南开大学
Nankai University

南　　开　　大　　学

计算机学院与网络空间安全学院

计算机网络实验报告

第一次实验：多人聊天系统的设计

罗瑞 2210529

年级：2022 级

专业：密码科学与技术

2024 年 10 月 20 日

目录

一、 实验要求

1. **聊天协议的完整说明**
   - 详细描述聊天协议的各个部分和工作原理。
2. **程序编写要求**
   - 使用 C 或 C++ 语言。
   - 使用基本的 Socket 函数，不允许使用封装后的类（如 CSocket）。
3. **程序实现要求**
   - 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，可以不是图形界面。
   - 程序应该有正常的退出方式。
4. **功能要求**
   - 支持多人聊天。
   - 支持英文和中文聊天。
5. **代码质量要求**
   - 程序结构清晰。
   - 具有较好的可读性。
6. **实验观察与提交**
   - 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、 协议设计

客户端和服务器之间通过 TCP 套接字进行通信，采用的消息协议包括时间戳和消息内容的组合。为了明确通信协议的规范性，我从消息类型、语法、语义和时序四个方面对协议进行详细说明。

另外，因为本聊天系统是基于 TCP 协议的，因此所有用户的消息都需要通过服务器处理并转发，因此以下对协议的说明都是当客户端成功连接服务器之后，客户端与服务器之间的通信协议。

(一) 消息类型

通信协议包括以下几种类型的消息：

- 群聊消息：用户发送的普通聊天信息，所有在线的用户都可以接收和显示。
- 指令系统：私聊消息：通过\msg 命令，发送给指定用户的消息。

- 指令系统：退出聊天：通过\exit 命令，用于退出聊天。
- 系统消息：服务器通知、错误信息、用户连接或断开的提示信息。这类消息由服务器生成并广播给所有用户，或发送给指定用户。

(二) 语法

每条消息包括两个部分：

- 时间戳：每条消息都附带发送的时间戳，表示消息发送的时刻。
- 消息内容：包括普通消息、私聊消息以及系统命令。用户发送的消息语法如下：
  - 用户发送消息输入格式：

    消息内容

  - 用户私聊发送输入格式：

    ```
    \msg target_user_name 消息内容
    ```

  - 退出命令格式：

    ```
    \exit
    ```

这样设计保证了在具有特殊功能的前提下不影响任何类型的正常消息的发送。比如消息 exit 就与\exit 成功区分开来，只有当消息以斜杠开头时，才会从指令字典从查找有没有对应的指令以及语法的正确性，否则消息发送失败，这样就保证了功能具体的完整性。

(三) 语义

- 时间戳：提供发送时的日期和时间，用于显示消息的发送时间，方便用户按时间顺序查看消息。
- 群聊消息：发送给所有在线用户，用于群组中的公共交流。
- 私聊消息：发送给指定用户的私人消息，仅接收者能够看到。
- 系统命令：
  - \exit：通知服务器当前用户退出，服务器可以关闭与该用户的连接。

(四) 时序

客户端与服务器之间的通信是基于时序的，通过先发时间戳再发消息内容的原则进行。

- 发送顺序：
  - 先发送时间戳：客户端或服务器先将时间戳发送给对方，表示消息的发送时间。
  - 发送消息内容：紧接着，消息内容（包括群聊消息、私聊消息或系统命令）将被发送。
- 消息发送时序：

- 群聊消息时序：
  - 客户端发送时间戳（20 字节）。
  - 客户端发送消息内容（128 字节）。
- 私聊消息时序：
  - 客户端发送时间戳（20 字节）。
  - 客户端发送私聊消息内容（128 字节，包含用户名和消息）。
- 退出命令时序：
  - 客户端发送时间戳（20 字节）。
  - 客户端发送\exit 命令（占用 128 字节的消息缓冲区）。

三、 程序模块功能与运行界面展示

(一) 程序功能模块说明：服务器端

服务器能够处理客户端连接、广播消息、处理私聊命令，以及记录日志等功能。它使用了 Winsock 库来实现网络通信，通过线程来管理每个客户端的消息处理。关键功能模块包括消息收发、日志记录、私聊功能以及退出命令的监控。

核心功能模块及代码解释：

- **CLIENT 结构体**：

CLIENT 结构体

```
struct CLIENT {
    SOCKET client;          // 客户端套接字
    char username[32];      // 用户名，最大长度为32字符
    char buffer[256];       // 信息缓冲区
    int index;              // 客户端在连接列表中的索引
    int IP;                 // 客户端IP地址
};
```

这个结构体用于保存每个客户端的相关信息。每个连接的客户端会有一个套接字 (client)、用户名 (username)、消息缓冲区 (buffer)、连接索引 (index) 和 IP 地址 (IP)。注意，我设置 client 结构体数组的上限是 20，这里就说明了最多支持 20 人次的多人聊天。

- **日志记录函数 (logMessage)**：

日志记录函数

```
void logMessage(const char* message) {
    ofstream logFile("server_log.txt", ios::app);
    if (logFile.is_open()) {
        char timeBuffer[20];
        getTime(timeBuffer, sizeof(timeBuffer));
        logFile << timeBuffer << " - " << message << endl;
        logFile.close();
    }
}
```

这个函数用于将消息记录到日志文件中。首先获取当前时间，并将时间和消息以追加模式写入日志文件。日志记录可以帮助服务器管理员追踪客户端的行为，例如加入、退出和私聊信息。

- **获取当前时间函数 (getTime)**:

获取当前时间函数

```
void getTime(char* buffer, size_t size) {
    time_t timep;
    time(&timep);
    struct tm timeInfo;
    localtime_s(&timeInfo, &timep);
    strftime(buffer, size, "%Y-%m-%d %H:%M:%S", &timeInfo);
}
```

该函数获取当前的系统时间，并将其格式化为 YYYY-MM-DD HH:MM:SS 的形式存入传入的字符缓冲区中。它用于标记每条日志消息的时间戳。

- **消息处理线程 (CommunicateThread)**:

消息处理线程

```
DWORD WINAPI CommunicateThread(LPVOID lparam) {
    CLIENT* Client = (CLIENT*)lparam;
    char revData[128];
    char recv_time[20] = { 0 };
    char full_message[256];

    while (true) {
        int ret = recv(Client->client, revData, 128, 0);
        if (ret > 0) {
            // 处理退出命令
            if (strcmp(revData, "\\exit") == 0) {
                // 处理客户端退出
            }

            // 处理私聊命令
            if (strncmp(revData, "\\msg ", 5) == 0) {
                // 处理私聊消息
            }

            // 将消息广播给其他客户端
            for (int i = 0; i < num; i++) {
                if (i != Client->index) {
                    send(chat_array[i].client, full_message, strlen(
                        full_message), 0);
                }
            }
        }
    }
```

```
28      }
```

每个客户端连接后，服务器会为其创建一个独立的线程负责接收和处理消息。线程持续运行，接收来自客户端的消息，并根据消息类型（普通消息或私聊）进行相应处理。普通消息会被广播给其他客户端，而私聊消息会根据特定格式发送给指定的用户。

- **处理私聊命令**:

处理私聊命令

```
1   if (strncmp(revData, "\\msg␣", 5) == 0) {
2       // 提取目标用户和私聊消息内容
3       string targetUser = ...;
4       string privateMessage = ...;
5
6       // 查找目标用户并发送私聊消息
7       bool userFound = false;
8       for (int i = 0; i < num; i++) {
9           if (strcmp(chat_array[i].username, targetUser.c_str()) == 0) {
10              send(chat_array[i].client, fullMessage.c_str(), fullMessage.
                    length(), 0);
11              userFound = true;
12              break;
13          }
14      }
15
16      if (!userFound) {
17          string errorMessage = "用户␣" + targetUser + "␣不在线或不存在。";
18          send(Client->client, errorMessage.c_str(), errorMessage.length(),
                0);
19      }
20  }
```

当收到 \msg 私聊命令时，服务器首先解析消息，提取目标用户和消息内容。然后遍历所有已连接的客户端，找到目标用户并发送私聊消息。如果目标用户不存在或不在线，则发送错误消息给发送者。

- **退出命令的监控 (checkExitCommand)**:

退出命令的监控

```
1   void checkExitCommand() {
2       string command;
3       while (serverRunning) {
4           cin >> command;
5           if (command == "exit") {
6               serverRunning = false;
7               logMessage("服务器退出");
8               exit(0);  // 直接退出程序
9           }
10      }
```

```
    }
```

这是一个独立的线程函数，用于监控服务器管理员输入的命令。输入 `exit` 时，服务器会停止运行，并记录服务器退出的日志。这使得服务器可以在运行期间被安全地关闭。

- **主函数 (main)**:

主函数

```
int main() {
    WORD sockVersion = MAKEWORD(2, 2);
    WSADATA wsaData;
    if (WSAStartup(sockVersion, &wsaData) != 0) {
        return 0;
    }

    SOCKET Srv = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    sockaddr_in sin;
    sin.sin_family = AF_INET;
    sin.sin_port = htons(49711);
    sin.sin_addr.S_un.S_addr = INADDR_ANY;

    bind(Srv, (LPSOCKADDR)&sin, sizeof(sin));
    listen(Srv, 2);

    // 启动监控退出命令的线程
    thread exitThread(checkExitCommand);

    while (serverRunning) {
        chat_array[num].client = accept(Srv, (SOCKADDR*)&ClientAddr, &len
            );
        recv(chat_array[num].client, chat_array[num].username, sizeof(
            chat_array[num].username), 0);

        // 创建与客户端通信的线程
        CreateThread(NULL, NULL, CommunicateThread, &chat_array[num], 0,
            &dwThreadId);

        num++;
    }

    closesocket(Srv);
    WSACleanup();
}
```

服务器的主函数负责初始化 Winsock，创建套接字并绑定到指定端口（49711），然后进入监听状态，等待客户端的连接。每当有新的客户端连接时，服务器会接收该客户端的用户名，并为其创建新的线程处理消息通信。这里实现的多线程使得不同用户的消息往来不会相互干扰。

(二) 程序功能模块说明：客户端

1. 主函数逻辑解释

- **Winsock 初始化**

初始化

```
WORD verRequested = MAKEWORD(2, 2);
WSADATA wsaData;
WSAStartup(verRequested, &wsaData);
```

功能：初始化 Windows 套接字库 (Winsock)，使其可以在程序中使用网络功能。

  - `MAKEWORD(2, 2)`：表示请求 Winsock 版本 2.2。
  - `WSAStartup`：初始化 Winsock 库，并将 `wsaData` 用作 Winsock 的相关信息存储。

- **创建客户端套接字**

创建套接字

```
SOCKET clientSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
```

功能：创建一个 TCP 客户端套接字，用于连接到服务器。

  - `AF_INET`：表示使用 IPv4 地址。
  - `SOCK_STREAM`：表示使用面向连接的 TCP 协议。
  - `IPPROTO_TCP`：指定协议为 TCP。

如果 `socket` 返回 `INVALID_SOCKET`，则说明套接字创建失败。

- **配置服务器地址信息**

配置服务器地址信息

```
sockaddr_in serverAddr;
serverAddr.sin_family = AF_INET;
serverAddr.sin_port = htons(49711);
serverAddr.sin_addr.S_un.S_addr = inet_addr("127.0.0.1");
```

功能：设置服务器的 IP 地址和端口号。

  - `sin_family`：指定地址族为 `AF_INET`（IPv4）。
  - `htons(49711)`：将端口号 49711 转换为网络字节顺序（大端）。
  - `inet_addr("127.0.0.1")`：将 IP 地址”127.0.0.1” 转换为网络字节顺序（服务器为本地回环地址）。

- **连接服务器**

连接服务器

```
if (connect(clientSocket, (SOCKADDR*)&serverAddr, sizeof(serverAddr)) ==
    SOCKET_ERROR) {
    cout << "连接服务器失败" << endl;
```

```
        cout << WSAGetLastError() << endl;
        return 0;
    }
```

功能：尝试连接到指定的服务器。

- `connect`：将客户端的套接字连接到指定的服务器地址 `serverAddr`。

如果连接失败，返回 `SOCKET_ERROR`，并使用 `WSAGetLastError` 获取错误码，打印失败信息后终止程序。

- **提示用户输入昵称**

提示用户输入昵称

```
    cout << "成功连接服务器" << endl;
    cout << "请输入您的昵称加入群聊: ";
    gets_s(nickname);  // 获取用户昵称
    send(clientSocket, nickname, sizeof(nickname), 0);  // 发送昵称到服务器
```

