

2. 客户端

- 客户端通过套接字与服务器建立连接，并在成功连接后接收分配的用户名。
- 客户端创建两个线程，一个用于接收服务器发送的消息，另一个用于发送用户输入的消息。
- 客户端能够解析用户输入，包括发送私聊消息和退出聊天室。
- 客户端能够与服务器进行文本消息通信，实现了广播和私聊功能。

(三) 可改进的地方

- 更详细的协议设计在协议设计部分，可以提供更详细的信息，如消息格式的具体定义，消息头的字段和含义等。这有助于更好地理解协议的工作原理。
- 错误处理和异常情况在代码分析部分，可以更详细地介绍如何处理错误和异常情况。这包括套接字通信中的错误处理、超时处理等。
- 扩展功能可以通过添加图形界面、加强安全性等使得聊天可视化功能更加完善。

通过这个实验，我学习了如何设计一个简单的聊天协议，并用 C++ 编程语言实现了客户端和服务器端程序。这次实验帮助我更好地理解了套接字编程和网络通信的基本概念，以及协议设计的重要性。我也学到了如何管理用户信息和状态，以实现一个简单但完整的聊天室应用。