



南開大學

Nankai University

计算机学院  
计算机网络实验报告

实验 1：利用 **Socket** 编写一个聊天程序

姓名：刘浩泽

学号：2212478

专业：计算机科学与技术

班级：计算机卓越班

2024 年 10 月 19 日

# 目录

<b>1 实验要求</b>	<b>2</b>
<b>2 基本模块构成</b>	<b>2</b>
2.1 服务端 . . . . .	2
2.1.1 主线程 . . . . .	2
2.1.2 子线程 . . . . .	4
2.2 客户端 . . . . .	4
2.2.1 主线程 . . . . .	4
2.2.2 接收服务端消息线程 . . . . .	6
2.2.3 发送用户输入线程 . . . . .	7
<b>3 聊天协议设计</b>	<b>7</b>
3.1 TCP 传输协议 . . . . .	7
3.2 服务端、客户端消息协议及退出机制（语法、语义） . . . . .	8
3.3 客户端命令输入协议（语法、语义） . . . . .	9
3.4 支持英文和中文信息 . . . . .	9
3.5 多人聊天程序中的时序设计 . . . . .	10
3.5.1 连接阶段 . . . . .	10
3.5.2 消息发送阶段 . . . . .	10
3.5.3 退出阶段 . . . . .	11
3.5.4 异常处理阶段 . . . . .	11
3.5.5 具体流程示例 . . . . .	11
3.6 其他聊天协议设计 . . . . .	12
<b>4 具体功能实现及代码讲解</b>	<b>12</b>
4.1 基本的聊天功能 . . . . .	12
4.2 用户获取帮助 . . . . .	14
4.3 文件上传功能 . . . . .	15
4.4 显示当前所有用户在线状态 . . . . .	17
4.5 发送私聊信息 . . . . .	18
4.6 强制指定用户下线 . . . . .	20
4.7 清空客户端历史消息 . . . . .	21
4.8 查看历史聊天记录 . . . . .	22
4.9 将指定用户添加到朋友圈 . . . . .	23
4.10 将指定用户添加到黑名单 . . . . .	25
<b>5 实验内容总结</b>	<b>25</b>

## 1 实验要求

(1) 给出你设计的聊天协议的完整说明。(2) 利用 C 或 C++ 语言, 使用基本的 Socket 函数完成程序, 不允许使用 CSocket 等封装后的类编写程序。(3) 使用流式 Socket 完成程序。(4) 程序应该有基本的对话界面, 但可以不是图形界面, 还应该有正常的退出方式。(5) 程序应该支持多人聊天, 支持英文和中文信息。(6) 编写的程序应该结构清晰, 具有较好的可读性。(7) 现场演示。(8) 提交程序源码、可执行代码和实验报告。

## 2 基本模块构成

多人聊天程序的实现主要由两个基本模块组成, 分别是服务端和客户端。服务端在启动时负责初始化聊天程序, 并开始监听客户端的连接请求。一旦有客户端连接, 服务端就会为每个客户端创建一个工作线程, 负责处理该客户端的消息收发。服务端将收到的消息转发给其他在线的客户端, 从而实现多人聊天的功能。此外, 服务端还需要维护在线客户端列表, 跟踪客户端的连接状态, 并定期向客户端广播在线用户信息。

客户端模块负责连接服务器, 并提供用户友好的操作界面。客户端接收用户输入的消息或命令, 并将其发送到服务器。同时, 客户端也会接收来自服务器的其他用户消息, 并在本地界面上显示出来。客户端支持丰富的聊天命令, 如私聊、上传文件等, 并定期刷新在线用户列表。当用户退出聊天程序时, 客户端会与服务器断开连接, 并将退出消息广播给其他用户。

### 2.1 服务端

服务端程序负责接收客户端的连接请求, 并为每个客户端创建一个专门的工作线程来处理与之的通信。服务端主线程负责监听和接受客户端连接, 而子线程则负责与特定客户端的通信细节, 包括接收消息、处理命令、转发消息给其他客户端等。

#### 2.1.1 主线程

- 首先初始化 Winsock 环境。通过调用 `WSAStartup` 函数来完成 Winsock 的初始化。
- 创建 TCP 服务端套接字。使用 `socket` 函数创建一个 `AF_INET` 地址族、`SOCK_STREAM` 流式套接字类型的套接字。
- 绑定套接字到指定的 IP 地址和端口号。使用 `bind` 函数将套接字绑定到 IP 地址 `INADDR_ANY` 和端口号 8080。
- 开始监听客户端连接请求。调用 `listen` 函数让服务端套接字进入被动监听状态, 可以接受最多 `MAX_CLIENTS` 个客户端的连接请求。
- 当有新的客户端连接时, 接受该连接并创建新的工作线程。使用 `accept` 函数接受客户端的连接请求, 并为该客户端创建一个新的工作线程来处理通信。
- 工作线程负责与该客户端的通信。包括接收客户端消息、转发消息给其他客户端等。
- 主线程继续监听其他客户端的连接请求。直到程序退出。

主线程代码实现如下所示:

---

```
1 void serverThread() {
2     WSADATA wsaData;
3     if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
4         cerr << "Failed to initialize Winsock." << endl;
5         return;
6     }
7     SOCKET serverSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
8     if (serverSocket == INVALID_SOCKET) {
9         cerr << "Failed to create server socket." << endl;
10        WSACleanup();
11        return;
12    }
13    sockaddr_in serverAddr;
14    memset(&serverAddr, 0, sizeof(serverAddr));
15    serverAddr.sin_family = AF_INET;
16    serverAddr.sin_port = htons(8080);
17    serverAddr.sin_addr.s_addr = INADDR_ANY;
18    if (bind(serverSocket, (SOCKADDR*)&serverAddr, sizeof(serverAddr)) == SOCKET_ERROR) {
19        cerr << "Failed to bind server socket." << endl;
20        closesocket(serverSocket);
21        WSACleanup();
22        return;
23    }
24    if (listen(serverSocket, MAX_CLIENTS) == SOCKET_ERROR) {
25        cerr << "Failed to listen on server socket." << endl;
26        closesocket(serverSocket);
27        WSACleanup();
28        return;
29    }
30    cout << "Server is listening on port 8080." << endl;
31    logMessage("Server started on port 8080.");
32    while (true) {
33        SOCKADDR_IN clientAddr;
34        int clientAddrSize = sizeof(clientAddr);
35        SOCKET clientSocket = accept(serverSocket, (SOCKADDR*)&clientAddr, nullptr);
36        if (clientSocket == INVALID_SOCKET) {
37            cerr << "Failed to accept client connection." << endl;
38            continue;
39        }
40        char ipStr[INET_ADDRSTRLEN];
41        inet_ntop(AF_INET, &clientAddr.sin_addr, ipStr, sizeof(ipStr));
```

```
42
43     // 创建新的客户端线程
44     thread(clientThread, clientSocket, string(ipStr)).detach();
45 }
46 closesocket(serverSocket);
47 WSACleanup();
48 }
```

---

### 2.1.2 子线程

- 接收客户端发送的用户名。使用 `recv` 函数接收客户端发送的用户名。
- 将该客户端信息添加到在线客户端列表中。利用 `make_shared` 创建一个 `ClientInfo` 对象, 存储客户端的套接字、用户名、IP 地址和在线状态, 并使用 `lock_guard` 加锁后将其添加到 `clients` 列表中。同时更新 `userStatusMap` 记录客户端的在线状态。
- 向其他在线客户端广播新用户加入的消息。使用 `broadcastMessage` 函数将新用户加入的消息广播给其他在线客户端。
- 进入消息处理循环, 接收客户端发送的各种消息和命令, 并做出相应的处理。使用 `recv` 函数接收客户端消息, 并根据消息内容调用不同的处理函数。
- 处理各种聊天命令, 如私聊、上传文件、踢出用户等。对于不同的命令, 调用相应的处理函数, 如 `handlePrivateMessage`、`uploadFile`、`handleAddFriendRequest` 等。
- 当客户端断开连接时, 从在线客户端列表中移除该客户端, 并向其他客户端广播离线消息。使用 `lock_guard` 加锁后从 `clients` 列表中移除该客户端, 并调用 `broadcastMessage` 函数广播离线消息。

子线程的代码涉及到具体功能实现, 会在第四部分进行具体讲解。

## 2.2 客户端

客户端程序主线程负责建立与服务端的连接, 并通过两个工作线程来处理与服务端的通信。主线程负责接收服务端发送的消息并显示在屏幕上, 而子线程则负责接收用户的输入并发送给服务端。