功能：提示用户输入昵称，并将输入的昵称发送给服务器。

- `gets_s(nickname)`：使用 `gets_s` 获取用户输入的昵称并保存到 `nickname` 字符数组中。
- `send`：将昵称发送给服务器，服务器会用这个昵称来识别用户。

- **创建接收和发送消息的线程**

创建接收和发送消息的线程

```
    HANDLE recvThread = CreateThread(NULL, 0, ReceiveMessages, (LPVOID)
        clientSocket, 0, NULL);
    HANDLE sendThread = CreateThread(NULL, 0, SendMessages, (LPVOID)
        clientSocket, 0, NULL);
```

功能：创建两个独立的线程，分别用于处理消息的接收和发送。

- `CreateThread`：
  - 第一个线程 `recvThread` 负责从服务器接收消息，运行 `ReceiveMessages` 函数。
  - 第二个线程 `sendThread` 负责向服务器发送消息，运行 `SendMessages` 函数。
- `clientSocket` 作为参数传递给两个线程，用于与服务器的通信。

- **等待线程结束**

线程逻辑

```
    WaitForSingleObject(recvThread, INFINITE);
    WaitForSingleObject(sendThread, INFINITE);
```

功能：主线程等待两个子线程的执行结束。

- `WaitForSingleObject`：阻塞主线程，直到指定的线程（接收和发送消息的线程）终止。

- **清理资源**

清理资源

```
CloseHandle(recvThread);
CloseHandle(sendThread);
```

功能：关闭线程句柄，释放系统资源。

- **返回 0 结束程序**

结束程序

```
return 0;
```

功能：程序执行结束，返回 0 表示正常退出。

总结：

主函数首先初始化 Winsock 库，创建客户端套接字并连接到服务器。然后通过输入昵称加入群聊，接着创建两个独立的线程分别处理接收和发送消息的功能。主函数等待两个线程的结束，并在程序退出时关闭线程句柄，释放系统资源。

2. 线程 a: 用于发送消息

发送消息线程处理

```
DWORD WINAPI SendMessages(LPVOID lParam) {
    SOCKET clientSocket = (SOCKET)(LPVOID)lParam;
    char sendBuffer[128] = { 0 };
    char sendTime[20] = { 0 };

    while (isServerRunning) {
        // 获取当前时间
        time_t currentTime = time(0);
        memset(sendBuffer, 0, sizeof(sendBuffer));  // 清空发送缓冲区
        strftime(sendTime, sizeof(sendTime), "%Y-%m-%d %H:%M:%S", localtime(&
            currentTime));

        // 使用 fgets() 获取用户输入，确保不会溢出
        cout << "请输入消息: ";
        if (fgets(sendBuffer, sizeof(sendBuffer), stdin) == nullptr) {
            cout << "读取输入失败。" << endl;
            continue;  // 如果读取失败，重新开始循环
        }

        // 移除末尾的换行符 '\n'
        size_t len = strlen(sendBuffer);
        if (len > 0 && sendBuffer[len - 1] == '\n') {
            sendBuffer[len - 1] = '\0';
        }

        // 如果输入的消息长度达到缓冲区上限，提示并继续下一轮输入
        if (strlen(sendBuffer) >= sizeof(sendBuffer)) {
```

```
27            cout << "输入消息过长，消息长度应小于␣" << sizeof(sendBuffer) <<
                  "␣字符。" << endl;
28            continue;
29        }
30
31        // 处理命令输入
32        if (sendBuffer[0] == '\\') {
33            // 判断具体命令
34            if (strcmp(sendBuffer, "\\exit") == 0) {
35                isServerRunning = false;  // 标记退出
36                int timeSent = send(clientSocket, sendTime, 20, 0);
37                int messageSent = send(clientSocket, sendBuffer, 128, 0);
38
39                if (messageSent < 0 || timeSent < 0) {
40                    cout << "发送失败！" << endl;
41                    break;
42                }
43                cout << "已退出聊天" << endl;
44                exit(0);  // 结束进程
45            }
46            else if (strncmp(sendBuffer, "\\msg␣", 5) == 0) {  // 私聊命令处
                  理
47                string fullMsg(sendBuffer);
48                size_t firstSpace = fullMsg.find('␣');  // 找到第一个空格
49
50                if (firstSpace != string::npos) {
51                    size_t secondSpace = fullMsg.find('␣', firstSpace + 1);
                          // 找到第二个空格
52
53                    if (secondSpace != string::npos) {
54                        string targetUser = fullMsg.substr(firstSpace + 1,
                              secondSpace - firstSpace - 1);  // 提取用户名
55                        string privateMsg = fullMsg.substr(secondSpace + 1);
                              // 提取消息内容
56
57                        // 检查是否为空
58                        if (targetUser.empty() || privateMsg.empty()) {
59                            cout << "私聊格式错误，用户名或消息不能为空。" <<
                                  endl;
60                        }
61                        else {
62                            // 构造完整私聊消息
63                            string fullMessage = string(sendTime) + "␣[私聊]␣
                                  " + string(nickname) + "␣->␣" + targetUser +
                                  ":␣" + privateMsg;
64                            int timeSent = send(clientSocket, sendTime, 20,
                                  0);
65                            int messageSent = send(clientSocket, sendBuffer,
```

```
                                128, 0);
66                          if (timeSent < 0 || messageSent < 0) {
67                              cout << "私聊消息发送失败! " << endl;
68                          }
69                          else {
70                              cout << "私聊消息已发送! " << endl;
71                          }
72                      }
73                  }
74                  else {
75                      cout << "私聊格式错误, 应为␣\\msg␣用户名␣消息。" <<
                            endl;
76                  }
77              }
78              else {
79                  cout << "私聊格式错误, 应为␣\\msg␣用户名␣消息。" << endl;
80              }
81              continue;  // 继续下一轮输入
82          }
83          else {
84              // 无效命令提示
85              cout << "无效命令:␣" << sendBuffer << ", 未发送。" << endl;
86              continue;  // 跳过发送
87          }
88      }
89
90      printf("%s", sendTime);
91      cout << "消息已发送" << endl;
92      // 发送普通群聊消息
93      int timeSent = send(clientSocket, sendTime, 20, 0);
94      int messageSent = send(clientSocket, sendBuffer, 128, 0);
95
96      if (messageSent < 0 || timeSent < 0) {
97          cout << "发送失败! " << endl;
98          break;
99      }
100 }
101 closesocket(clientSocket);  // 关闭套接字
102 WSACleanup();  // 清理Winsock资源
103 return 0;
104 }
105
106
107   }
```

这个处理消息发送的线程的功能定义比较完整，包括以下几个方面：

- **时间戳生成**：在每条消息发送前，使用 `strftime()` 函数生成当前系统时间，并将其与消息一起发送。

- **消息输入**：使用 `gets()` 函数从控制台读取用户输入的消息。这里已经加入了消息长度的检查（`strlen(sendBuffer)`），确保消息不会超出缓冲区大小（128 字符）。
- **命令解析与处理**：如果消息以 `\` 开头，表示它是一个命令。支持的命令有：
  - `\exit`：退出聊天，结束程序。
  - `\msg`：发送私聊消息，要求格式为 `\msg username message`，并将私聊消息发送给特定用户。
- **消息发送**：通过 `send()` 函数发送系统时间和消息到服务器。如果发送成功，将显示发送成功的提示，否则提示发送失败。

3. 线程 b：用于接收信息

接收消息处理线程

```
  // 接收消息的线程
DWORD WINAPI ReceiveMessages(LPVOID lParam) {
    SOCKET clientSocket = (SOCKET)(LPVOID)lParam;
    char receiveBuffer[256] = { 0 };
    int receiveLen = 0;

    while (isServerRunning) {  // 检查服务器运行状态
        memset(receiveBuffer, 0, sizeof(receiveBuffer));  // 清空缓冲区
        receiveLen = recv(clientSocket, receiveBuffer, 256, 0);
        if (receiveLen <= 0) {  // 检查是否接收失败或连接断开
            cout << "接收消息失败!" << endl;
            isServerRunning = false;  // 停止循环
            break;
        }
        else {
            cout << "<群聊>" << receiveBuffer << endl;
        }
    }
    return 0;
}
  }
```

- `while (isServerRunning)`：只要服务器处于运行状态，线程就会一直循环等待接收消息。
- `memset(receiveBuffer, 0, sizeof(receiveBuffer))`:每次接收前清空接收缓冲区 `receiveBuffer`，确保之前的消息内容不会残留在缓冲区中。
- `recv(clientSocket, receiveBuffer, 256, 0)`：调用 `recv` 函数从服务器接收消息：
  - 第一个参数是客户端的套接字 `clientSocket`。
  - 第二个参数是接收缓冲区 `receiveBuffer`。
  - 第三个参数是接收的最大长度（256 字节）。
  - 第四个参数为标志位，这里传入 0 表示默认接收行为。

该函数会将服务器发送的消息写入到 `receiveBuffer` 中，并返回实际接收到的字节数。

- `if (receiveLen <= 0)`：如果 `recv` 返回值小于或等于 0，表示接收失败或服务器关闭了连接。在这种情况下，打印错误信息” 接收消息失败”，并将 `isServerRunning` 置为 `false`，结束接收循环。
- `else`：如果接收到有效的消息，将其打印到控制台，显示为 < 群聊 > 消息内容。

