南開大學
Nankai University

南　　开　　大　　学

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：李雅帆

学号：2213041

年级：2022 级

专业：信息安全

2024 年 10 月 19 日

目录

一、 聊天协议

(一) 聊天协议基本属性

1. 通信基础

   本聊天程序基于 TCP/IP 协议实现，通过套接字（Socket）进行网络通信。

2. 网络库

   使用 WinSock2 库，必须包含以下头文件：

```
#include <winsock2.h>
#include <ws2tcpip.h>
```

   连接时需要链接 `ws2_32.lib`。

```
#pragma comment(lib, "ws2_32.lib")
```

3. 服务器端口和缓冲区

   服务器端口号定义为 8888，缓冲区大小为 1024 字节。

```
#define PORT 8888
#define BUFFER_SIZE 1024
```

4. 多线程处理

   采用多线程技术来处理多个客户端的连接和消息。

```
thread(handleClient, clientSocket, clientSockets.size()).detach();
```

5. 消息广播

   服务器端实现消息广播功能，向所有连接的客户端发送消息，除了发送者。

```
void broadcastMessage(const string& message, SOCKET excludeSocket);
```

(二) 语法

1. 编码

   - 所有消息均采用 UTF-8 编码。

2. 消息格式

   - 系统消息：服务器与客户端初始化信息、异常处理信息。

   示例：

```
服务器已启动，等待客户端连接...
连接服务器成功
服务器的连接已断开
您已成功离开聊天室
......
```

- 连接消息：用户加入服务器时发送的消息。

示例：

```
用户 [1] 已加入聊天
```

- 文本消息：用户在聊天室发送的聊天消息。

格式如下：

```
用户 [<用户ID>]: <消息内容>
```

示例：

```
用户 [1]: 你好
```

- 离开消息：用户离开服务器时发送的消息。

示例：

```
用户 [1] 已退出聊天
```

(三) 时序

1. 服务器启动。
2. 客户端连接到服务器。
   - 客户端启动并尝试连接到服务器。
   - 服务器接受连接，并为每个客户端分配一个唯一的用户 ID。
3. 用户加入聊天。
   - 客户端连接成功后，服务器广播欢迎消息：用户 [ ] 已加入聊天。
4. 消息发送与接收。
   - 客户端可以发送消息，格式为 MESSAGE;< 用户 ID>;< 消息内容 >。
   - 服务器接收到消息后，以“ID：消息内容”的格式广播给所有连接的客户端。
5. 用户离开。
   - 客户端输入 EXIT，发送离开消息到服务器。
   - 服务器广播用户离开的消息：用户 [ ] 已离开聊天，并关闭与该客户端的连接。
6. 服务器关闭。
   - 服务器监听命令，当在服务器端输入 EXIT 时，关闭所有连接并停止服务。

（四） 命令

1. MESSAGE
   - 用户发送普通聊天消息时，格式为 MESSAGE; 用户 ID; 消息内容。
   - 服务器接收后，将该消息广播给所有其他用户。
2. JOIN
   - 当用户成功连接服务器时，发送 JOIN 消息。
   - 服务器处理后，广播用户加入的消息。
3. LEAVE
   - 当用户断开连接或输入 EXIT 时，服务器接收 LEAVE 消息并广播用户离开聊天室的消息。
4. EXIT
   - 当服务器接收到 EXIT 命令时，关闭所有客户端连接并终止服务，广播服务器关闭的消息。