### 2.2.1 主线程

- 首先初始化 Winsock 环境。通过调用 `WSAStartup` 函数来完成 Winsock 的初始化。
- 创建一个 TCP 客户端套接字。使用 `socket` 函数创建一个 `AF_INET` 地址族、`SOCK_STREAM` 流式套接字类型的套接字。
- 连接到服务端。使用 `connect` 函数连接到服务端的 IP 地址和端口号。
- 创建两个工作线程: 一个负责接收服务端消息并显示, 另一个负责接收用户输入并发送给服务端。
- 等待两个工作线程执行结束。使用 `join` 函数等待两个线程执行完毕。

- 清理 Winsock 环境并退出程序。调用 WSACleanup 函数清理 Winsock 环境。

主线程代码实现如下所示：

---

```
1 // 设置控制台/命令行窗口的代码页为 GBK
2 SetConsoleOutputCP(936);
3 SetConsoleCP(936);
4 WSADATA wsaData;
5 if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
6     cerr << "无法初始化 Winsock。" << endl;
7     return 1;
8 }
9 SOCKET sockfd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
10 if (sockfd == INVALID_SOCKET) {
11     cerr << "无法创建客户端套接字。" << endl;
12     WSACleanup();
13     return 1;
14 }
15 sockaddr_in serverAddr;
16 memset(&serverAddr, 0, sizeof(serverAddr));
17 serverAddr.sin_family = AF_INET;
18 serverAddr.sin_port = htons(8080);
19 if (inet_pton(AF_INET, "192.168.0.1", &serverAddr.sin_addr) <= 0) {
20     cerr << "IP 地址格式错误。" << endl;
21     closesocket(sockfd);
22     WSACleanup();
23     return 1;
24 }
25 if (connect(sockfd, (SOCKADDR*)&serverAddr, sizeof(serverAddr)) == SOCKET_ERROR) {
26     cerr << "无法连接到服务器。" << endl;
27     closesocket(sockfd);
28     WSACleanup();
29     return 1;
30 }
31 thread clientThreadInstance(clientThread, sockfd);
32 thread inputThreadInstance(inputThread, sockfd);
33 clientThreadInstance.join();
34 inputThreadInstance.join();
35 WSACleanup();
36 return 0;
```

---

### 2.2.2 接收服务端消息线程

客户端的这个子线程负责接收服务端发送的消息并显示在屏幕上：

- 使用 `recv` 函数从套接字中读取服务端发送的消息。
- 根据消息内容, 采用不同的颜色将消息显示在屏幕上。例如, 将私聊消息显示为亮蓝色。
- 在显示完服务端消息后, 提示用户输入新的消息。

接收服务端消息线程的代码实现如下所示：

```
1 int flag = 1;
2 void clientThread(SOCKET sockfd) {
3     char buffer[BUFFER_SIZE];
4     int bytesRead;
5     // 发送用户名
6     cout << "请输入您的用户名: ";
7     string username;
8     if (flag) {
9         getline(cin, username);
10    }
11    cout << "输入消息 ('q'退出, '/upload < 文件路径 >'上传文件): " << endl;
12    flag = 0;
13    send(sockfd, username.c_str(), username.length(), 0);
14    while (true) {
15        // 接收服务器消息
16        bytesRead = recv(sockfd, buffer, BUFFER_SIZE - 1, 0);
17        if (bytesRead <= 0) {
18            cerr << "连接服务器中断。" << endl;
19            closesocket(sockfd);
20            return;
21        }
22        buffer[bytesRead] = '\0';
23        string serverMessage(buffer);
24        // 检查是否是私聊消息
25        if (serverMessage.find("[私聊]") == 0) {
26            // 如果是私聊消息, 使用不同的颜色显示
27            setConsoleColor(11); // 11 表示亮蓝色
28            cout << endl << serverMessage << endl;
29            setConsoleColor(7); // 恢复默认颜色
30        }
31        else if (serverMessage.find("User") == 0) {
32            setConsoleColor(11); // 11 表示亮蓝色
33            cout << endl << serverMessage << endl;
```

```

34         setConsoleColor(7); // 恢复默认颜色
35     }
36     else {
37         cout << endl << "[服务器] " << endl << buffer << endl; // 日志输出
38     }
39     // 输出完服务器消息后, 提示用户输入消息
40     cout << "输入消息 ('q'退出, '/upload < 文件路径 >'上传文件): ";
41 }
42 }

```

### 2.2.3 发送用户输入线程

客户端的这个子线程负责接收用户输入并发送给服务端:

- 提示用户输入消息。
- 检查用户输入, 如果是 q 则退出程序, 如果是 /upload < 文件路径 > 则执行文件上传功能。
- 如果是普通聊天消息, 则使用 send 函数将其发送给服务端。
- 如果是特殊命令, 如私聊、踢出用户、添加好友等, 则解析命令并发送给服务端。

发送用户输入线程的代码涉及到具体功能实现, 会在第四部分进行具体讲解。

## 3 聊天协议设计

### 3.1 TCP 传输协议

实验要求使用流式套接字完成, 因此本实验采取 **TCP 传输协议**。TCP 是面向连接的协议, 能够确保数据的顺序传输和完整性, 适合多人聊天系统中的消息传递。TCP 传输协议的实现方式如下, 本实验采取多线程的方式来处理客户端连接。对于服务端的 server 线程, 主要用来维护一个服务器套接字, 监听客户端连接, 并通过多线程处理每个客户端的连接。其中使用 TCP 协议来创建服务端套接字, 并将套接字绑定到指定的 IP 地址和端口号。

```

1  SOCKET serverSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
2  sockaddr_in serverAddr;
3  memset(&serverAddr, 0, sizeof(serverAddr));
4  serverAddr.sin_family = AF_INET;
5  serverAddr.sin_port = htons(8080);
6  serverAddr.sin_addr.s_addr = INADDR_ANY;
7  if (bind(serverSocket, (SOCKADDR*)&serverAddr, sizeof(serverAddr)) == SOCKET_ERROR) {
8      cerr << "Failed to bind server socket." << endl;
9      closesocket(serverSocket);
10     WSACleanup();

```



```
11         return;  
12     }
```

同时客户端也采取这种方式建立 TCP 流式套接字与服务端进行连接, 建立连接后, 同样使用 TCP 协议的通信方式, 实现消息的发送和接收。

### 3.2 服务端、客户端消息协议及退出机制（语法、语义）

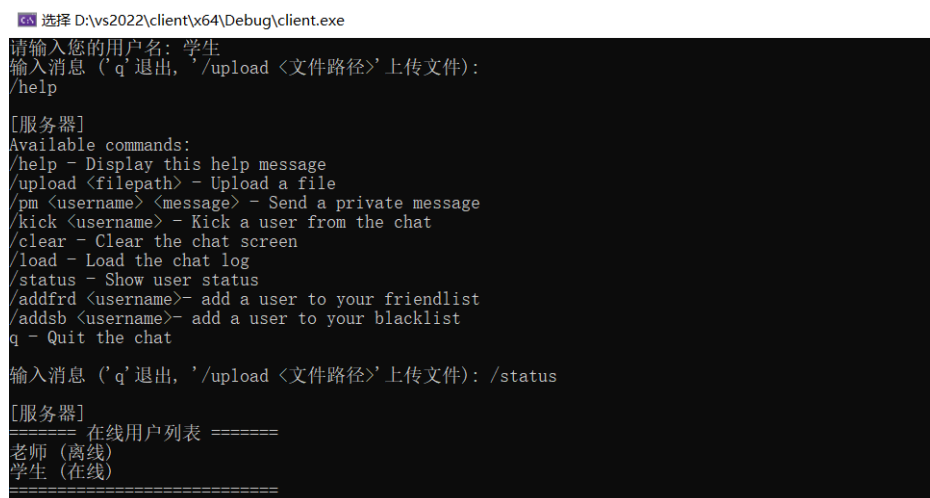
首先服务端启动时首先会打印一个**欢迎界面**用于用户界面交互, 并提供一个命令提示符, 提示用户输入消息或命令。图 3.1 中可以看到服务端的输出包括: 聊天室启动时输出监听端口信息 (格式为 [Message + 端口号] 和 [服务端启动时间 + Message]), 用户加入或退出聊天室时输出相关信息 (格式为 [事件发生时间 + 用户名 (IP) + Message]) 以及用户发送消息时, 格式化输出消息内容 (格式为 [消息发送时间 + 用户: 用户名 (IP) + said: Message])。不同类型的消息输出分别使用白色, 绿色, 黄色和红色进行区分。并且当用户加入或退出聊天室时, 打印当前聊天室的在线用户列表, 提示剩余用户当前在线人数。

```
D:\vs2022\server\x64\Debug\server.exe  
***** 欢迎进入多人聊天系统 *****  
*****  
请输入消息或命令 (如: /upload 发送文件, 'q' 退出, /help获取功能索引):  
Server is listening on port 8080.  
2024-10-18 01:23:05 - Server started on port 8080.  
2024-10-18 01:23:59 - 老师 (204.204.204.204) has joined the chat.  
  
===== 在线用户列表 =====  
老师 (在线)  
=====  
2024-10-18 01:24:11 - Broadcast: 时间: Fri Oct 18 01:24:11 2024  
用户: 老师 (204.204.204.204)  
said: hello, 大家好  
老师 has left the chat.  
2024-10-18 01:30:16 - 老师 has left the chat.  
  
===== 在线用户列表 =====  
老师 (离线)  
=====
```