(三) 程序运行效果展示

运行流程：首先运行 server.exe 开启聊天服务器，再运行程序 CSNet1.2.exe，多次运行客户端程序依次上线用户：小红、小兰、jimmy

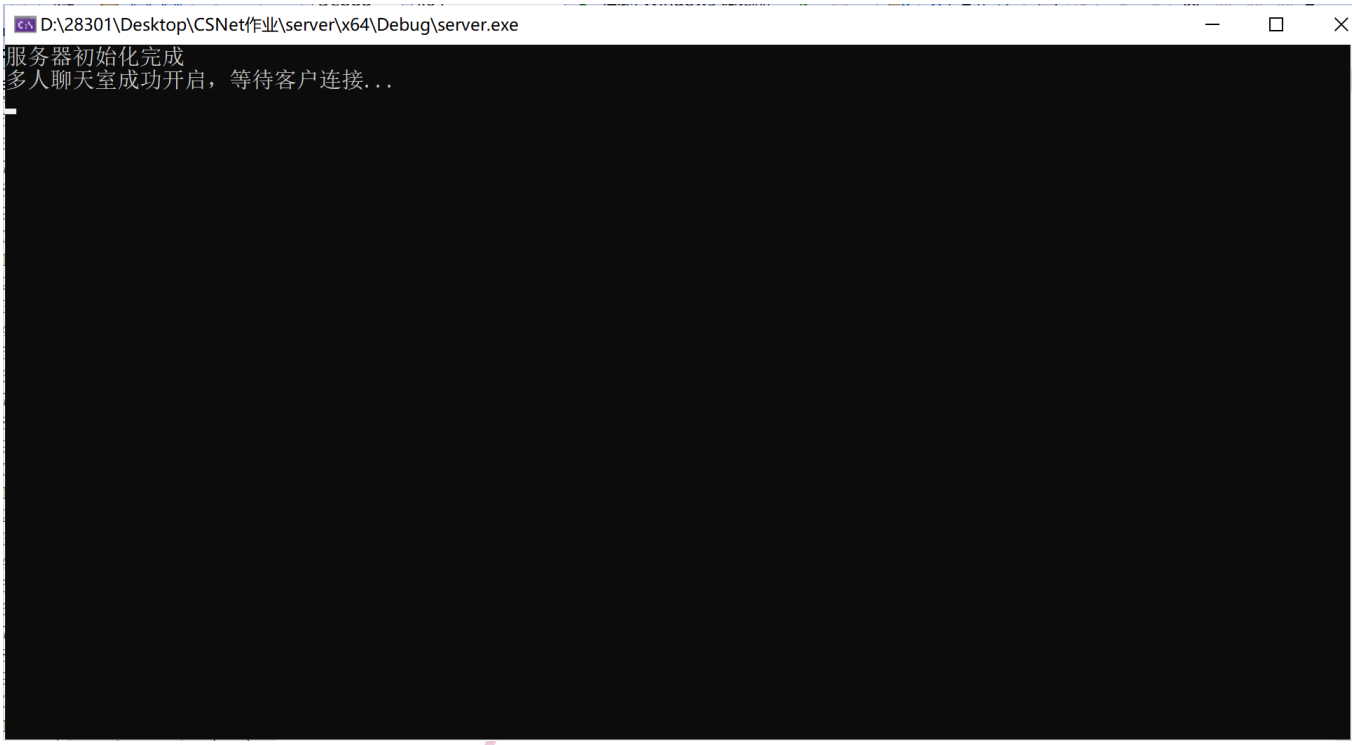


图 1: 服务器启动

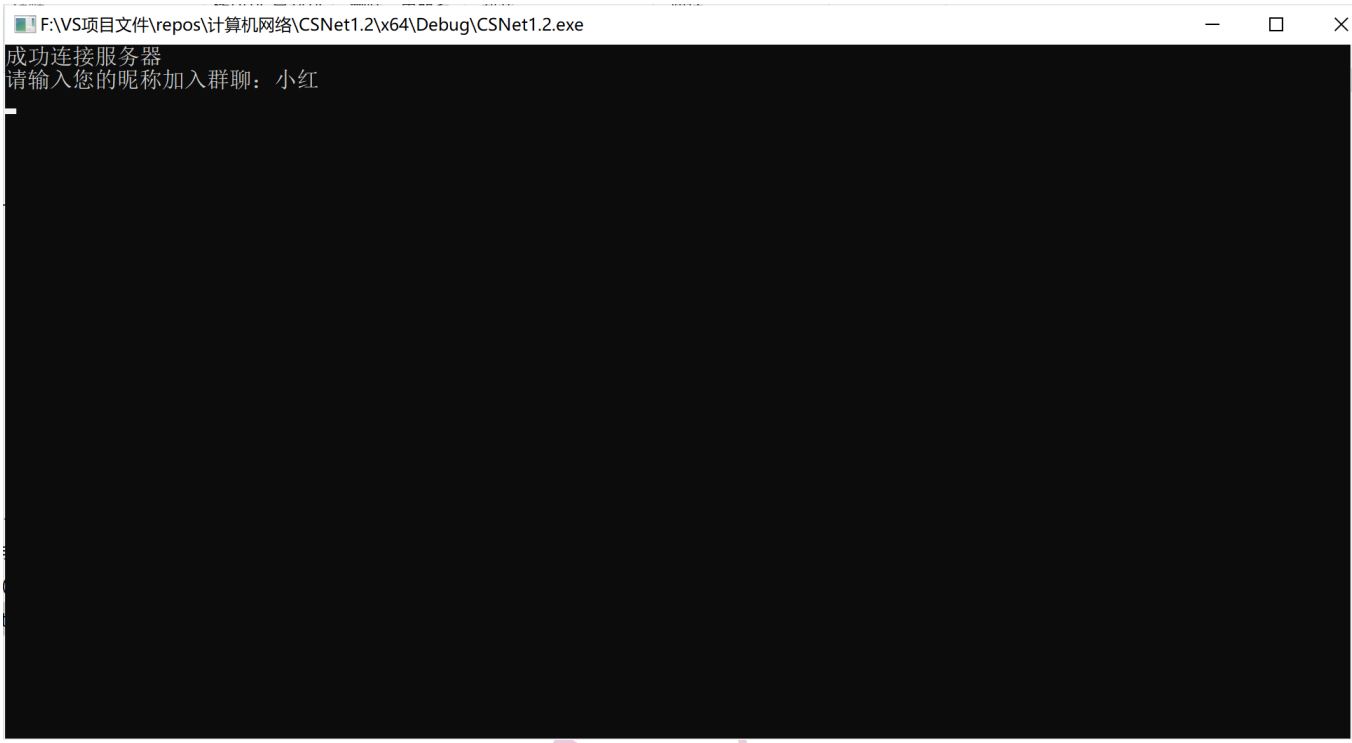


图 2: 小红上线

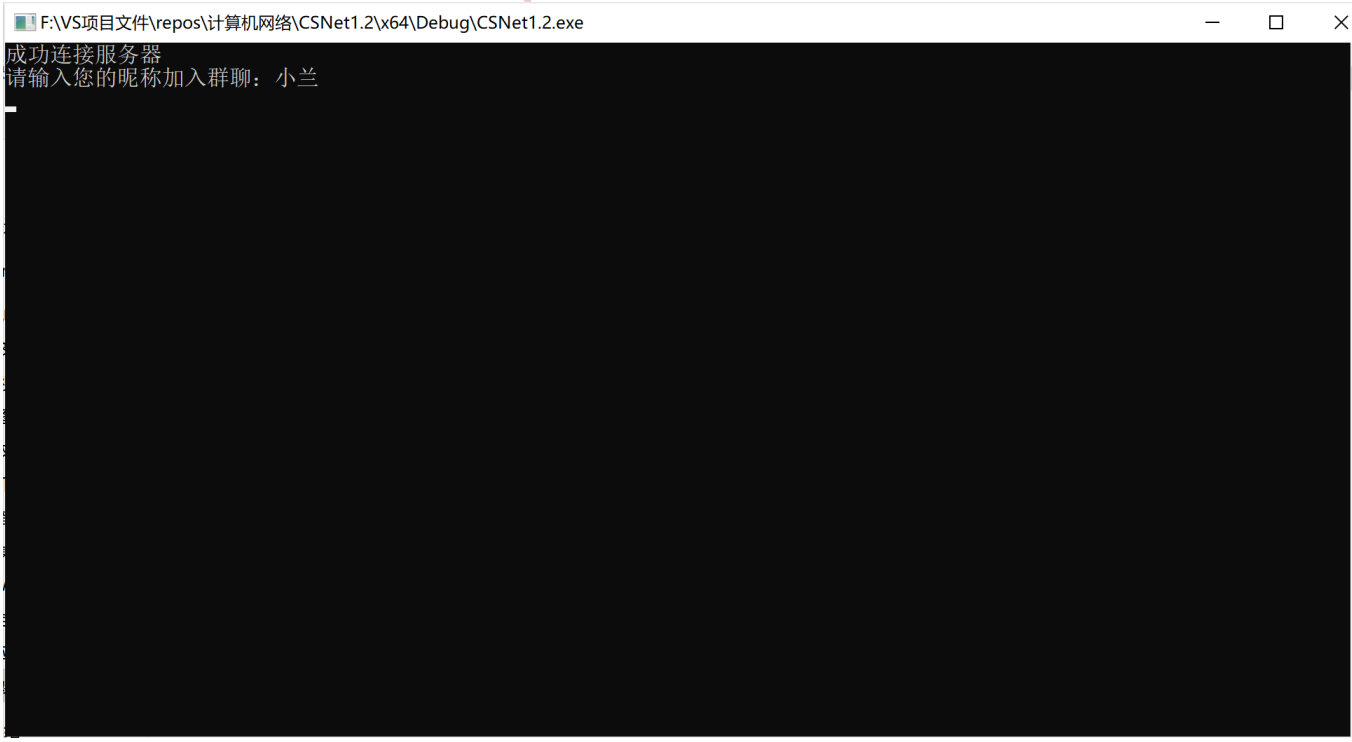


图 3: 小兰上线

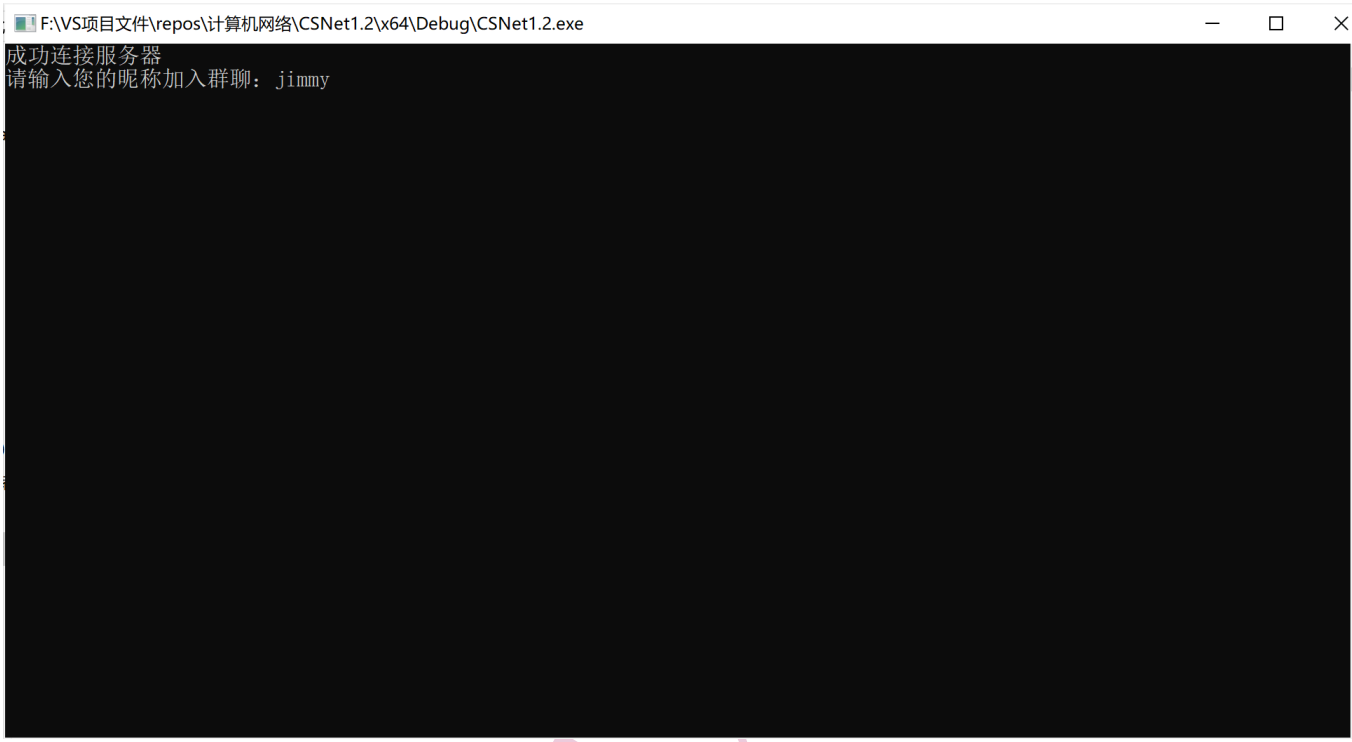


图 4: Jimmy 上线

此时，我们的服务器会依次显示上线的所有信息，以及先上线的人要能看到后上线的”群聊提示”:

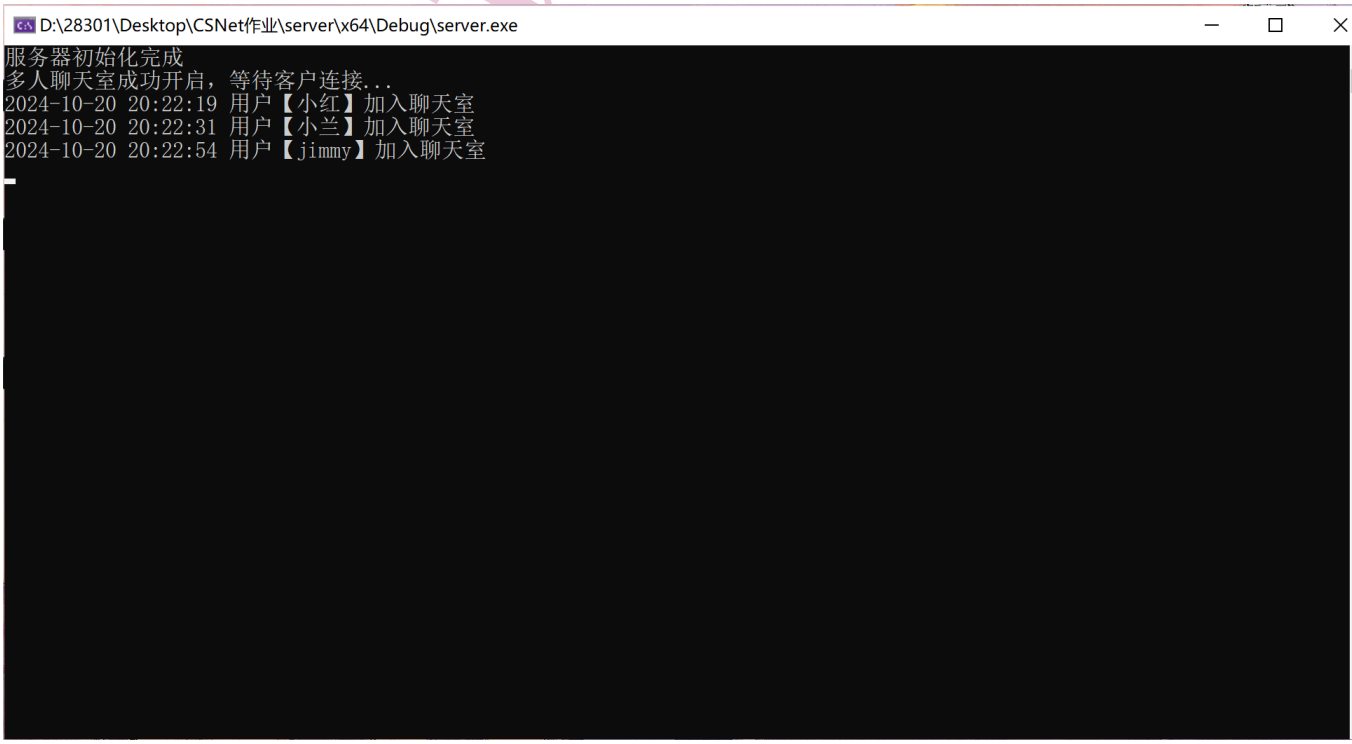


图 5: 服务器显示

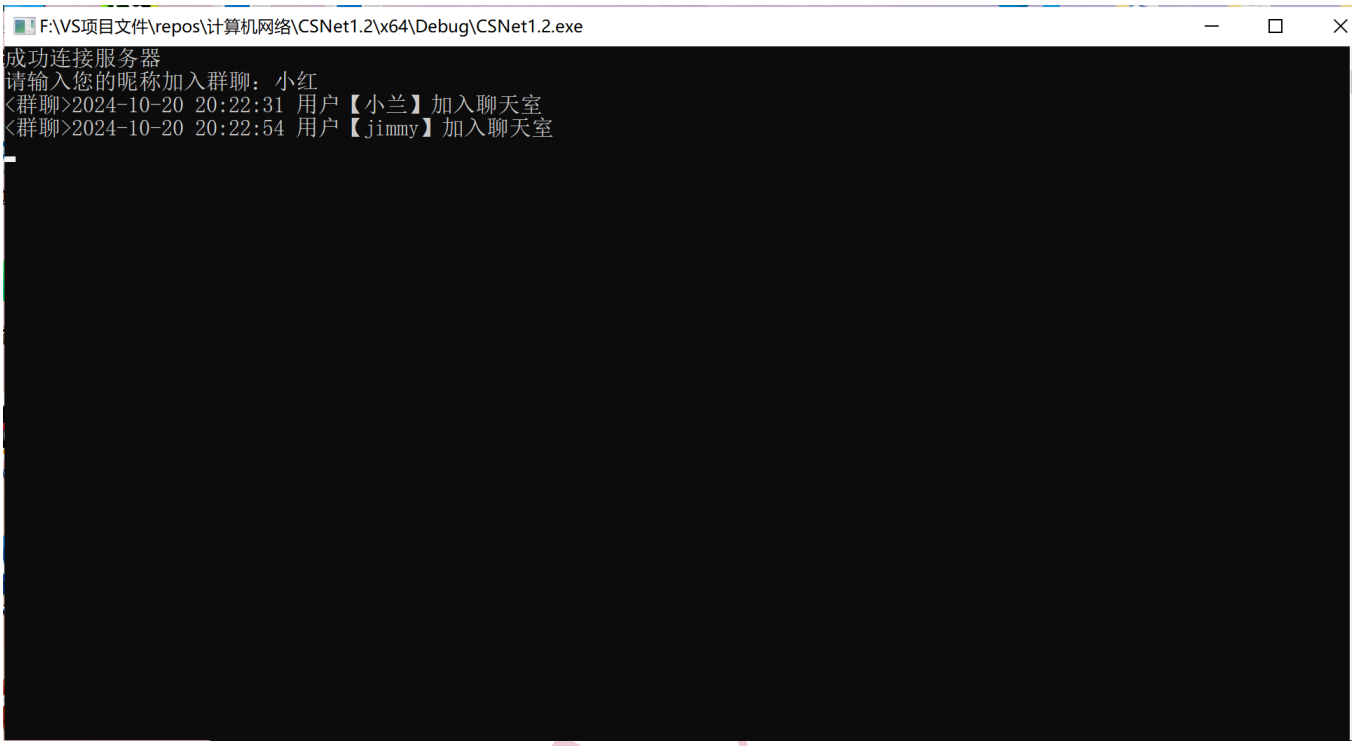


图 6: 小红收到后来上线的消息

现在，小红发送群聊消息：“大家计网作业写完了吗?”，应该所有人都能看到消息的转发。

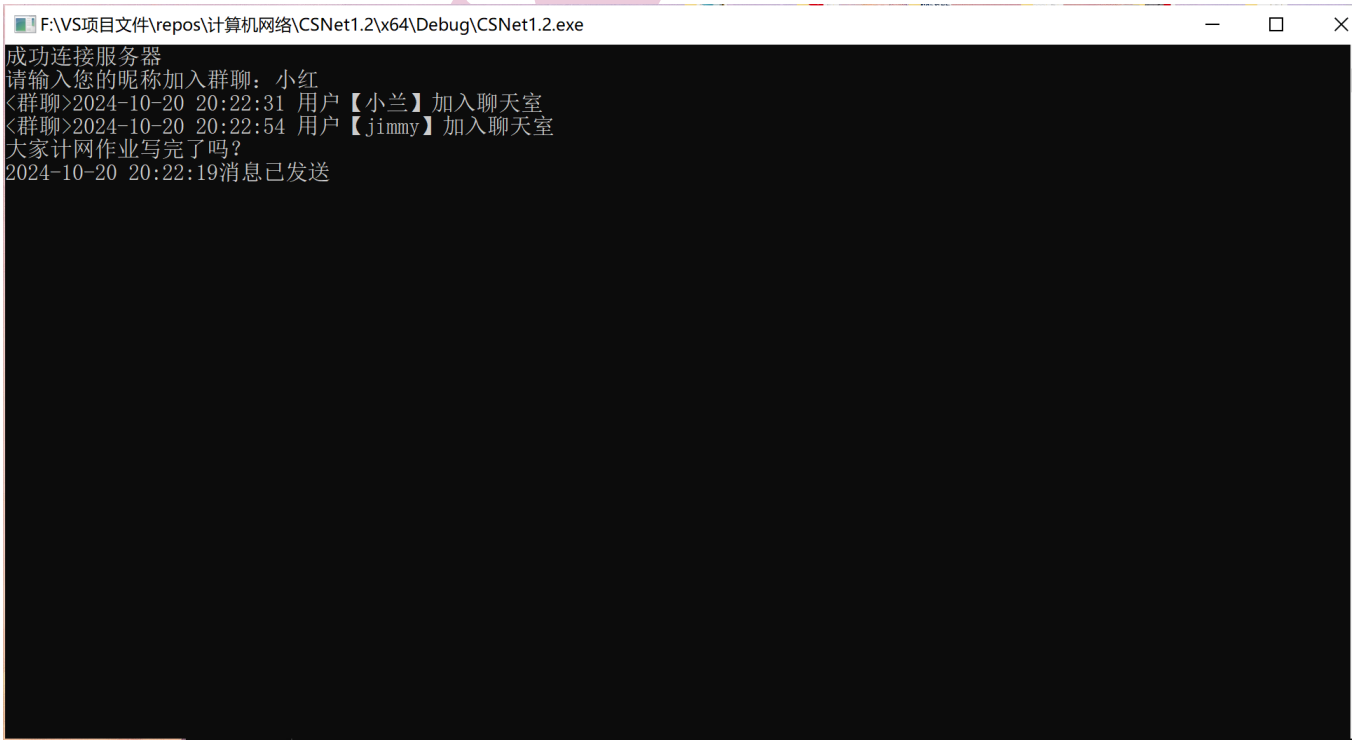


图 7: 小红发群聊消息

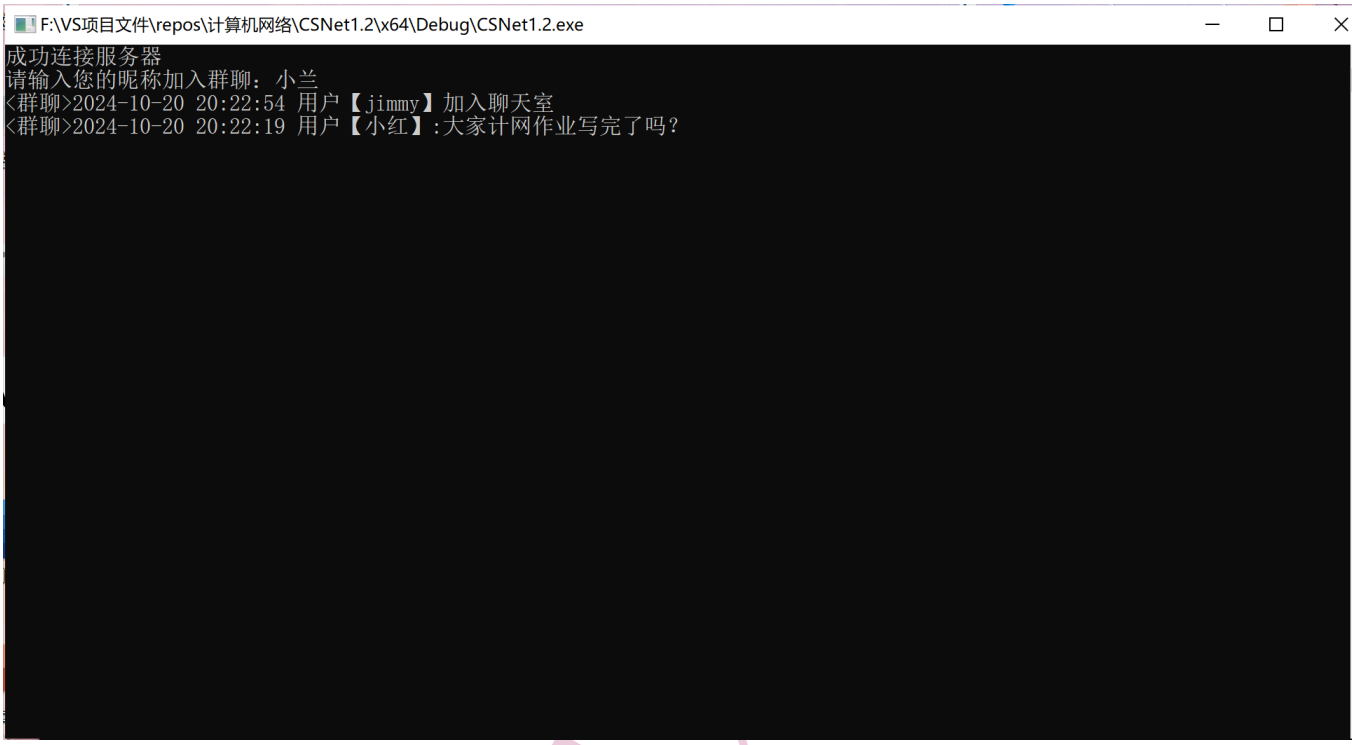


图 8: 小兰收到消息

这时 jimmy 回答一个超过 128 字符长度的消息，我的程序为了防止溢出设置了超限则循环发送（分成多次），但是由于中文的编码问题，超限的部分就不能正常显示了，但是程序是没有出现崩溃的。