（五） 创新

1. 用户 ID ： 服务器为每个连接的客户端分配一个唯一的 ID。
2. 系统消息：向聊天室的用户可以通知有关其他用户的消息。

二、 聊天室的具体实现

(一) 服务器端

```cpp
#include <iostream>
#include <thread>
#include <winsock2.h>
#include <ws2tcpip.h>
#include <vector>
#include <string>
#include <mutex>
using namespace std;

#pragma comment(lib, "ws2_32.lib")

#define PORT 8888              // 服务器端口号
#define BUFFER_SIZE 1024       // 接收数据的缓冲区大小。

SOCKET serverSocket;
vector<SOCKET> clientSockets;
mutex clientMutex;

// 向客户端发送消息
void broadcastMessage(const string& message, SOCKET excludeSocket) {
    lock_guard<mutex> lock(clientMutex);
    for (SOCKET clientSocket : clientSockets) {
        if (clientSocket != excludeSocket) {
            send(clientSocket, message.c_str(), message.size(), 0);
        }
    }
}

// 处理与单个客户端的连接
void handleClient(SOCKET clientSocket, int userID) {
    char buffer[BUFFER_SIZE];

    string userIDMessage = "您是 " + to_string(userID) + " 号用户\n";
    send(clientSocket, userIDMessage.c_str(), userIDMessage.size(), 0);

    string welcomeMessage = "用户 [" + to_string(userID) + "] 已加入聊天\n";
    broadcastMessage(welcomeMessage, clientSocket);
    cout << welcomeMessage;

    while (true) {
        int bytesReceived = recv(clientSocket, buffer, sizeof(buffer) - 1, 0)
            ;
        if (bytesReceived <= 0) {
            break;
```

```
44          }
45
46          buffer[bytesReceived] = '\0';
47          string fullMessage = "用户␣[" + to_string(userID) + "]:␣" + string(
                buffer) + "\n";
48          broadcastMessage(fullMessage, clientSocket);
49          cout << fullMessage;
50      }
51
52      closesocket(clientSocket);
53      {
54          lock_guard<mutex> lock(clientMutex);
55          clientSockets.erase(remove(clientSockets.begin(), clientSockets.end()
                , clientSocket), clientSockets.end());
56      }
57
58      string leaveMessage = "用户␣[" + to_string(userID) + "]␣已离开聊天\n";
59      broadcastMessage(leaveMessage, clientSocket);
60      cout << leaveMessage;
61  }
62
63  void listenForExit() {
64      string command;
65      while (true) {
66          getline(cin, command);
67          if (command == "EXIT") {
68              broadcastMessage("服务器的连接已断开\n", INVALID_SOCKET);
69              {
70                  lock_guard<mutex> lock(clientMutex);
71                  for (SOCKET clientSocket : clientSockets) {
72                      closesocket(clientSocket);
73                  }
74                  clientSockets.clear();
75              }
76              closesocket(serverSocket);
77              WSACleanup();
78              exit(0);
79          }
80      }
81  }
82
83  int main() {
84      WSADATA wsaData;
85      if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
86          cerr << "WSAStartup␣初始化失败" << endl;
87          return 1;
88      }
89
```

```
90      serverSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
91      if (serverSocket == INVALID_SOCKET) {
92          cerr << "创建套接字失败: " << WSAGetLastError() << endl;
93          WSACleanup();
94          return 1;
95      }
96
97      sockaddr_in serverAddr;
98      serverAddr.sin_family = AF_INET;
99      serverAddr.sin_port = htons(PORT);
100     serverAddr.sin_addr.s_addr = INADDR_ANY;
101
102     if (bind(serverSocket, (sockaddr*)&serverAddr, sizeof(serverAddr)) ==
            SOCKET_ERROR) {
103         cerr << "绑定地址失败: " << WSAGetLastError() << endl;
104         closesocket(serverSocket);
105         WSACleanup();
106         return 1;
107     }
108
109     if (listen(serverSocket, SOMAXCONN) == SOCKET_ERROR) {
110         cerr << "监听失败: " << WSAGetLastError() << endl;
111         closesocket(serverSocket);
112         WSACleanup();
113         return 1;
114     }
115
116     cout << "服务器已启动, 等待客户端连接..." << endl;
117
118     thread exitThread(listenForExit);
119     exitThread.detach();
120
121     while (true) {
122         SOCKET clientSocket = accept(serverSocket, nullptr, nullptr);
123         if (clientSocket == INVALID_SOCKET) {
124             cerr << "接受连接失败" << WSAGetLastError() << endl;
125             continue;
126         }
127
128         {
129             lock_guard<mutex> lock(clientMutex);
130             clientSockets.push_back(clientSocket);
131         }
132
133         thread(handleClient, clientSocket, clientSockets.size()).detach();
134     }
135
136     // 清理和关闭
```

```
137     closesocket(serverSocket);
138     WSACleanup();
139     return 0;
140 }
```