图 3.1: 多人聊天程序消息格式

客户端的输出包括: 客户端启动时会请求输入当前用户的用户名, 并提供命令提示符, 提示用户输入消息或命令。当前用户或其他聊天室用户发送消息后, 消息发送到服务端的同时也会在客户端进行显示, 模拟微信聊天时向上滑动可以查看历史聊天记录的功能。同时服务器的回显内容会带有 [服务器] 的标志, 表明这是历史状态信息, 供用户进行查看。当用户输入 q 时, 客户端会与服务器断开连接。客户端显示退出的消息提示, 并且服务端对于用户退出事件进行消息广播来告知其他用户。

### 3.3 客户端命令输入协议（语法、语义）



```

选择 D:\vs2022\client\x64\Debug\client.exe
请输入您的用户名: 学生
输入消息 ('q' 退出, '/upload <文件路径>' 上传文件):
/help

[服务器]
Available commands:
/help - Display this help message
/upload <filepath> - Upload a file
/pm <username> <message> - Send a private message
/kick <username> - Kick a user from the chat
/clear - Clear the chat screen
/load - Load the chat log
/status - Show user status
/addfrd <username>- add a user to your friendlist
/addsb <username>- add a user to your blacklist
q - Quit the chat

输入消息 ('q' 退出, '/upload <文件路径>' 上传文件): /status

[服务器]
==== 在线用户列表 =====
老师 (离线)
学生 (在线)
=====

输入消息 ('q' 退出, '/upload <文件路径>' 上传文件):

```

图 3.2: 客户端命令格式

用户在客户端输入 `/help` 可以获取当前多人聊天程序支持的所有功能及命令。命令格式如图 3.2 所示, 支持的命令包括 `/help`: 获取多人聊天程序支持的所有功能及命令, `/upload <filepath>`: 上传指定文件到服务器, `/pm <username> <message>`: 向指定用户发送私聊消息, `/kick <username>`: 强制指定用户下线, `/clear`: 清空请求该命令用户客户端的所有内容, `/load`: 获取历史聊天记录, 显示在请求该命令用户的客户端界面, `/status`: 显示当前多人聊天程序中的所有用户状态, 展示在请求该命令用户的客户端界面, `/addfrd <username>`: 将指定用户添加到当前用户的朋友圈, `/addsb <username>`: 将指令用户添加到当前用户的黑名单。图中以 `/status` 命令进行了展示, 可以看到用户界面中显示了当前多人聊天程序中的所有用户状态。

### 3.4 支持英文和中文信息

本次实验中, 聊天程序的传输报文、服务端以及客户端的终端输出均采用 **GBK 编码方式**, 从而同时支持英文和中文信息的输出。

同时在上传文件成功后, 服务端会将文件内容输出到终端并广播给所有用户文件内容。这里由于 windows 平台上文件编码格式为 UTF-8 编码, 而控制台和命令行窗口则使用 GBK 编码。如果直接获取文件内容, 并将其原封不动地发送到服务端和输出到终端, 中文部分就会出现乱码。因此我们需要在客户端解析文件内容时, 先指定 UTF-8 编码读取文件流, 然后将 UTF-8 编码转换成 GBK 编码, 最后再将报文传输到服务端进行输出。这样多人聊天程序在进行聊天消息输出和上传文件时才不会出现中文乱码的情况。

在 `main` 函数里需要设置控制台和命令行窗口的代码页为 GBK 编码方式, 同时需要实现一个上述的 `Utf8ToGbk` 编码转化函数, 具体代码实现如下所示:

```

1 // 设置控制台/命令行窗口的代码页为 GBK
2 SetConsoleOutputCP(936);
3 SetConsoleCP(936);
4 std::string Utf8ToGbk(const std::string& utf8)
5 {

```

```

6     int len = MultiByteToWideChar(CP_UTF8, 0, utf8.c_str(), -1, NULL, 0);
7     std::unique_ptr<wchar_t[]> wstr(new wchar_t[len + 1]);
8     memset(wstr.get(), 0, (len + 1) * sizeof(wchar_t));
9     MultiByteToWideChar(CP_UTF8, 0, utf8.c_str(), -1, wstr.get(), len);
10    len = WideCharToMultiByte(CP_ACP, 0, wstr.get(), -1, NULL, 0, NULL, NULL);
11    std::unique_ptr<char[]> str(new char[len + 1]);
12    memset(str.get(), 0, (len + 1) * sizeof(char));
13    WideCharToMultiByte(CP_ACP, 0, wstr.get(), -1, str.get(), len, NULL, NULL);
14    return std::string(str.get());
15 }

```

实现效果如图 3.3 所示，可以看到文件内容和聊天信息中的中文、英文信息均可以正常输出，不会出现中文乱码的情况。

```

D:\vs2022\server\x64\Debug\server.exe
***** 欢迎进入多人聊天系统 *****
*****

请输入消息或命令 (如: /upload 发送文件, 'q' 退出, /help获取功能索引):
Server is listening on port 8080.
2024-10-18 15:34:13 - Server started on port 8080.
2024-10-18 15:34:32 - lhz (204.204.204.204) has joined the chat.

===== 在线用户列表 =====
lhz (在线)
=====
2024-10-18 15:34:44 - Broadcast: 时间: Fri Oct 18 15:34:44 2024
用户: lhz (204.204.204.204)
said: 我是lhz
2024-10-18 15:35:02 - Broadcast: 时间: Fri Oct 18 15:35:02 2024
用户: lhz (204.204.204.204)
said: /clea
2024-10-18 15:35:13 - Broadcast: 时间: Fri Oct 18 15:35:13 2024
用户: lhz (204.204.204.204)
said: adasadsad
this is a test file!!!
welcome to the chat!
欢迎加入!!

D:\vs2022\client\x64\Debug\client.exe
选择 D:\vs2022\client\x64\Debug\client.exe
输入消息 ('q' 退出, '/upload <文件路径>' 上传文件): /help

[服务器]
Available commands:
/help - Display this help message
/upload <filepath> - Upload a file
/pm <username> <message> - Send a private message
/kick <username> - Kick a user from the chat
/clear - Clear the chat screen
/load - Load the chat log
/status - Show user status
/addfrd <username>- add a user to your friendlist
/addsb <username>- add a user to your blacklist
q - Quit the chat

输入消息 ('q' 退出, '/upload <文件路径>' 上传文件): /upload d:/1.txt

文件上传成功! d:/1.txt

[服务器]
时间: Fri Oct 18 15:35:13 2024
用户: lhz (204.204.204.204)
said: adasadsad
this is a test file!!!
welcome to the chat!
欢迎加入!!

输入消息 ('q' 退出, '/upload <文件路径>' 上传文件):

```

图 3.3: 支持英文和中文信息

### 3.5 多人聊天程序中的时序设计

#### 3.5.1 连接阶段

1. 客户端启动，建立与服务器的连接。
2. 客户端向服务器发送用户名。
3. 服务器接收用户名，并将该用户添加到在线用户列表中，发送确认消息。

#### 3.5.2 消息发送阶段

1. 客户端发送消息或命令到服务器。
2. 服务器接收消息，验证命令类型：
  - 普通消息：服务器广播消息给所有在线用户。

- 特殊命令:

- /help: 服务器发送帮助信息给请求的客户端。
- /upload: 服务器接收文件上传请求, 客户端发送文件名和文件内容。
- /pm <username> <message>: 服务器解析私聊命令并向指定用户转发消息。
- /kick <username>: 服务器处理踢人命令, 踢除指定用户并通知其他在线用户。
- /addfrd <username>: 服务器处理添加好友请求, 将用户添加到好友列表。
- /addsb <username>: 服务器处理添加黑名单请求, 将用户添加到黑名单。
- /load: 服务器加载聊天记录并发送给请求的客户端。
- /status: 服务器返回在线用户列表状态。
- /clear: 服务器清除聊天屏幕 (在客户端执行)。

### 3.5.3 退出阶段

1. 客户端发送退出命令 q。
2. 服务器接收退出命令, 更新用户状态, 将该用户标记为离线, 并通知其他用户。
3. 服务器关闭该用户的 socket 连接。

### 3.5.4 异常处理阶段

1. 若客户端在连接过程中或消息传递过程中断开连接, 服务器将捕获该异常, 更新用户状态并通知其他在线用户。

### 3.5.5 具体流程示例

#### 用户连接

1. 客户端 → 发送用户名 → 服务器
2. 服务器 → 添加用户 → 更新在线用户列表
3. 服务器 → 通知所有在线用户, 用户已加入。

#### 用户发送消息

1. 客户端 → 发送消息 → 服务器
2. 服务器 → 广播消息 → 所有在线用户

#### 用户执行命令

1. 客户端 → 发送 /help → 服务器
2. 服务器 → 返回帮助信息 → 客户端

#### 文件上传

1. 客户端 → 发送上传命令 → 服务器
2. 服务器 → 接收文件名和文件内容 → 存储文件

3. 服务器 → 通知其他用户文件已上传。

### 私聊功能

1. 客户端 → 发送 `/pm <username> <message>` → 服务器
2. 服务器 → 查找用户 → 发送私聊信息 → 指定用户。

### 用户退出

1. 客户端 → 发送 `q` → 服务器
2. 服务器 → 更新用户状态 → 通知其他在线用户。

## 3.6 其他聊天协议设计

除了前述的消息传输协议、服务端和用户端命令格式和文件上传协议外，我们还需要考虑聊天程序的其他功能需求，并相应地设计相关的协议：

- **最大支持用户数：**我们定义了一个宏常量 `MAX_CLIENTS = 100` 来限制聊天室最多同时支持 100 个在线用户。这个数值可以根据实际需求进行调整。
- **消息缓冲区大小：**为了支持即时性较强的聊天消息传输，我们定义了一个消息缓冲区大小 `BUFFER_SIZE = 1024`(即 1KB)，并且当输入消息长度超出范围时会进行提示。
- **文件传输缓冲区大小：**为了支持文件的上传和传输，我们定义了一个较大的文件传输缓冲区大小 `FILE_BUFFER_SIZE = 409600`(即 400KB)。在代码中我们采用了分段读取和发送的方式，每次从文件中读取 409600 字节的数据到缓冲区 `buffer` 中，然后再通过 `send` 函数发送给客户端。这个值可以根据实际测试结果进行优化，并且当输入消息长度超出范围时会进行提示。
- **用户状态管理：**为了跟踪在线用户的状态，我们在服务端维护了一个用户列表，记录每个用户的上线/下线状态以及其他相关信息，并及时广播给其他用户。
- **互斥锁的实现：**为了保护 `clients` 列表这个共享资源的访问，我们使用了 `clientsMutex` 互斥锁。当多个工作线程同时操作 `clients` 列表时，互斥锁可以确保数据的一致性和线程安全。
- 在服务端接受到客户端的连接请求后，会为每个客户端创建一个独立的工作线程来处理其消息收发。采取多线程的方式，可以使每一个客户端在任意时刻发送任意数量的消息，同时可以顺利接收到来自其他人的消息，确保用户在聊天过程中能够获得良好的交互体验。

## 4 具体功能实现及代码讲解

Github 链接：[https://github.com/lhz191/nku\\_computer\\_network.git](https://github.com/lhz191/nku_computer_network.git)

### 4.1 基本的聊天功能

`void inputThread(SOCKET sockfd)` 函数是客户端用于接收用户输入并发送消息的线程。其主要功能包括：

- 用户输入提示：通过 `cout` 输出提示，让用户知道可以输入聊天消息、上传文件或退出。

- 消息发送: 当用户输入消息时, 将其获取到 message 变量中。然后通过 send(sockfd, message.c\_str(), message.length(), 0) 将消息发送到服务端。
- 退出机制: 如果用户输入“q”, 说明用户想要退出聊天, 此时会输出提示信息并关闭 sockfd 连接。
- 清屏功能: 如果用户输入“/clear”, 则会调用 clearScreen() 函数清空屏幕, 并重新输出提示信息。
- 文件上传: 如果用户输入以“/upload”开头的消息, 说明想要上传文件。程序会解析出文件路径, 并以二进制方式打开该文件。然后将文件内容按照 UTF-8 编码读取到缓冲区 buffer 中, 并转换为 GBK 编码后发送到服务端。最后输出文件上传成功的提示信息。
- 私聊消息: 如果用户输入以“/pm”开头的消息, 说明想要发送私聊消息。程序会直接将整个消息发送到服务端, 由服务端负责转发给指定的用户。
- 踢出用户: 如果用户输入以“/kick”开头的消息, 说明想要踢出某个用户。程序会将整个消息发送到服务端, 由服务端负责处理。
- 添加好友: 如果用户输入以“/addfrd”开头的消息, 说明想要添加好友。程序会解析出好友的用户名, 并组装成 /addfrd < 用户名 > 的格式发送到服务端。
- 添加黑名单: 如果用户输入以“/addsb”开头的消息, 说明想要将某个用户加入黑名单。程序会解析出黑名单用户的用户名, 并组装成 /addsb < 用户名 > 的格式发送到服务端。

具体代码实现如下所示, 后续部分展示主要功能的具体实现方式及相关代码:

---

```
1 int flag1;
2 void inputThread(SOCKET sockfd) {
3     char buffer[BUFFER_SIZE];
4     int bytesRead;
5     while (true) {
6         // 提示用户输入消息
7         cout << "输入消息 ('q'退出, '/upload < 文件路径 >'上传文件): ";
8         string message;
9         if (flag1 == 1) {
10             getline(cin, message);
11         }
12         flag1 = 1;
13         if (message == "q") {
14             cout << "您已退出聊天。" << endl;
15             closesocket(sockfd);
16             return;
17         }
18         if (message == "/clear")
19         {
20             clearScreen();
21             cout << "输入消息 ('q'退出, '/upload < 文件路径 >'上传文件): ";
```

```

22         continue;
23     }
24     /* 其他功能的用户输入解析 */
25     /* 篇幅限制, 这里省略, 后续进行详细介绍 */
26     //普通聊天信息直接发送到服务端
27     send(sockfd, message.c_str(), message.length(), 0);
28 }

```

当用户加入多人聊天室时, 服务端会用绿色字体提示所有用户该用户加入了聊天室, 并将信息广播到其他用户的客户端。当用户输入 q 离开多人聊天室时, 服务端会用红色字体提示所有用户该用户离开了聊天室, 并将信息广播到其他用户的客户端。实现效果如图 4.4 所示。