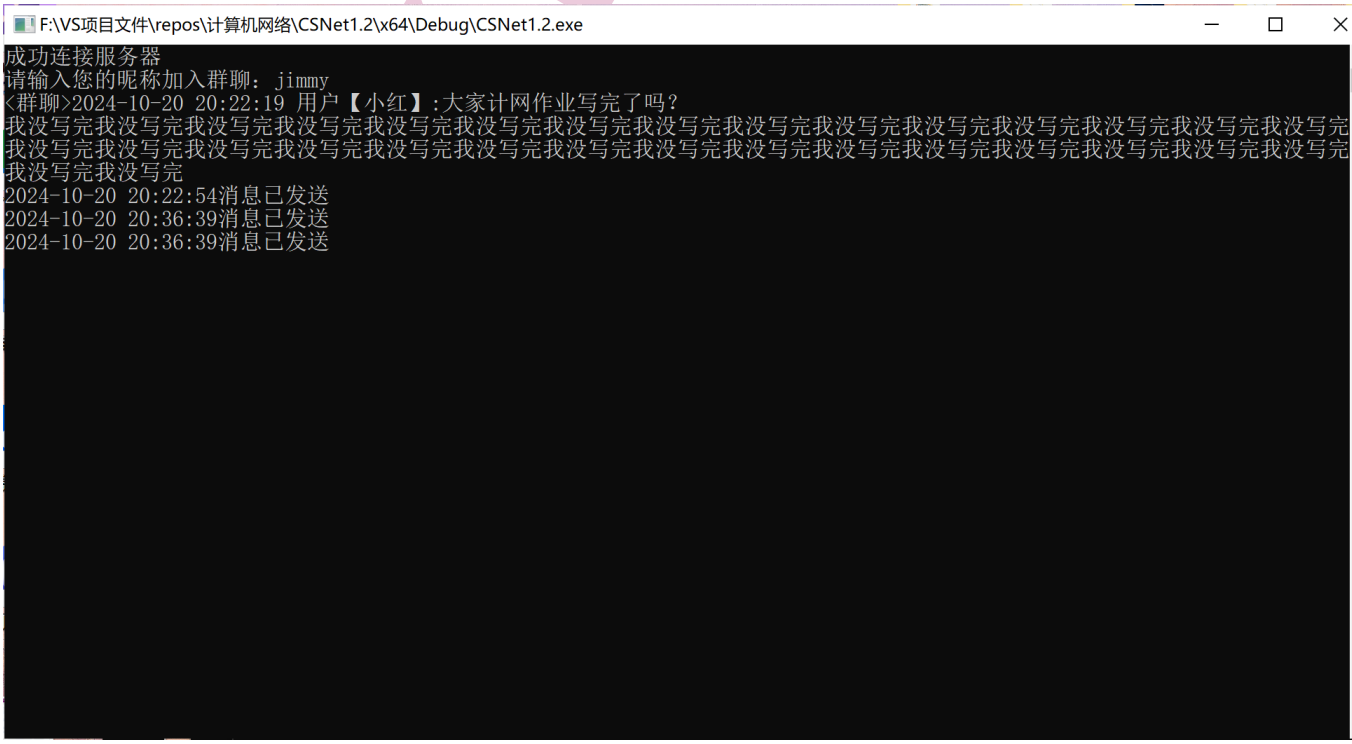


图 9: jimmy 发送一段超限（128 字符）的消息

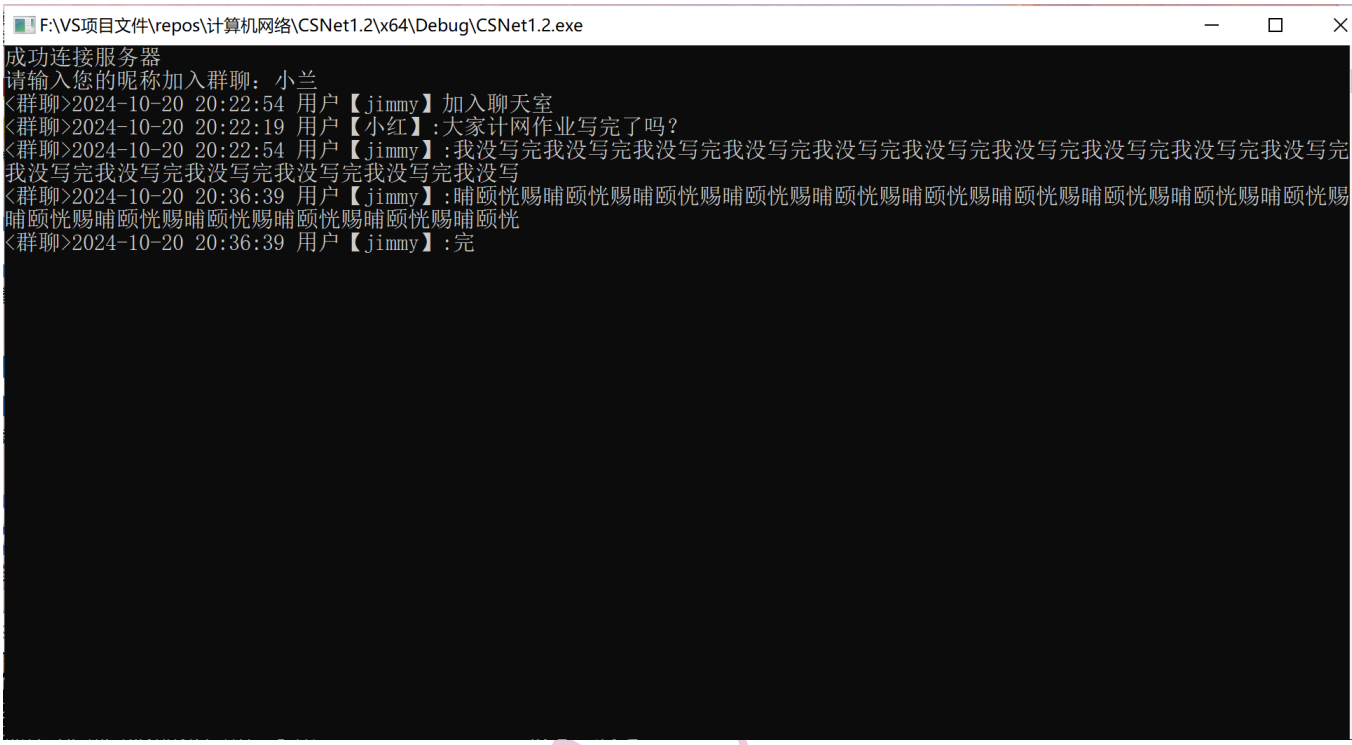


图 10: 小兰收到消息

（小兰替 jimmy 道歉）群聊发送了一段中英文混合的话，可以看到服务器正常显示，这时所有用户都是可以正常看到的.