- 服务端代码实现了一个基本的聊天室服务器，主要用于接受和处理多个客户端的连接。
- 通过 WSAStartup 初始化 Winsock 库，随后创建一个 TCP 套接字，并将其绑定到本地的 8888 端口以监听来自客户端的连接请求。
- 服务器在启动后进入一个无限循环，使用 accept 函数接受新连接，并为每个客户端启动一个独立的处理线程。
- 这些线程通过 handleClient 函数进行管理，功能包括发送欢迎消息、接收客户端消息并将其广播给其他连接的客户端。
- 每当一个客户端发送消息时，服务器会将消息格式化为包含用户 ID 的字符串，并通过 broadcastMessage 函数将其发送到其他所有客户端，确保实时沟通。
- 服务器还设置了一个专门的线程来监听用户输入，允许输入"EXIT" 命令以安全关闭服务器。
- 在这个过程中，使用 mutex 确保对客户端套接字列表的访问是线程安全的，避免出现竞态条件。
- 当客户端断开连接时，相应的线程会清理资源，广播离开消息并从客户端列表中删除已离开的客户端。

(二) 客户端

```
1  #include <iostream>
2  #include <thread>
3  #include <winsock2.h>
4  #include <ws2tcpip.h>
5  #include <string>
6
7  using namespace std;
8
9  #pragma comment(lib, "ws2_32.lib")
10
11 #define PORT 8888
12 #define BUFFER_SIZE 1024
13
14 SOCKET clientSocket;
15
16 void receiveMessages() {
17     char buffer[BUFFER_SIZE];
18     while (true) {
19         int bytesReceived = recv(clientSocket, buffer, sizeof(buffer) - 1, 0)
               ;
```

```
20          if (bytesReceived <= 0) {
21              cerr << "您已成功离开聊天室~" << endl;
22              break;
23          }
24          buffer[bytesReceived] = '\0';
25          cout << buffer; // 输出接收到的消息
26      }
27  }
28
29  int main() {
30      WSADATA wsaData;
31      if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
32          cerr << "WSAStartup初始化失败" << endl;
33          return 1;
34      }
35
36      clientSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
37      if (clientSocket == INVALID_SOCKET) {
38          cerr << "创建套接字失败: " << WSAGetLastError() << endl;
39          WSACleanup();
40          return 1;
41      }
42
43      sockaddr_in serverAddr;
44      serverAddr.sin_family = AF_INET;
45      serverAddr.sin_port = htons(PORT);
46      inet_pton(AF_INET, "127.0.0.1", &serverAddr.sin_addr);
47
48      if (connect(clientSocket, (sockaddr*)&serverAddr, sizeof(serverAddr)) ==
            SOCKET_ERROR) {
49          cerr << "连接失败:" << WSAGetLastError() << endl;
50          cerr << "请检查服务器是否运行以及网络连接。" << endl;
51          closesocket(clientSocket);
52          WSACleanup();
53          return 1;
54      }
55
56      cout << "连接服务器成功" << endl;
57      cout << "*********************************************" << endl;
58      cout << "欢迎来到聊天室! " << endl;
59      cout << "(输入EXIT可退出聊天哦) " << endl;
60      cout << "*********************************************" << endl;
61
62      thread(receiveMessages).detach();
63
64      char message[BUFFER_SIZE];
65      while (true) {
66          cin.getline(message, BUFFER_SIZE);
```

```
67          if (strcmp(message, "EXIT") == 0) {
68              break;
69          }
70          send(clientSocket, message, static_cast<size_t>(strlen(message)), 0);
71      }
72      closesocket(clientSocket);
73      WSACleanup();
74      return 0;
75  }
```