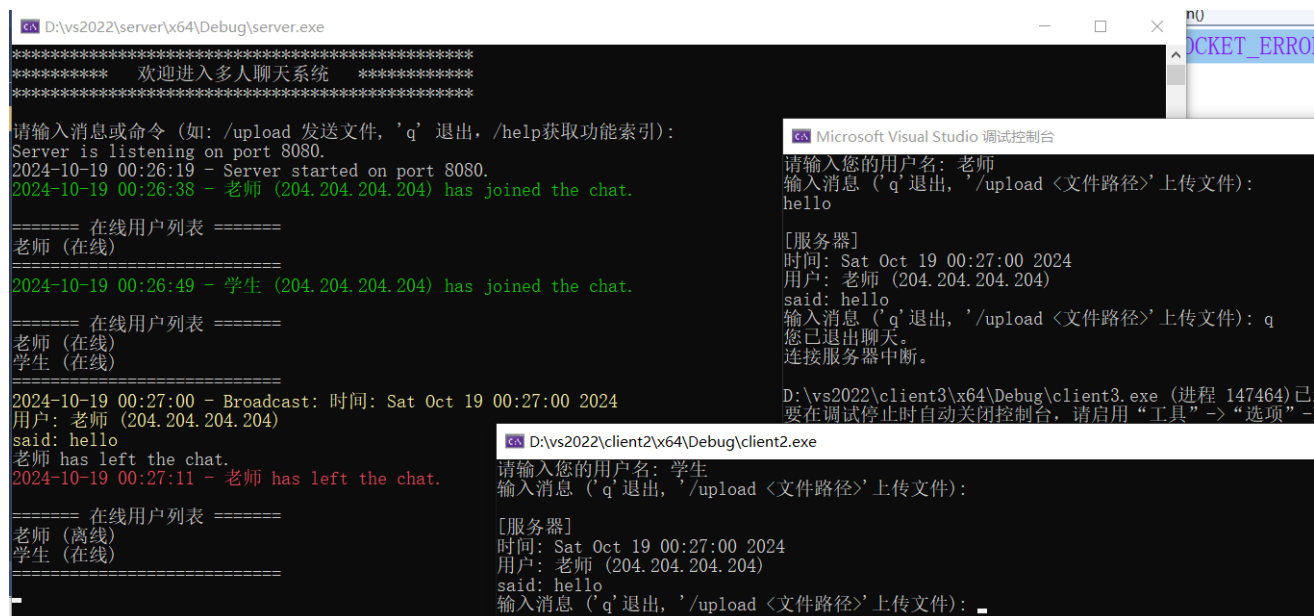


图 4.4: 用户加入和退出时多人聊天室进行提示

## 4.2 用户获取帮助

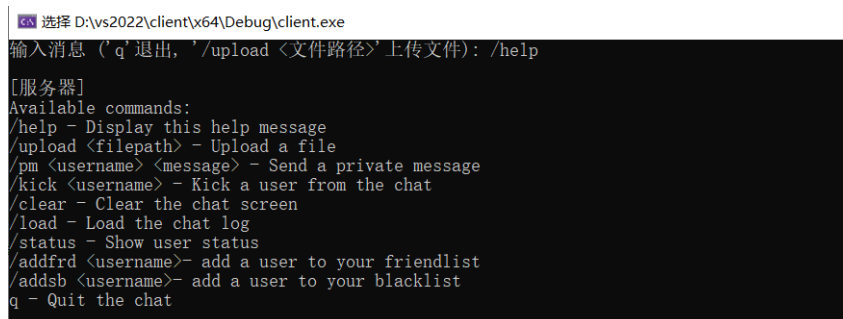


图 4.5: 用户获取帮助

当服务端接受到客户端发来的获取帮助的命令时, 服务端会构造一条详细的帮助信息字符串。该字符串包含了客户端可用的各种命令及其功能说明。然后, 服务端会将这条帮助信息通过 send 函数发



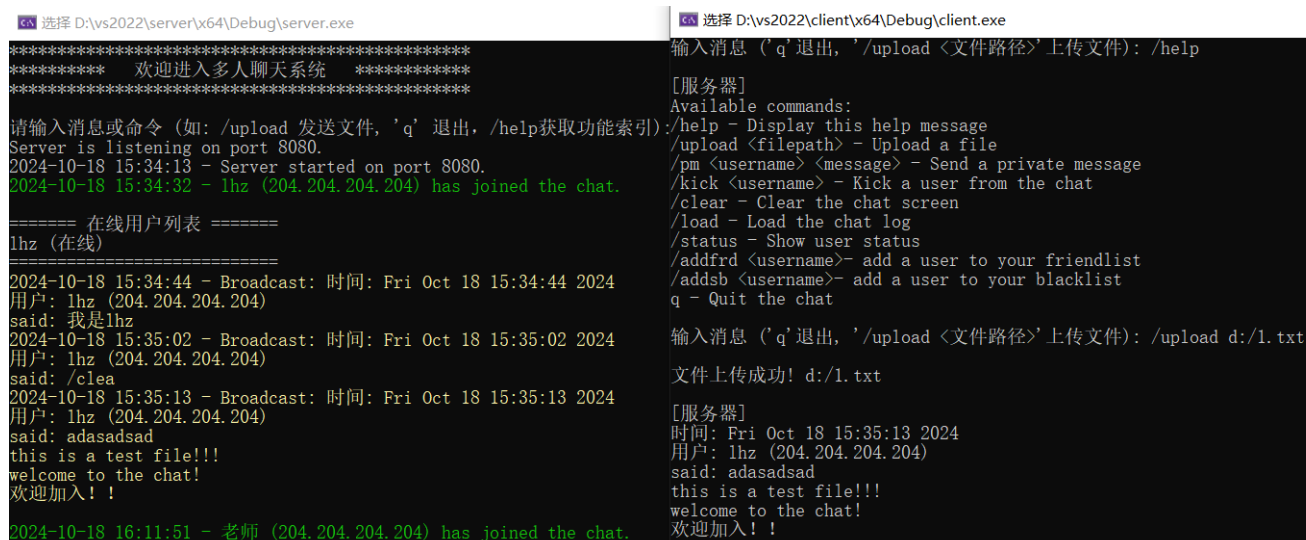
送回给客户端。并且该信息只会在请求帮助的客户端的界面上进行显示，不会像聊天信息一样广播给其他用户或显示在公屏。具体代码实现如下所示：

```

1  if (message == HELP_COMMAND1) {
2      string helpMessage = "Available commands:\n";
3      helpMessage += "/help - Display this help message\n";
4      helpMessage += "/upload <filepath> - Upload a file\n";
5      helpMessage += "/pm <username> <message> - Send a private message\n";
6      helpMessage += "/kick <username> - Kick a user from the chat\n";
7      helpMessage += "/clear - Clear the chat screen\n";
8      helpMessage += "/load - Load the chat log\n";
9      helpMessage += "/status - Show user status\n";
10     helpMessage += "/addfrd <username>- add a user to your friendlist\n";
11     helpMessage += "/addsb <username>- add a user to your blacklist\n";
12     helpMessage += "q - Quit the chat\n";
13     send(clientSocket, helpMessage.c_str(), helpMessage.length(), 0);
14     continue;
15 }

```

### 4.3 文件上传功能



The screenshot displays two terminal windows. The left window, titled '选择 D:\vs2022\server\x64\Debug\server.exe', shows the server's output. It includes a welcome message, a list of online users, and a broadcast message indicating that user 'lhzh' (204.204.204.204) has joined the chat. The right window, titled '选择 D:\vs2022\client\x64\Debug\client.exe', shows the client's input and output. The user enters the command '/upload d:/1.txt', and the server responds with '文件上传成功! d:/1.txt'.

图 4.6: 文件上传功能

当客户端接受到用户输入的文件上传指令后，会从消息中提取出文件路径 `filePath`，接着以二进制模式打开该文件，如果打开失败则输出错误信息并继续等待下一条消息。然后获取文件的大小 `fileSize`。设置文件流的编码为 UTF-8，以确保文件内容能够正确传输。接着开始循环读取文件内容：每次从文件中读取 409600 字节的数据到缓冲区 `buffer` 中。然后获取实际读取的字节数 `bytesRead`。将 UTF-8 编码的 `buffer` 转换为 GBK 编码的 `gbkBuffer`。如果 `bytesRead` 大于 0，则将 `gbkBuffer` 通



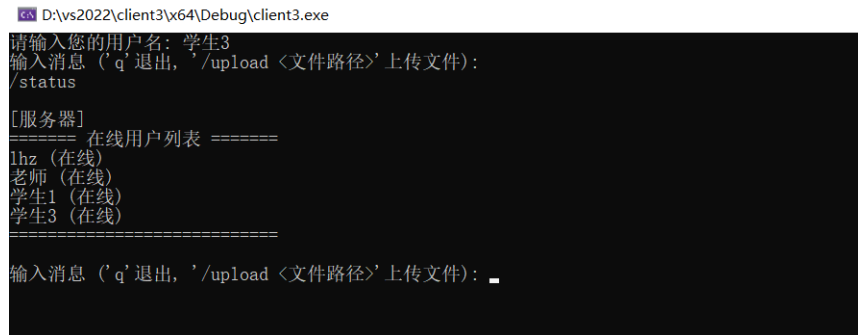
过 send 函数发送给客户端。当文件读取完成后, 输出”文件上传成功!”的提示信息, 并关闭文件。具体代码实现如下所示:

---

```
1 // 文件上传功能
2 if (message.substr(0, 7) == "/upload") {
3     string filePath = message.substr(8); // 获取文件路径
4     ifstream file(filePath, ios::binary);
5     if (!file.is_open()) {
6         cerr << "无法打开文件: " << filePath << endl;
7         continue;
8     }
9     // 获取文件大小
10    file.seekg(0, ios::end);
11    streamsize fileSize = file.tellg();
12    file.seekg(0, ios::beg);
13    // 以 UTF-8 编码发送文件内容
14    char buffer[409600]; // 定义一个缓冲区
15    file.imbue(std::locale("zh_CN.UTF-8")); // 设置文件流的编码为 UTF-8
16    while (file) {
17        // 读取文件的一部分并发送
18        file.read(buffer, sizeof(buffer));
19        streamsize bytesRead = file.gcount(); // 获取实际读取的字节数
20        // 将 UTF-8 编码的 buffer 转换为 GBK 编码
21        std::string gbkBuffer = Utf8ToGbk(std::string(buffer, bytesRead));
22        //cout << buffer << endl;
23        //cout << gbkBuffer << endl;
24        if (bytesRead > 0) {
25            send(sockfd, gbkBuffer.c_str(), gbkBuffer.length(), 0);
26        }
27    }
28    file.close();
29    continue;
30 }
```

---

#### 4.4 显示当前所有用户在线状态



```

D:\vs2022\client3\x64\Debug\client3.exe
请输入您的用户名: 学生3
输入消息 ('q' 退出, '/upload <文件路径>' 上传文件):
/status

[服务器]
===== 在线用户列表 =====
lh2 (在线)
老师 (在线)
学生1 (在线)
学生3 (在线)
=====

输入消息 ('q' 退出, '/upload <文件路径>' 上传文件):

```

图 4.7: 显示当前所有用户在线状态

当服务端接收到用户请求当前所有用户在线状态的命令时，会构建一条包含所有在线用户信息的状态消息字符串 `statusMessage`。为了确保线程安全地访问 `clients` 容器，服务端会先对该容器加锁。然后遍历 `clients` 容器，获取每个用户的在线状态，并将其添加到 `statusMessage` 中。最后，服务端会将 `statusMessage` 通过 `send` 函数发送给请求状态信息的客户端。发送完成后，服务端继续等待下一条来自客户端的消息。并且该信息只会在请求该命令的客户端的界面上进行显示，不会像聊天信息一样广播给其他用户或显示在公屏。具体代码实现如下所示：

---

```

1  if (message == STATUS_COMMAND) {
2      // Build the user status message and send it to the requesting client
3      string statusMessage = "===== 在线用户列表 =====\n";
4      lock_guard<mutex> lock(clientsMutex);
5      for (const auto& client : clients) {
6          statusMessage += client->username + (client->isOnline ? " (在线)\n" :
7              " (离线)\n");
8      }
9      statusMessage += "===== \n";
10     send(clientSocket, statusMessage.c_str(), statusMessage.length(), 0);
11     continue;
12 }

```

---

## 4.5 发送私聊信息

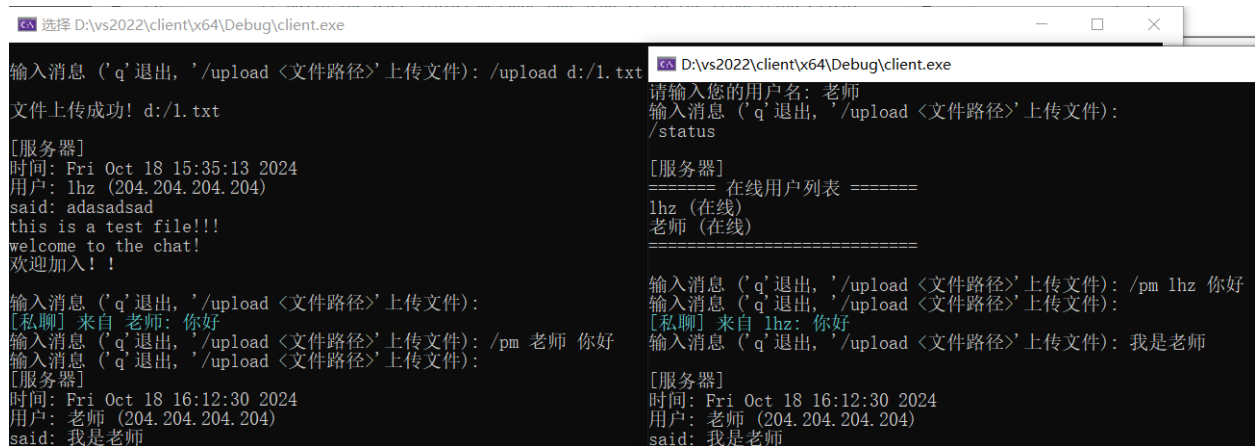


图 4.8: 发送私聊信息

当服务端接收到用户请求发送私聊信息的命令时，首先，服务端会解析出消息中的目标用户名 `recipient` 和私聊内容 `privateMessage`。为了确保线程安全，服务端会先对 `clients` 容器加锁。然后，服务端会遍历 `clients` 容器，查找目标用户是否在线。如果找到了目标用户，服务端会构建一条格式化的私聊消息 `formattedMessage` (例如”[私聊] 来自张三: 你好!”)。最后，服务端会通过 `send` 函数将 `formattedMessage` 发送给目标用户。如果在遍历 `clients` 容器时没有找到目标用户，或者目标用户不在线，服务端会给发起私聊请求的客户端发送一条错误提示。发送完毕后，服务端继续等待下一条消息。并且该信息只会在请求该命令的客户端的界面上进行显示，不会像聊天信息一样广播给其他用户或显示在公屏。具体代码实现如下所示：

```

1  if (message.substr(0, PRIVATE_MESSAGE_COMMAND.size()) == PRIVATE_MESSAGE_COMMAND) {
2      // 解析私聊命令
3      string recipient, privateMessage;
4      // 找到第一个空格
5      size_t firstSpacePos = message.find(" ", PRIVATE_MESSAGE_COMMAND.size());
6      if (firstSpacePos == string::npos) {
7          send(clientSocket,
8              "Invalid private message format. Usage: /pm <username> <message>\n", 64, 0);
9          continue;
10     }
11     // 找到第二个空格
12     size_t secondSpacePos = message.find(" ", firstSpacePos + 1);
13     if (secondSpacePos != string::npos) {
14         // 提取目标用户名
15         recipient = message.substr(firstSpacePos + 1, secondSpacePos -
16             firstSpacePos - 1);
17         // 提取私聊消息
18         privateMessage = message.substr(secondSpacePos + 1);
19     }

```

```
20     else {
21         // 如果没有第二个空格, 说明没有消息内容
22         send(clientSocket,
23             "Invalid private message format. Usage: /pm <username> <message>\n", 64, 0);
24         continue;
25     }
26     // 去掉多余的空白
27     recipient.erase(0, recipient.find_first_not_of(" \t\n")); // 去掉前导空格
28     recipient.erase(recipient.find_last_not_of(" \t\n") + 1); // 去掉后导空格
29     privateMessage.erase(0, privateMessage.find_first_not_of(" \t\n")); // 去掉前导空格
30     // 查找目标用户并发送私聊信息
31     lock_guard<mutex> lock(clientsMutex);
32     bool recipientFound = false;
33     for (const auto& client : clients) {
34         if (client->username == recipient && client->isOnline) {
35             string formattedMessage = "[私聊] 来自 " + username + ": " +
36                 privateMessage;
37             send(client->sockfd, formattedMessage.c_str(),
38                 formattedMessage.length(), 0); // 发送私聊信息
39             recipientFound = true;
40             break;
41         }
42     }
43     if (!recipientFound) {
44         string errorMsg = "用户 " + recipient + " 不在线或不存在.\n";
45         send(clientSocket, errorMsg.c_str(), errorMsg.length(), 0);
46     }
47     continue;
48 }
```