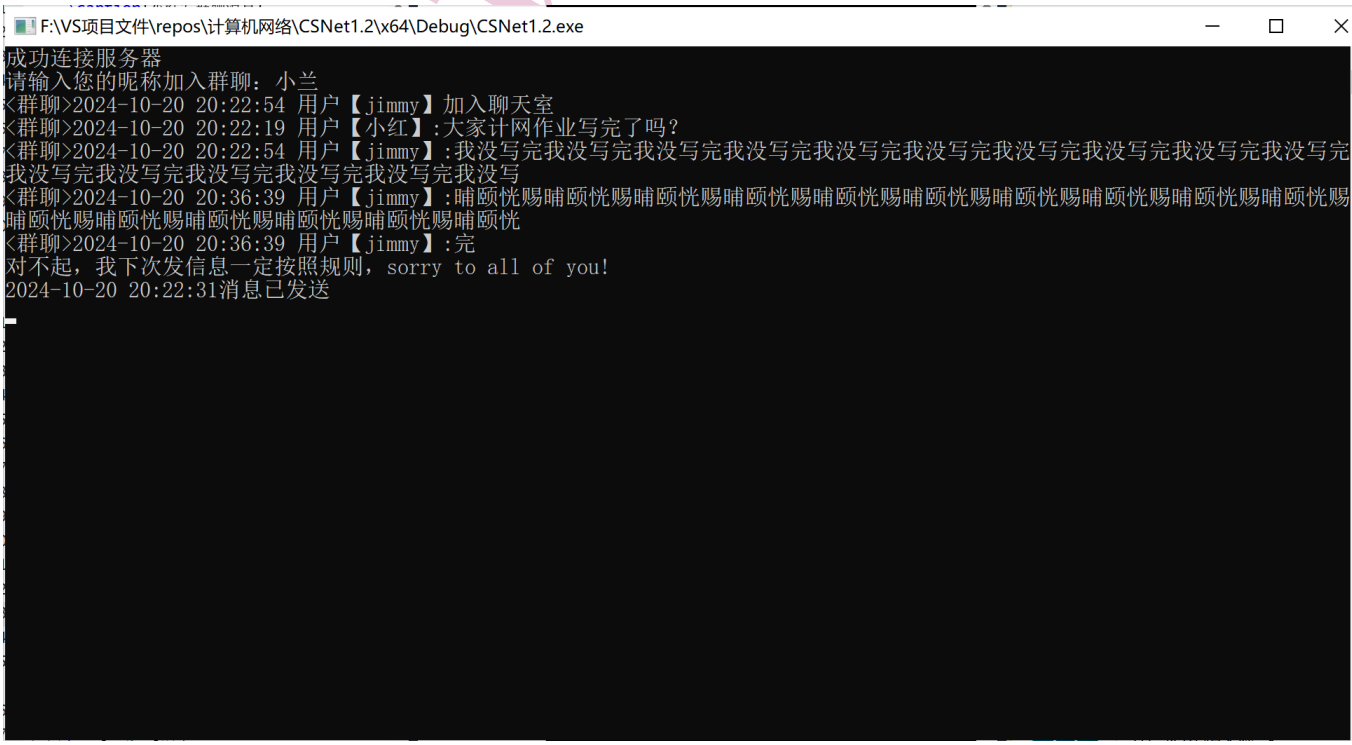


图 11: 小兰发送私聊消息

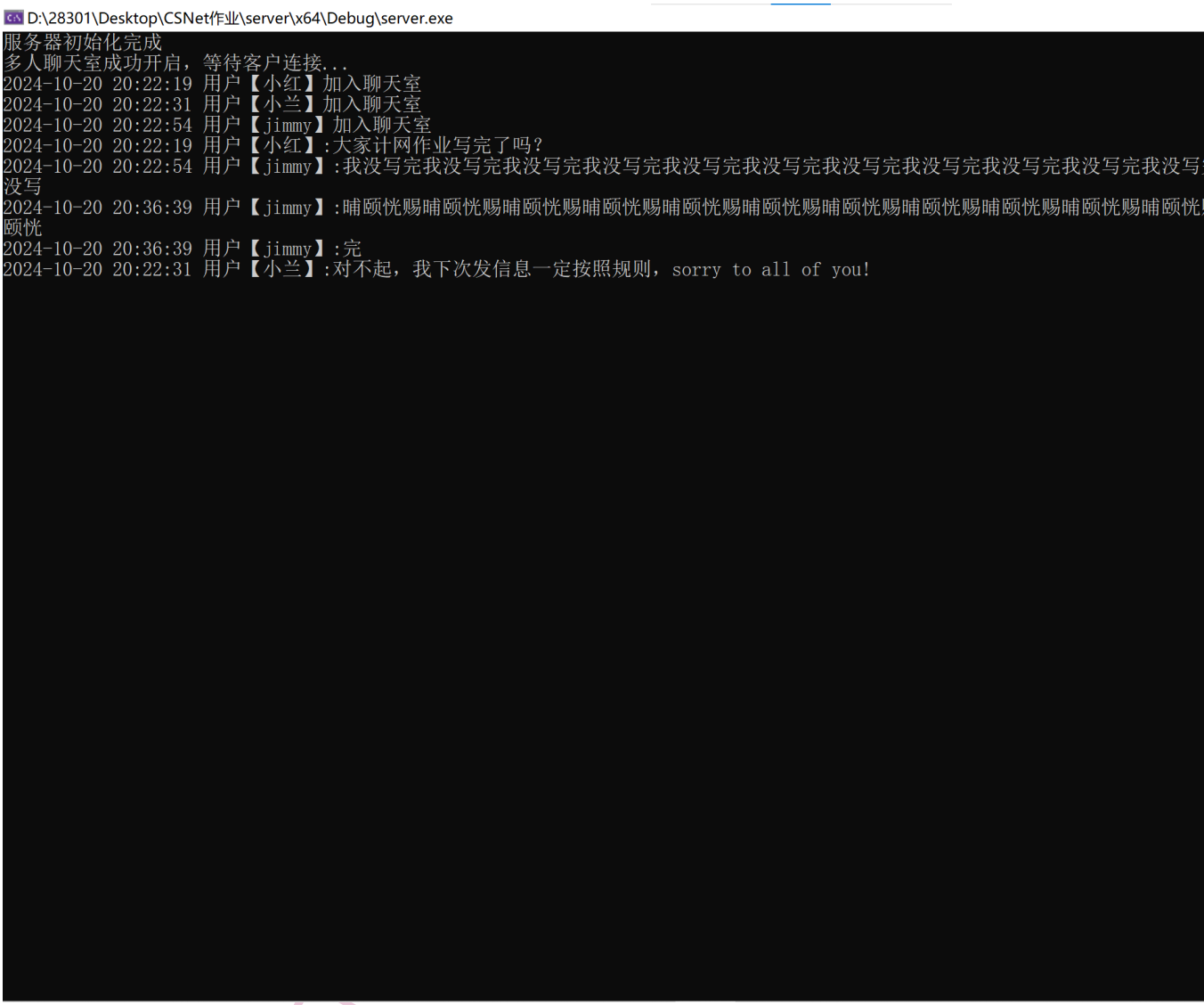


图 12: jimmy 收到消息

图 13: 服务器后台消息

这时候，jimmy 向小兰**私聊**表示感谢提醒聊天规则。此时只有小兰能看到私聊消息，小红无法看到。

图 14: jimmy 向小兰私聊

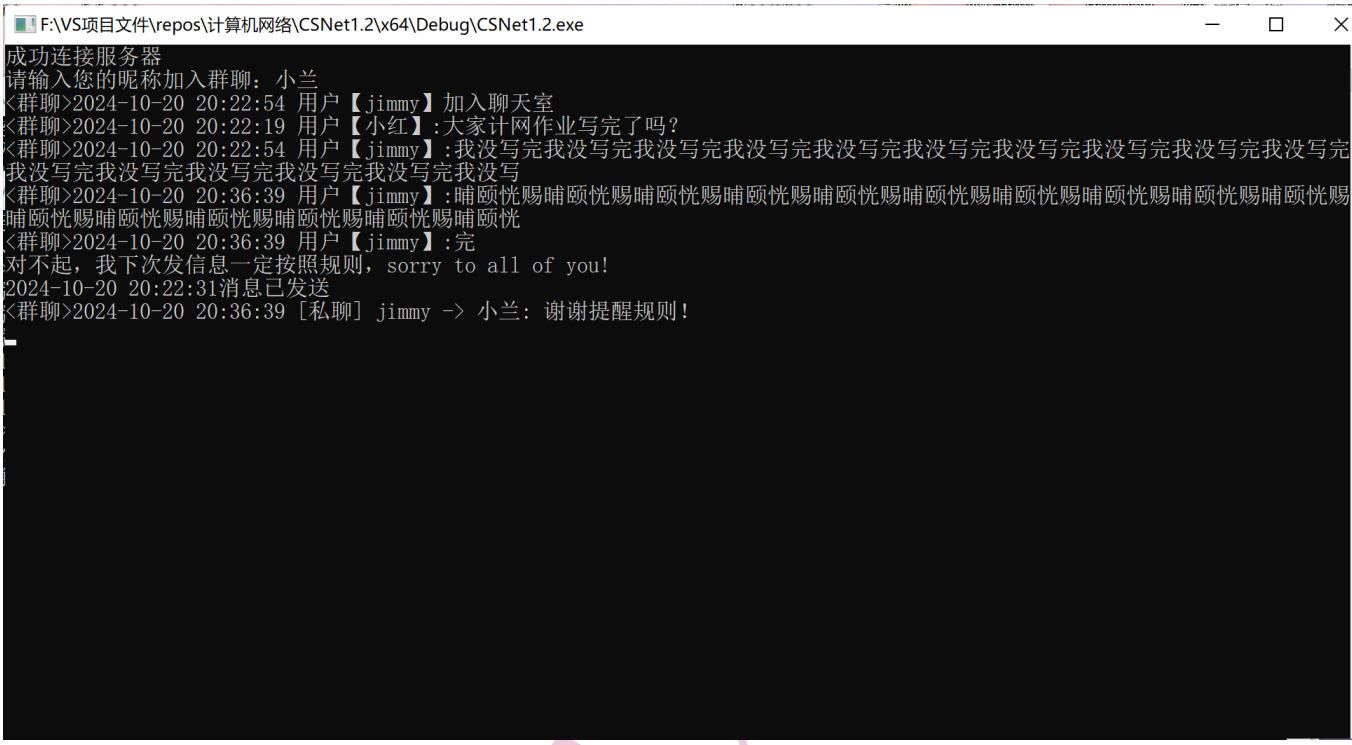


图 15: 小兰收到消息

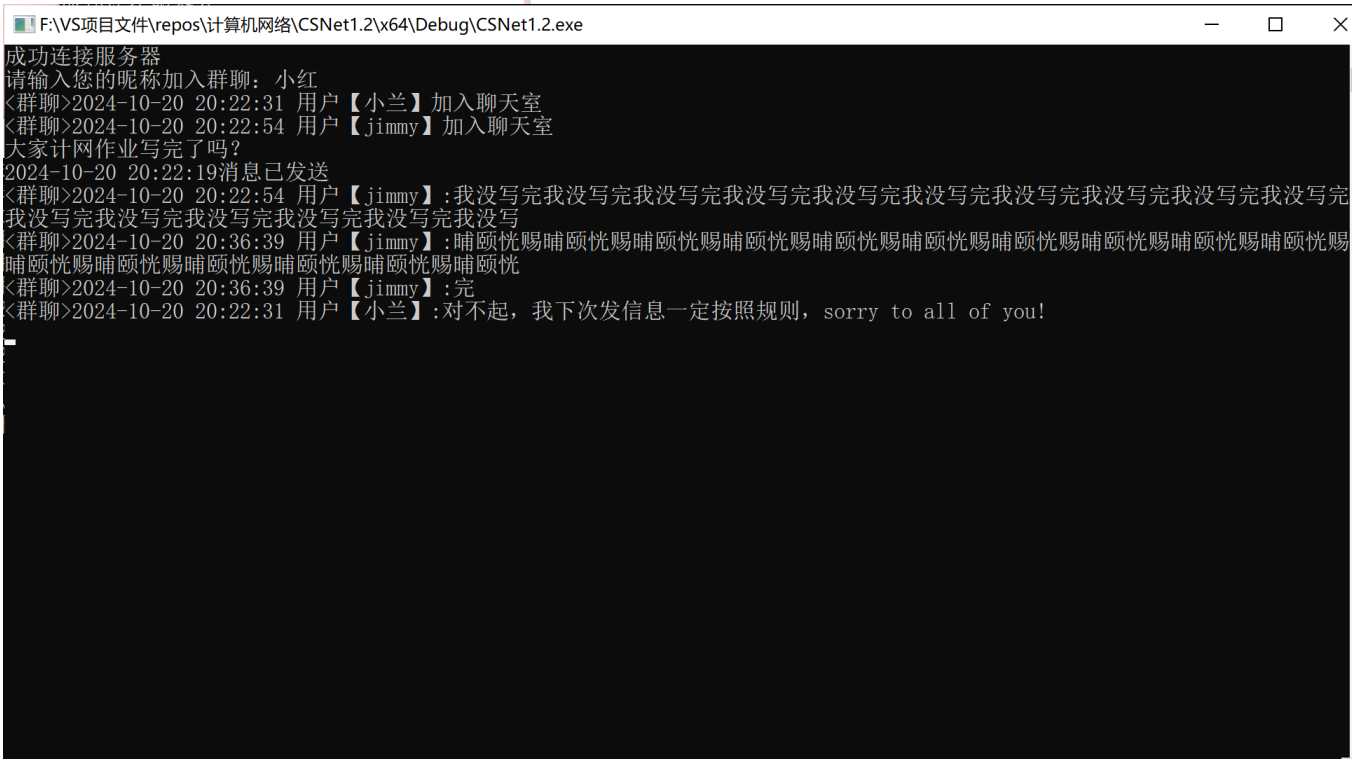


图 16: 小红看不到此私聊消息

此时，jimmy 退出聊天。当输入错误的指令会进行回溯，因此保证了消息的类型完整性:（程序直接退出了没有来得及截图）jimmy 先输入 text

exi] 由于指令不合法，回溯，然后输入 text
exit 成功退出聊天室。其他用户收到转发的群聊人员退出消息：

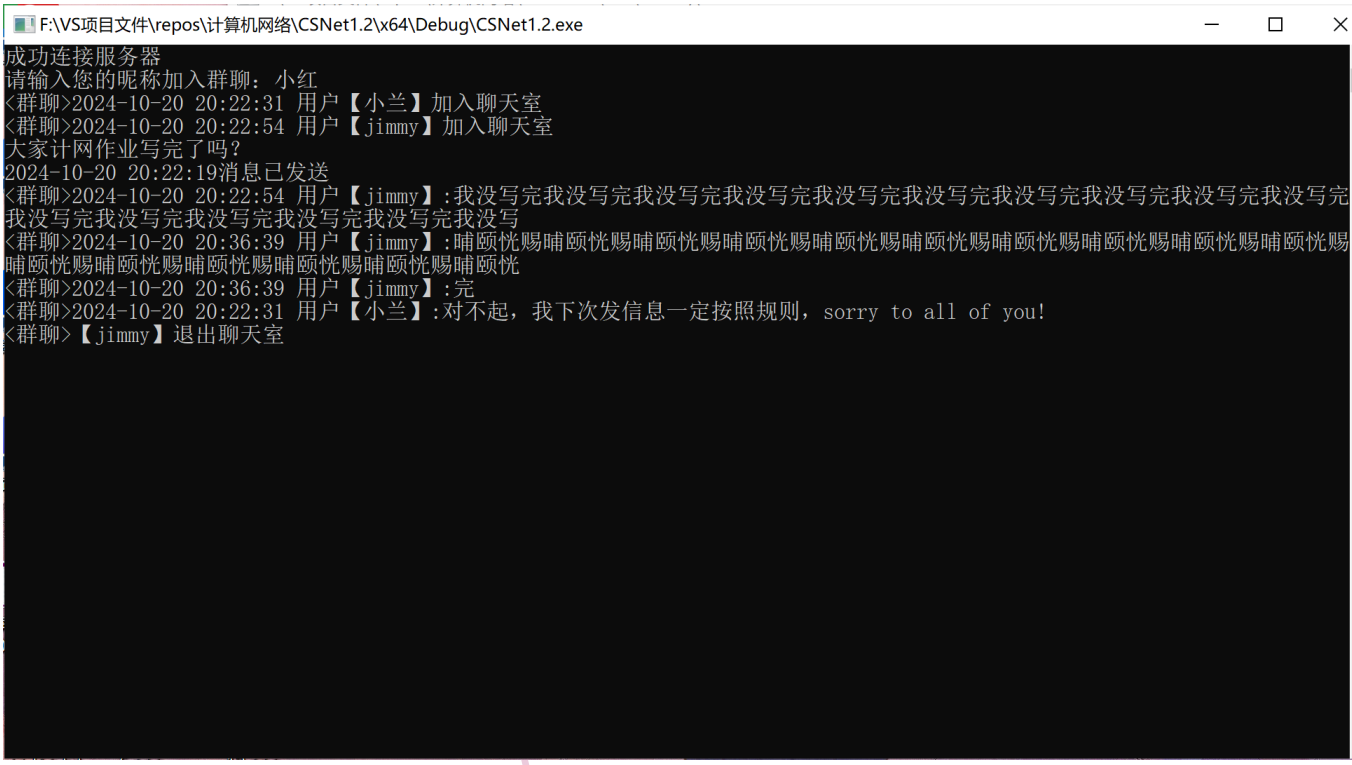


图 17: 小红看到 jimmy 的退出提示

小红、小兰依次退出聊天室，最后聊天服务器关闭，整个过程没有任何程序意外报错。

图 18: 小红准备退出

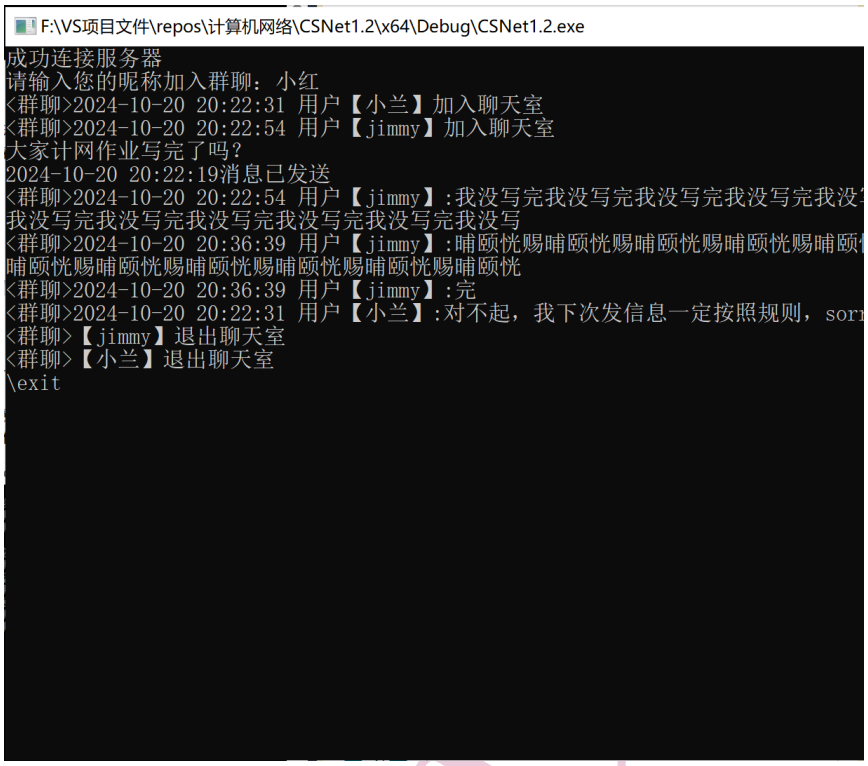

F:\VS项目文件\repos\计算机网络\CSNet1.2\x64\Debug\CSNet1.2.exe
成功连接服务器
请输入您的昵称加入群聊：小红
<群聊>2024-10-20 20:22:31 用户【小兰】加入聊天室
<群聊>2024-10-20 20:22:54 用户【jimmy】加入聊天室
大家计网作业写完了吗？
2024-10-20 20:22:19消息已发送
<群聊>2024-10-20 20:22:54 用户【jimmy】:我没写完我没写完我没写完我没写完我没写
我没写完我没写完我没写完我没写完我没写完我没写
<群聊>2024-10-20 20:36:39 用户【jimmy】:晡颐恍赐晡颐恍赐晡颐恍赐晡颐恍赐晡颐恍
晡颐恍赐晡颐恍赐晡颐恍赐晡颐恍赐晡颐恍赐晡颐恍
<群聊>2024-10-20 20:36:39 用户【jimmy】:完
<群聊>2024-10-20 20:22:31 用户【小兰】:对不起，我下次发信息一定按照规则，sorr
<群聊>【jimmy】退出聊天室
<群聊>【小兰】退出聊天室
\exit