- 客户端代码实现了一个简单的聊天室客户端，使用 Winsock 库进行网络通信。
- main 函数首先初始化 Winsock，创建一个 TCP 套接字，并连接到本地服务器（端口号为 8888）。
- 成功连接后，打印欢迎信息，并启动一个名为 receiveMessages 的线程，该线程不断接收并输出服务器发送的消息。
- 主线程则通过循环等待用户输入消息，若输入”EXIT”，则退出循环。
- 在消息发送过程中，客户端将用户输入的消息发送到服务器。
- 最后，客户端关闭套接字并进行资源清理。
- 该客户端支持实时消息接收和发送，提供了基本的聊天室功能，用户可以与其他客户端进行互动。

三、 程序界面展示

(一) 启动服务器端

- 如图1所示，服务器端成功启动后会出现“服务器已启动，等待客户端连接...”的字样。

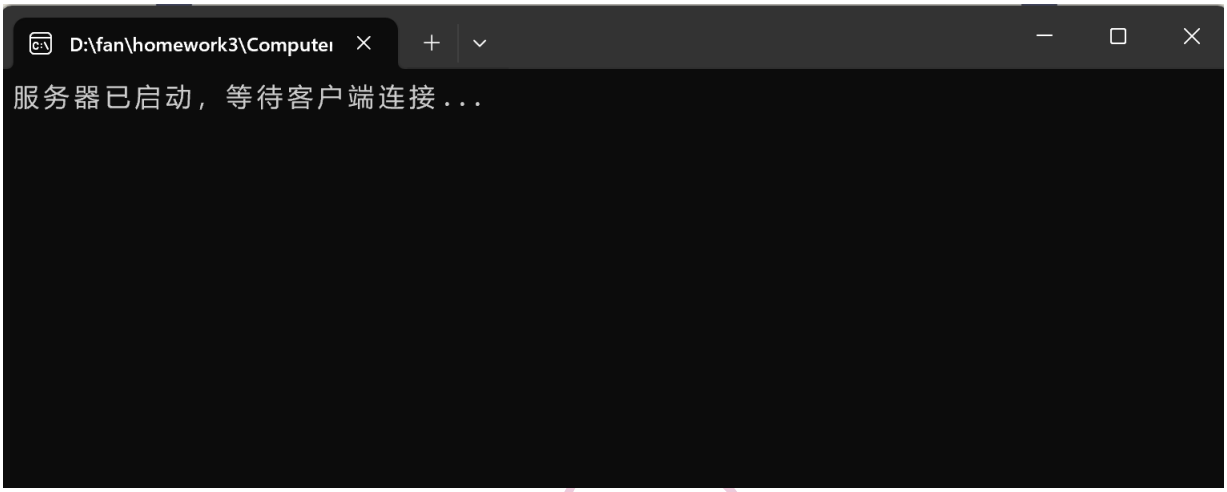

图 1: 启动服务器

(二) 用户端加入聊天室

- 如图2所示，用户端请求连接到服务器端，连接成功后，用户加入聊天室。
- 成功加入聊天室后，服务器端给用户发送消息通知用户 ID。
- 在服务器端公告用户加入消息。
- 先加入的用户会接收到服务器端发送的后面用户加入聊天室的通知。