---

## 4.6 强制指定用户下线

```

D:\vs2022\server\x64\Debug\server.exe
2024-10-18 19:32:20 - Broadcast: 时间: Fri Oct 18 19:32:20 2024
用户: 学生1 (204.204.204.204)
said: 你们好
2024-10-18 19:32:57 - 学生 has been kicked from the chat by 学生1
学生 has been kicked from the chat by 学生1
2024-10-18 19:33:21 - Broadcast: 时间: Fri Oct 18 19:33:14 2024
用户: 学生 (204.204.204.204)
said: 12
学生 has left the chat.
2024-10-18 19:33:21 - 学生 has left the chat.

===== 在线用户列表 =====
lhz (在线)
老师 (在线)
学生 (离线)
学生1 (在线)

2024-10-18 19:33:50 - lhz has been kicked from the chat by 学生1
lhz has been kicked from the chat by 学生1
lhz has left the chat.
2024-10-18 19:33:52 - lhz has left the chat.

===== 在线用户列表 =====
lhz (离线)
老师 (在线)
学生 (离线)
学生1 (在线)

D:\vs2022\client\x64\Debug\client.exe
输入消息 ('q'退出, '/upload <文件路径>'上传文件): /status

[服务器]
===== 在线用户列表 =====
老师 (在线)
lhz (在线)
学生 (在线)
学生1 (在线)

输入消息 ('q'退出, '/upload <文件路径>'上传文件): 你们好

[服务器]
时间: Fri Oct 18 19:32:20 2024
用户: 学生1 (204.204.204.204)
said: 你们好
输入消息 ('q'退出, '/upload <文件路径>'上传文件): /kick 学生
输入消息 ('q'退出, '/upload <文件路径>'上传文件):

[服务器]
时间: Fri Oct 18 19:33:14 2024
用户: 学生 (204.204.204.204)
said: 12
输入消息 ('q'退出, '/upload <文件路径>'上传文件): /kick lhz
输入消息 ('q'退出, '/upload <文件路径>'上传文件):

```

图 4.9: 强制指定用户下线

当服务端接收到用户强制指定用户下线的命令时，首先从消息中提取出要踢出的目标用户名 `targetUsername`。然后加锁保护 `clients` 容器的访问，防止多线程访问冲突。接着遍历 `clients` 容器，查找目标用户：如果目标用户在线，则将其在线状态设为 `false`，记录日志，并关闭其对应的 `socket`。如果目标用户已经离线，则向客户端发送错误消息。然后更新用户状态映射 `userStatusMap`。最后释放锁，继续等待下一条消息。

```

1  if (message.substr(0, KICK_COMMAND.size()) == KICK_COMMAND) {
2      // Parse the kick command
3      string targetUsername = message.substr(KICK_COMMAND.size() + 1);
4      // Kick the user
5      lock_guard<mutex> lock(clientsMutex);
6      auto it = clients.begin(); // 使用迭代器遍历
7      while (it != clients.end()) {
8          if ((*it)->username == targetUsername) {
9              if ((*it)->isOnline) {
10                 (*it)->isOnline = false;
11                 logMessage(targetUsername +
12                     " has been kicked from the chat by " + username, 12);
13                 cout << targetUsername + " has been kicked from the chat by "
14                     + username << endl;
15                 // Check if the socket is still valid before closing it
16                 if ((*it)->sockfd != INVALID_SOCKET) {
17                     if (closesocket((*it)->sockfd) == SOCKET_ERROR) {
18                         logMessage("Failed to close socket for user " +

```

```

19         targetUsername, 12);
20     }
21 }
22     userStatusMap[targetUsername] = false;
23 }
24     else {
25         string errorMsg = "用户 " + targetUsername + " 已经离线。\\n";
26         send(clientSocket, errorMsg.c_str(), errorMsg.length(), 0);
27     }
28     break;
29 }
30     else {
31         ++it;
32     }
33 }
34     continue;
35 }

```

#### 4.7 清空客户端历史消息

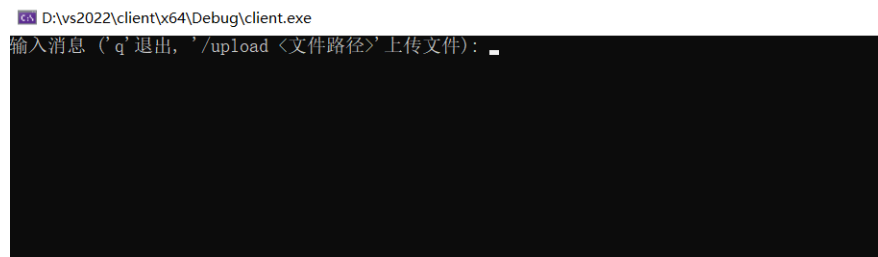


图 4.10: 清空客户端历史消息

该命令由客户端进行处理，当客户端收到/clear 指令后，会调用 clearScreen 函数来清空客户端的屏幕或控制台。然后输出一条提示信息，告知用户可以继续输入消息 ('q' 退出, '/upload < 文件路径 >' 上传文件)。接着继续等待下一条用户输入的消息。

```

1     if (message == "/clear")
2     {
3         clearScreen();
4         cout << "输入消息 ('q'退出, '/upload < 文件路径 >'上传文件): ";
5         continue;
6     }

```

## 4.8 查看历史聊天记录

```

输入消息 ('q' 退出, '/upload <文件路径>' 上传文件): /load
[服务器]
2024-10-18 19:05:31 - Server started on port 8080.
输入消息 ('q' 退出, '/upload <文件路径>' 上传文件):
[服务器]
2024-10-18 19:05:55 - 老师 (204.204.204.204) has joined the chat.
2024-10-18 19:06:06 - lhz (204.204.204.204) has joined the chat.
2024-10-18 19:06:14 - 学生 (204.204.204.204) has joined the chat.
2024-10-18 19:06:22 - 学生1 (204.204.204.204) has joined the chat.
2024-10-18 19:09:08 - Broadcast: 时间: Fri Oct 18 19:09:08 2024
用户: lhz (204.204.204.204)
said: hello
2024-10-18 19:09:17 - Broadcast: 时间: Fri Oct 18 19:09:17 2024
用户: lhz (204.204.204.204)
said: 你们好
2024-10-18 19:09:28 - Broadcast: 时间: Fri Oct 18 19:09:28 2024
用户: lhz (204.204.204.204)
said: adasadsad
this is a test file!!!
welcome to the chat!
欢迎加入!!
输入消息 ('q' 退出, '/upload <文件路径>' 上传文件):

```

图 4.11: 查看历史聊天记录

多人聊天程序的聊天记录会被保存到服务端本地的 `chat_log.txt` 的聊天记录文件中，并且当服务端启动时，会自动清空该文件的所有内容。当服务端接收到用户查看历史聊天记录的命令时，服务端会打开 `chat_log.txt`。如果文件打开成功，服务端会逐行读取文件内容，并通过 `send` 函数将每行内容发送给客户端。在每行内容之后，还会发送一个换行符。如果文件无法打开，服务端会构建一条错误消息 `errorMessage`，并将其发送给客户端。发送完毕后，服务端继续等待下一条来自客户端的消息。并且该信息只会在请求该命令的客户端的界面上进行显示，不会像聊天信息一样广播给其他用户或显示在公屏。具体代码实现如下所示：

---

```

1  if (message.substr(0, LOAD_COMMAND.size()) == LOAD_COMMAND) {
2      // Load the chat log and send it to the requesting client
3      ifstream logFile("chat_log.txt");
4      if (logFile.is_open()) {
5          string line;
6          while (getline(logFile, line)) {
7              send(clientSocket, line.c_str(), line.length(), 0);
8              send(clientSocket, "\n", 1, 0); // Send newline character
9          }
10         logFile.close();
11     }
12     else {
13         string errorMessage = "Failed to open chat log file.\n";
14         send(clientSocket, errorMessage.c_str(), errorMessage.length(), 0);
15     }
16     continue;
17 }

```

---

## 4.9 将指定用户添加到朋友圈

```
[服务器]
===== 老师 的好友列表 =====
- lhz
=====

输入消息 ('q' 退出, '/upload <文件路径>' 上传文件): /addfrd 学生
输入消息 ('q' 退出, '/upload <文件路径>' 上传文件):
User '学生' has been added to your friendlist.
输入消息 ('q' 退出, '/upload <文件路径>' 上传文件):
[服务器]

===== 老师 的好友列表 =====
- lhz
- 学生
=====

输入消息 ('q' 退出, '/upload <文件路径>' 上传文件): /addfrd 学生1
输入消息 ('q' 退出, '/upload <文件路径>' 上传文件):
User '学生1' has been added to your friendlist.
输入消息 ('q' 退出, '/upload <文件路径>' 上传文件):
[服务器]

===== 老师 的好友列表 =====
- lhz
- 学生
- 学生1
=====

输入消息 ('q' 退出, '/upload <文件路径>' 上传文件):
```