图 19: 小兰准备退出

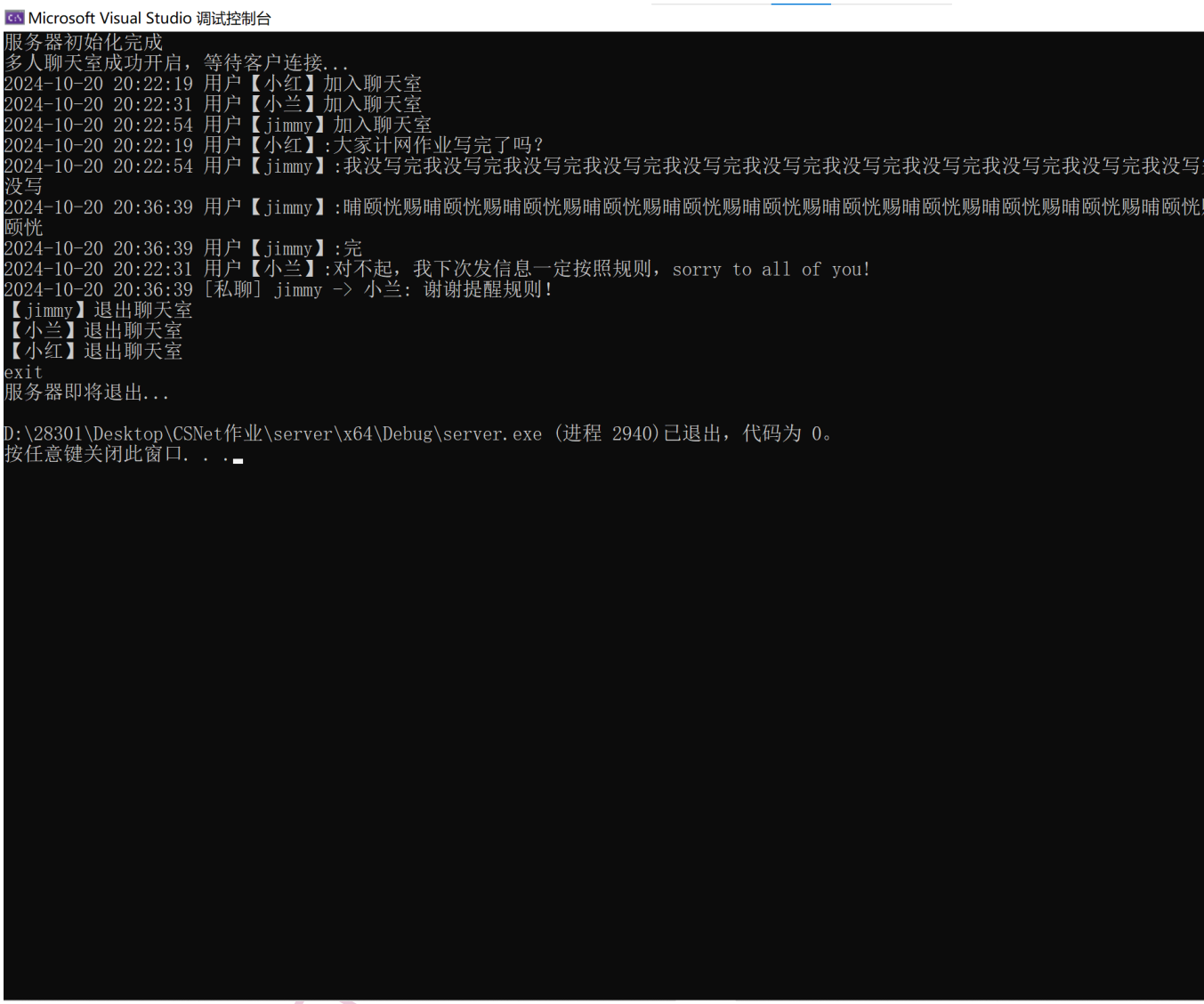


图 20: 服务器看到他们退出的提示并最后关闭

四、 问题及思考

(一) 关于运行逻辑合理性的解释

1. 需要服务器先上线，用户再上线，否则报错

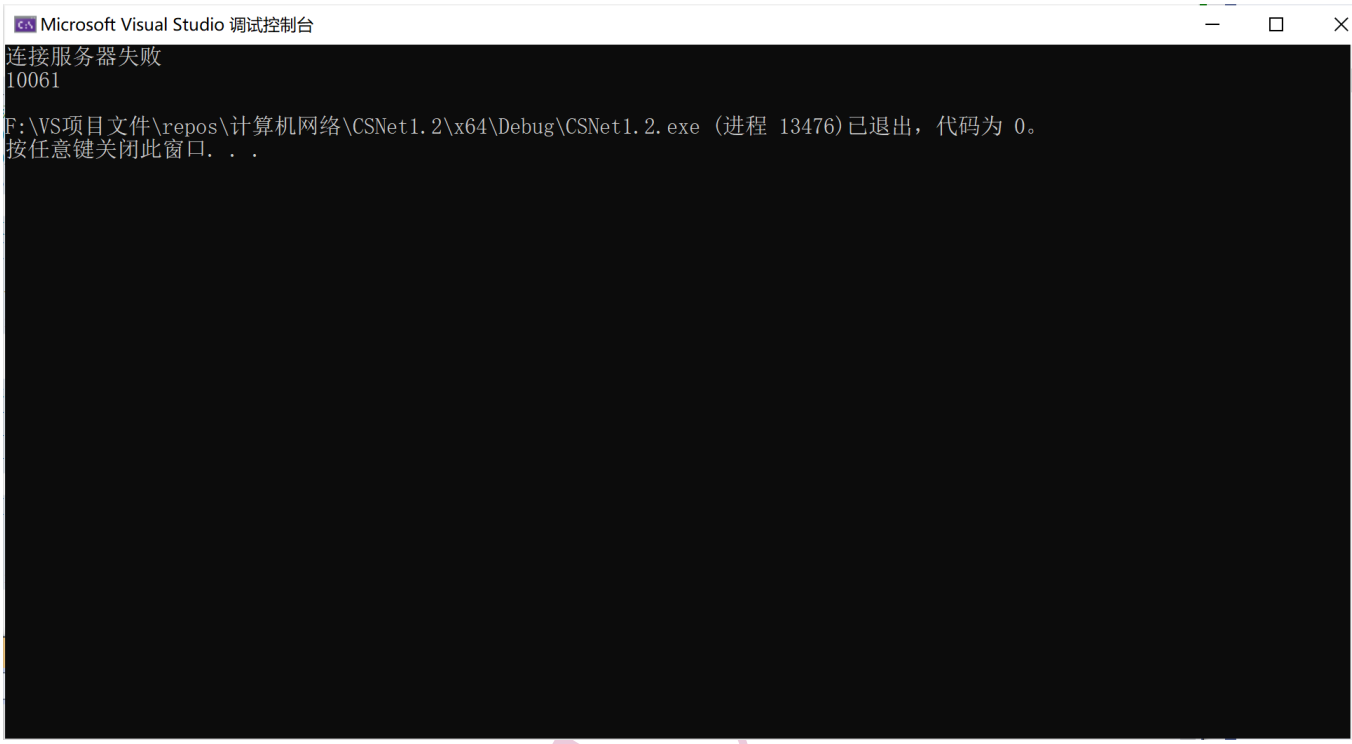


图 21: 服务器未启动，连接失败

2. 多线程互不干扰

首先体现的是服务器为每一个新用户的消息处理都分配一个新的线程，这样保证了不同用户不会相互干扰；其次，一个用户在与服务器进行通信的过程中，为了保证消息的接收和发送不能互相冲突，因此给消息的发送和消息的接收分别分配不同的线程，这样互不干扰。但是这里的互不干扰指的是发送消息的线程和接收消息的线程互不干扰，但是有时候用户在打字输入时，接收到消息，应该怎么办呢？实际上这种问题并不是利用多线程就能解决的问题，多线程解决的是程序内部的处理独立性，但是刚才的问题其实是控制台位置的冲突问题，由于本程序重点在于计算机网络的通信问题与线程处理问题，因此聊天界面相对简陋，每个用户的控制台只有一个输入输出接口，解决这个问题，用户可以先在别的地方打完完整的信息（类似于成熟的商业软件里的输入框），再把输入框的完整信息复制并粘贴到聊天发送入口发送即可。

(二) 关于数据丢失的测试

前面定义，用户昵称字符长度最多为 32, 聊天信息长度最多为 128，当用户发送的消息字符长度超过 128 时，由于我的程序是使用 fgets() 获取用户输入的，因此长度范围内和溢出的消息会分段发送，每段的长度就是 128.