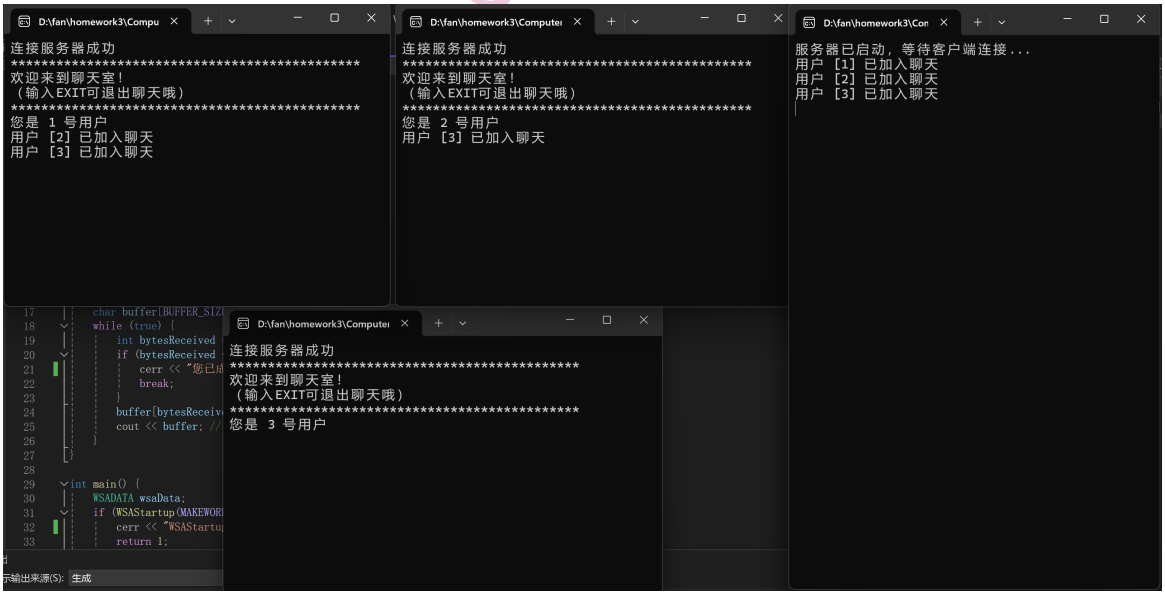

图 2: 加入聊天室

(三) 多人聊天

- 如图3所示，用户发送消息，服务器端打印所有消息。
- 服务器端将用户发送的消息广播给其他所有的用户。
- 中文、英文、长消息和短消息都发送成功。