图 4.12: 将指定用户添加到朋友圈

当服务端接收到用户将指定用户添加到朋友圈的命令时，首先服务端需要获取发起添加好友请求的客户端用户名 `requesterUsername`，然后调用 `getUsernameFromSocket` 函数，根据客户端的 `sockfd` 找到对应的用户名。接着服务端从消息中提取出要添加的好友用户名 `friendUsername`。接下来，服务端调用 `handleAddFriendRequest` 函数来处理这个添加好友的请求：它首先在 `clients` 容器中查找请求者 `requester` 和目标好友 `friendUser`。如果都找到了，则将好友用户名添加到请求者的 `friendsList` 中。添加成功后，服务端构建一条格式化的提示消息 `formattedMessage` (格式如 “User ‘张三’ has been added to your friendlist.”) 并发送给请求者。最后，服务端调用 `printFriendsList` 函数获取请求者当前的好友列表信息 `friendsListMessage`，并将其发送给请求者。并且该信息只会在请求该命令的客户端的界面上进行显示，不会像聊天信息一样广播给其他用户或显示在公屏。具体代码实现如下所示：

```
1 // 获取与指定套接字关联的用户名
2 string getUsernameFromSocket(SOCKET sockfd) {
3     for (auto& client : clients) {
4         if (client->sockfd == sockfd) {
5             return client->username;
6         }
7     }
8     return "";
9 }
10 // 处理添加好友请求
11 void handleAddFriendRequest(const string& requesterUsername,
12 const string& friendUsername)
13 {
```



```
14     std::shared_ptr<ClientInfo> requester = nullptr;
15     std::shared_ptr<ClientInfo> friendUser = nullptr;
16     // 使用 for 循环查找请求者
17     for (auto& client : clients) {
18         if (client->username == requesterUsername) {
19             requester = client;
20         }
21         if (client->username == friendUsername) {
22             friendUser = client;
23         }
24         // 一旦找到两者，退出循环
25         if (requester && friendUser) break;
26     }
27     if (!requester) {
28         // 请求者不存在
29         return;
30     }
31     if (!friendUser) {
32         // 好友不存在
33         return;
34     }
35     // 将好友添加到请求者的好友列表
36     requester->friendsList.push_back(friendUsername);
37 }
38 if (message.substr(0, ADDFRDCOMMAND.size()) == ADDFRDCOMMAND) {
39     string requesterUsername = getUsernameFromSocket(clientSocket);
40     string friendUsername = message.substr(8); // 获取好友用户名
41     handleAddFriendRequest(requesterUsername, friendUsername);
42     // 提示用户添加成功
43     string formattedMessage = "User '" + friendUsername + "'
44     has been added to your friendlist.";
45     send(clientSocket, formattedMessage.c_str(), formattedMessage.length(), 0);
46     // 调用 printFriendsList 并获取返回的消息
47     std::string friendsListMessage = printFriendsList(requesterUsername);
48     // 将好友列表发送给请求者
49     send(clientSocket, friendsListMessage.c_str(), friendsListMessage.length(), 0);
50     continue;
51 }
```

#### 4.10 将指定用户添加到黑名单

```

输入消息 ('q'退出, '/upload <文件路径>'上传文件): /addsb lhz
输入消息 ('q'退出, '/upload <文件路径>'上传文件):
User 'lhz' has been added to your blacklist.
输入消息 ('q'退出, '/upload <文件路径>'上传文件):
[服务器]

===== 学生 的黑名单 =====
- lhz
=====

输入消息 ('q'退出, '/upload <文件路径>'上传文件): /addsb 学生1
输入消息 ('q'退出, '/upload <文件路径>'上传文件):
User '学生1' has been added to your blacklist.
输入消息 ('q'退出, '/upload <文件路径>'上传文件):
[服务器]

===== 学生 的黑名单 =====
- lhz
- 学生1
=====

输入消息 ('q'退出, '/upload <文件路径>'上传文件):

```

图 4.13: 将指定用户添加到黑名单

逻辑与添加指定用户到朋友圈类似，当服务端接收到用户将指定用户添加到黑名单的命令时，调用 `getUsernameFromSocket` 函数获取发起请求的用户名 `requesterUsername`。并且从消息中提取出要添加到黑名单的用户名 `sbUsername`。接着调用 `handleAddSbRequest` 函数来处理这个添加黑名单的请求（这个函数的具体实现与前面的 `handleAddFriendRequest` 类似）。然后构建一条提示信息 `formattedMessage`，告知用户添加成功，并且将 `formattedMessage` 通过 `send` 函数发送给请求者。最后调用 `printblackList` 函数获取请求者当前的黑名单列表消息 `blackListMessage`，并将其发送给请求者。然后继续等待下一条来自客户端的消息。并且该信息只会在请求该命令的客户端的界面上进行显示，不会像聊天信息一样广播给其他用户或显示在公屏。具体代码实现如下所示：

---

```

1  if (message.substr(0, ADDSBCOMMAND.size()) == ADDSBCOMMAND) {
2      string requesterUsername = getUsernameFromSocket(clientSocket);
3      string sbUsername = message.substr(7); // 获取黑名单用户名
4      handleAddSbRequest(requesterUsername, sbUsername);
5      // 提示用户添加成功
6      string formattedMessage = "User '" + sbUsername +
7      "' has been added to your blacklist.";
8      send(clientSocket, formattedMessage.c_str(), formattedMessage.length(), 0);
9      std::string blackListMessage = printblackList(requesterUsername);
10     // 将好友列表发送给请求者
11     send(clientSocket, blackListMessage.c_str(), blackListMessage.length(), 0);
12     continue;
13 }

```

---

## 5 实验内容总结

本次实验实现了一个基本的聊天系统，包含了以下几个主要功能：

- 基本的聊天功能：客户端能够发送文本消息，服务端收到消息后能够正确识别并广播给其他在线用户。服务端还会将消息记录到聊天记录文件中。
- 文件上传功能：客户端能够发送文件，服务端收到文件后能够正确保存并通知其他在线用户有新文件上传。服务端还会将文件上传信息记录到聊天记录文件中。
- 获取当前多人聊天程序中所有用户状态：客户端能够发送查询在线用户状态的请求，服务端收到请求后能够遍历当前客户端列表，并将在线用户信息反馈给请求的客户端。
- 将指定用户添加到朋友圈：服务端能够正确识别客户端的添加好友请求，并维护用户的好友关系，同时将变更信息反馈给客户端。
- 将指定用户添加到黑名单：服务端能够正确处理客户端的添加黑名单请求，并及时更新和反馈用户的黑名单信息。
- 查看历史聊天记录：服务端能够从聊天记录文件中读取历史消息内容，并发送给查看请求的客户端。
- 清空客户端历史消息：客户端能够正确识别并响应用户的清屏指令，及时清空聊天窗口的历史记录。
- 强制指定用户下线：服务端能够准确识别并处理管理员或其他用户的强制下线请求，安全地将目标用户踢出聊天系统。

通过这些功能的实现，我们可以看到该聊天系统具有较完善的用户管理和聊天记录管理能力。服务端能够及时响应客户端的各种操作请求，并采取相应的措施，既保证了系统的正常运行，也提高了用户的使用体验。

在实现过程中，我注意到以下几个关键点：

- 服务端需要维护一个客户端列表 `clients`，用于管理在线用户的状态和信息。并配合使用互斥锁 `clientsMutex` 来保证线程安全。
- 服务端需要能够准确识别和解析客户端发来的各种命令，并针对不同的命令采取相应的处理逻辑。
- 服务端需要能够记录聊天日志，并能从日志文件中读取历史消息内容，反馈给客户端。
- 客户端需要能够正确响应用户的清屏指令，及时清空聊天窗口的历史记录。

总的来说，此次实验涉及到了客户端-服务端架构、用户管理、聊天记录管理等多个方面的内容，体现了一个基本聊天系统的设计思路。通过实践使我加深了对这些概念和技术的理解，为今后计算机网络课程的进一步学习奠定了基础。