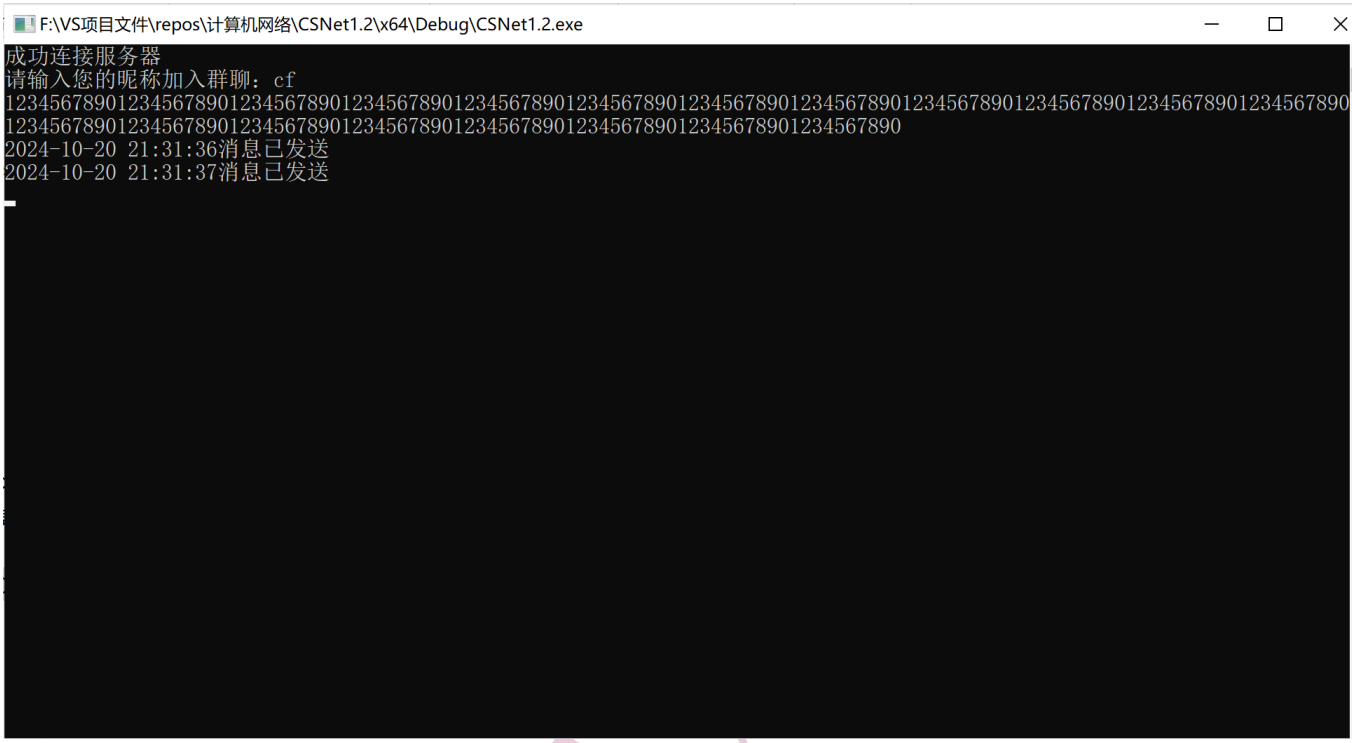


图 22: 分段发送消息

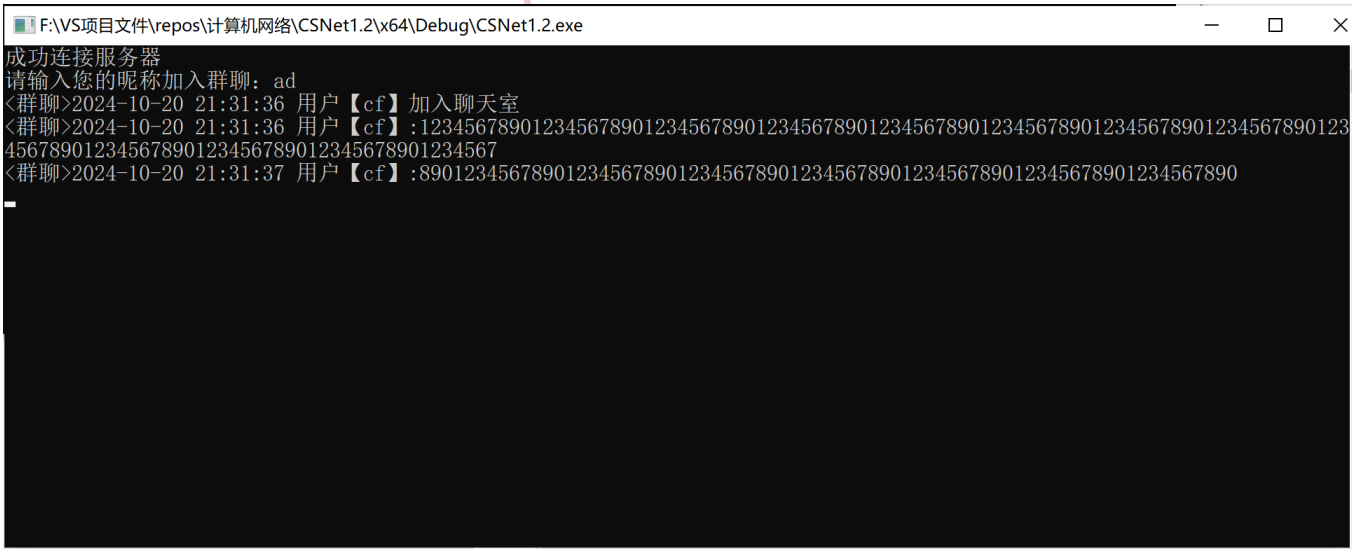


图 23: 分段接收消息

根据这个实验和前面的程序演示，当一次发送的消息长度合乎规则时，无论是中文还是英文，都不会丢失任何数据，数据都能完整正确地传输给服务器并转发；但当一次发送的消息长度大于协议规定的 128 个字符时，对于英文和一般 ASCII 字符，可以多次分段发送，也不会产生数据丢失，但是对于中文，相对于协议规定的溢出的部分则可能丢失数据。

无论是什么情况下的什么种类字符，程序都不会崩溃。

(三) 实验中遇到的问题及解决

在本实验中，一开始我不懂得如何让线程始终保持，不会退出，在客户端创建线程之后，程序就立即结束了。

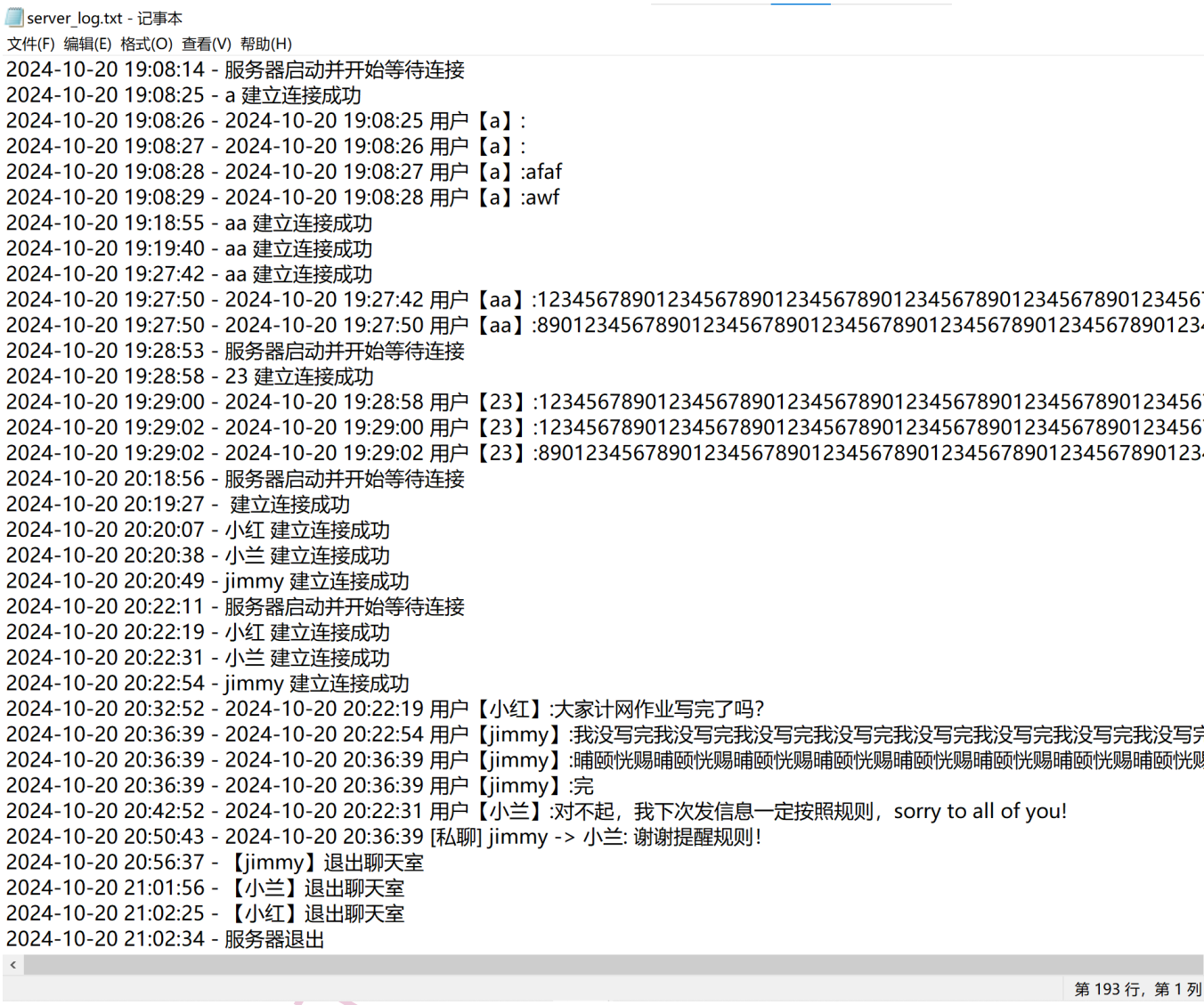
线程建立

```
1  HANDLE recvThread = CreateThread(NULL, 0, ReceiveMessages, (LPVOID)
       clientSocket, 0, NULL);
2  HANDLE sendThread = CreateThread(NULL, 0, SendMessages, (LPVOID)clientSocket,
       0, NULL);
```

程序创建线程后立即结束的原因在于，主程序并没有等待线程执行完毕就继续运行并结束了。这是因为主线程和子线程是并行执行的，主线程在启动子线程后不做任何等待控制，会直接执行剩余代码并退出。由于主线程的退出，子线程也随之被终止。为了解决这个问题，我引入 WaitForSingleObject(recvThread, INFINITE); 和 WaitForSingleObject(sendThread, INFINITE);。它们的作用是让主线程等待这两个线程的执行完成，再继续往下执行。这样主程序就不会在子线程完成之前结束，从而确保子线程能够正常运行和完成任务。

(四) 此多人聊天系统的优点总结

- **多线程处理**：每个客户端连接后，服务器为其创建一个独立的线程，支持多个客户端并发处理消息，实现了多人聊天室的功能，保证了用户体验的流畅性。
- **退出命令管理**：服务器支持实时检测控制台输入，通过单独的线程检测服务器的“exit”命令，实现服务器安全退出。
- **日志记录**：通过日志文件记录用户加入、退出及消息发送等重要事件，便于后续问题排查和系统维护。日志文件会自动生成在 server.exe 同目录下.
- **私聊功能**：支持用户使用特定命令 text(
msg) 进行私聊，满足特定消息定向传输的需求。
- **异常处理**：代码中对套接字的初始化、绑定、接受连接等操作进行错误检测与处理，避免因异常导致的程序崩溃，提高了系统的健壮性。


server_log.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
2024-10-20 19:08:14 - 服务器启动并开始等待连接
2024-10-20 19:08:25 - a 建立连接成功
2024-10-20 19:08:26 - 2024-10-20 19:08:25 用户【a】:
2024-10-20 19:08:27 - 2024-10-20 19:08:26 用户【a】:
2024-10-20 19:08:28 - 2024-10-20 19:08:27 用户【a】:afaf
2024-10-20 19:08:29 - 2024-10-20 19:08:28 用户【a】:awf
2024-10-20 19:18:55 - aa 建立连接成功
2024-10-20 19:19:40 - aa 建立连接成功
2024-10-20 19:27:42 - aa 建立连接成功
2024-10-20 19:27:50 - 2024-10-20 19:27:42 用户【aa】:12345678901234567890123456789012345678901234567890123456
2024-10-20 19:27:50 - 2024-10-20 19:27:50 用户【aa】:89012345678901234567890123456789012345678901234567890123
2024-10-20 19:28:53 - 服务器启动并开始等待连接
2024-10-20 19:28:58 - 23 建立连接成功
2024-10-20 19:29:00 - 2024-10-20 19:28:58 用户【23】:12345678901234567890123456789012345678901234567890123456
2024-10-20 19:29:02 - 2024-10-20 19:29:00 用户【23】:12345678901234567890123456789012345678901234567890123456
2024-10-20 19:29:02 - 2024-10-20 19:29:02 用户【23】:89012345678901234567890123456789012345678901234567890123
2024-10-20 20:18:56 - 服务器启动并开始等待连接
2024-10-20 20:19:27 -  建立连接成功
2024-10-20 20:20:07 - 小红 建立连接成功
2024-10-20 20:20:38 - 小兰 建立连接成功
2024-10-20 20:20:49 - jimmy 建立连接成功
2024-10-20 20:22:11 - 服务器启动并开始等待连接
2024-10-20 20:22:19 - 小红 建立连接成功
2024-10-20 20:22:31 - 小兰 建立连接成功
2024-10-20 20:22:54 - jimmy 建立连接成功
2024-10-20 20:32:52 - 2024-10-20 20:22:19 用户【小红】:大家计网作业写完了吗?
2024-10-20 20:36:39 - 2024-10-20 20:22:54 用户【jimmy】:我没写完我没写完我没写完我没写完我没写完我没写完我没写完我没写完
2024-10-20 20:36:39 - 2024-10-20 20:36:39 用户【jimmy】:晡颐恍赐晡颐恍赐晡颐恍赐晡颐恍赐晡颐恍赐晡颐恍赐晡颐恍赐晡颐恍赐
2024-10-20 20:36:39 - 2024-10-20 20:36:39 用户【jimmy】:完
2024-10-20 20:42:52 - 2024-10-20 20:22:31 用户【小兰】:对不起，我下次发信息一定按照规则，sorry to all of you!
2024-10-20 20:50:43 - 2024-10-20 20:36:39 [私聊] jimmy -> 小兰: 谢谢提醒规则!
2024-10-20 20:56:37 - 【jimmy】退出聊天室
2024-10-20 21:01:56 - 【小兰】退出聊天室
2024-10-20 21:02:25 - 【小红】退出聊天室
2024-10-20 21:02:34 - 服务器退出
```

第 193 行，第 1 列

图 24: 日志记录