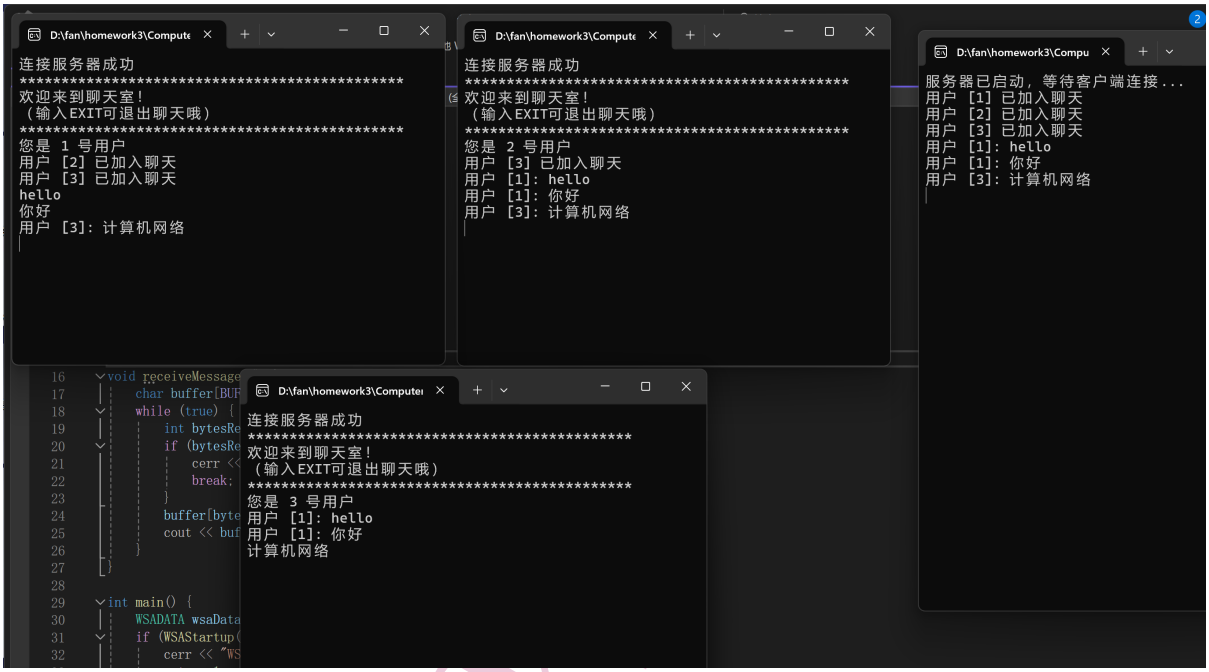

图 3: 聊天

(四) 退出聊天

- 用户端输入 EXIT 退出聊天室，服务端广播该用户的退出消息给其他用户并在服务端打印，如图4所示。

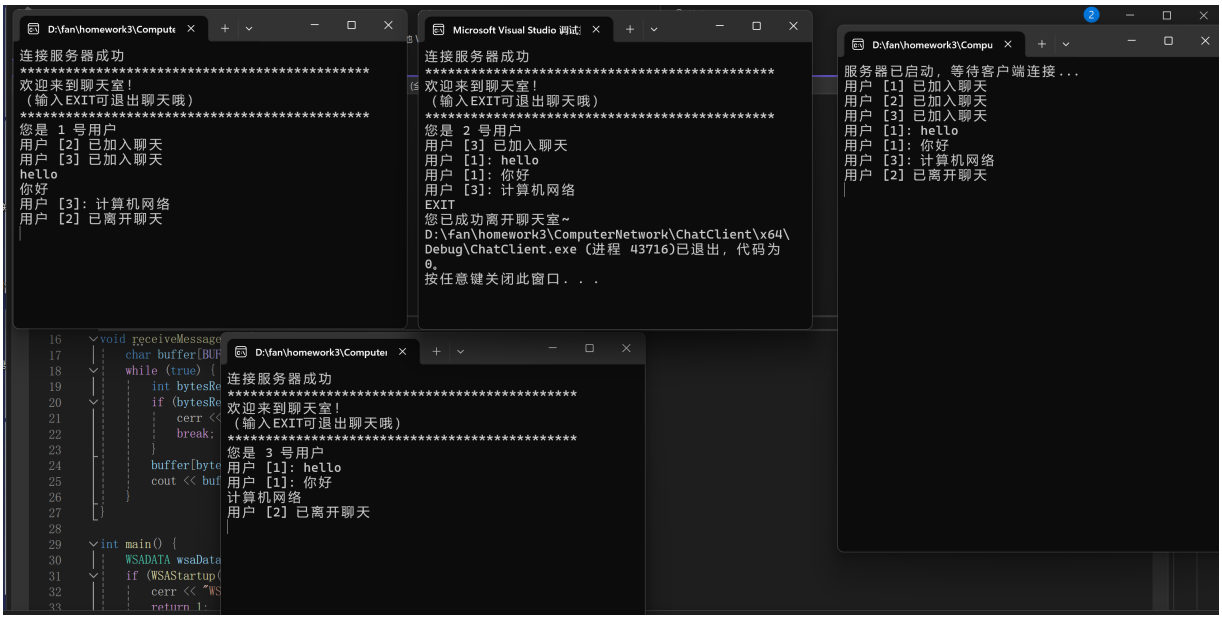

图 4: 用户端退出

- 某一用户退出聊天室后，其他用户还可以继续正常聊天，如图5所示。

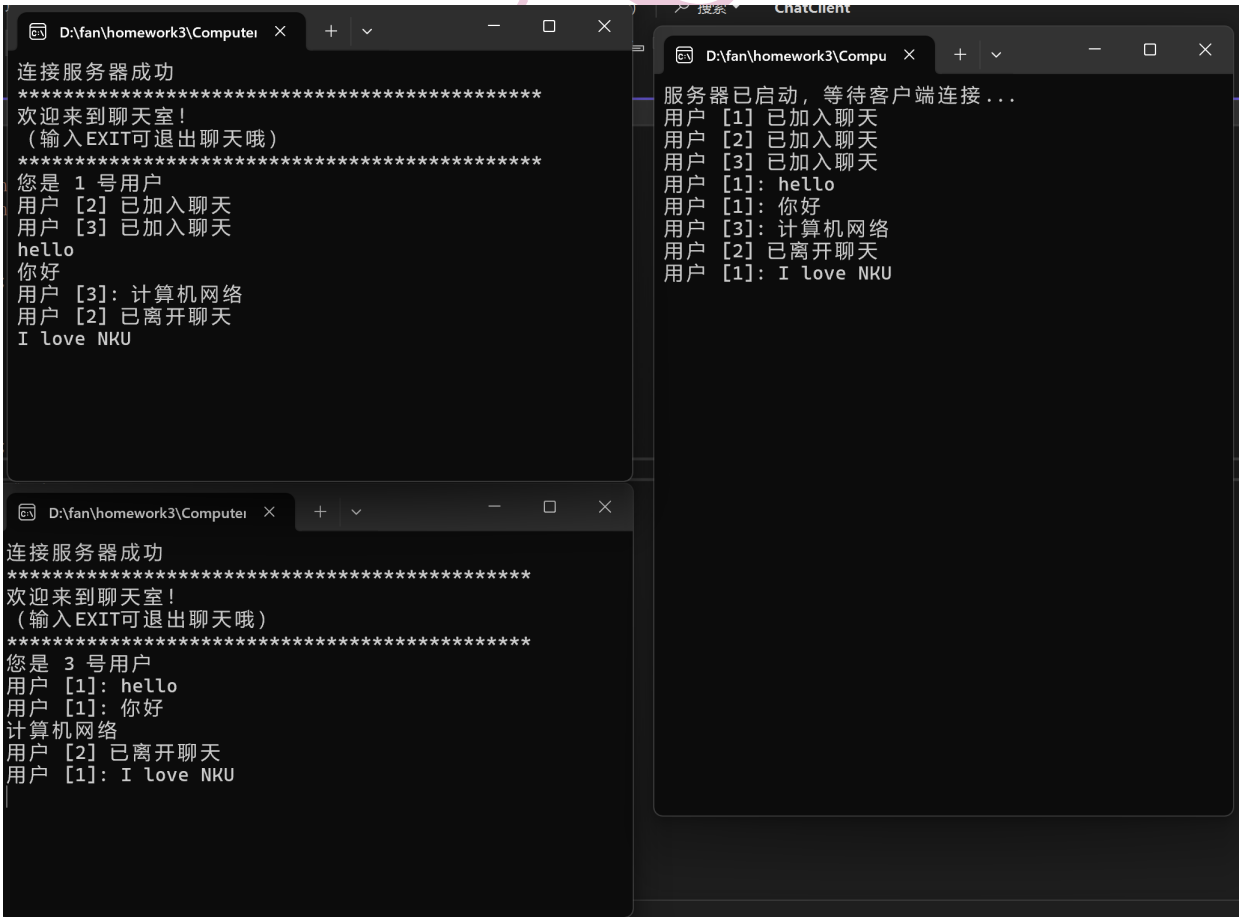

图 5

- 服务器端输入 EXIT 也可以直接结束聊天，用户端会接收到“服务亲的连接已断开”消息，如图6所示。

图 6: 服务端退出