计算机网络第一次实验报告

2211421 何禹姗

实验1：利用Socket，编写一个聊天程序

1、应用层协议设计

（1）消息类型、语法、语义

欢迎消息 WELCOME:<client_id>：用户进入聊天室时发送的欢迎消息，用户连接后，服务端发送欢迎消息给该用户，并广播给其他用户。

群聊消息 MESSAGE:：支持多人聊天，服务端将此消息广播给所有在线用户（除了发送者）。

私聊消息 PRIVATE:<target_id>:：针对特定用户的私聊消息，服务端解析消息中的 `<target_id>` 和 `<message>`，并将私聊消息发送给目标用户。

退出消息 QUIT：用户退出聊天室的消息，用户输入 `QUIT` 后，服务端广播该用户退出的消息，并关闭连接。

（2）消息时序

客户端连接

- 客户端连接到服务端，服务端分配一个 `client_id` 并记录连接信息。
- 服务端向客户端发送 `WELCOME:<client_id>` 消息。
- 服务端广播 `WELCOME:<client_id>` 消息给所有在线用户。

群聊消息发送

- 客户端发送 `MESSAGE:<message>` 到服务端。
- 服务端广播 `MESSAGE:<message>` 给所有在线用户（除了发送者）。

私聊消息发送

- 客户端发送 `PRIVATE:<target_id>:<message>` 到服务端。
- 服务端解析消息并发送 `PRIVATE:<target_id>:<message>` 给目标用户。

退出消息发送

- 客户端发送 `QUIT` 到服务端。
- 服务端广播 `NOTIFY:用户退出` 消息给所有在线用户，并关闭连接。

（3）设计思路

服务器撰写步骤

1、创建socket套接字 socket(AF_INET, SOCK_STREAM, 0)

2、给socket绑定端口号 bind()

3、给socket开启监听属性 listen()

4、等待客户端连接 accept()

5、开始通讯

6、关闭连接

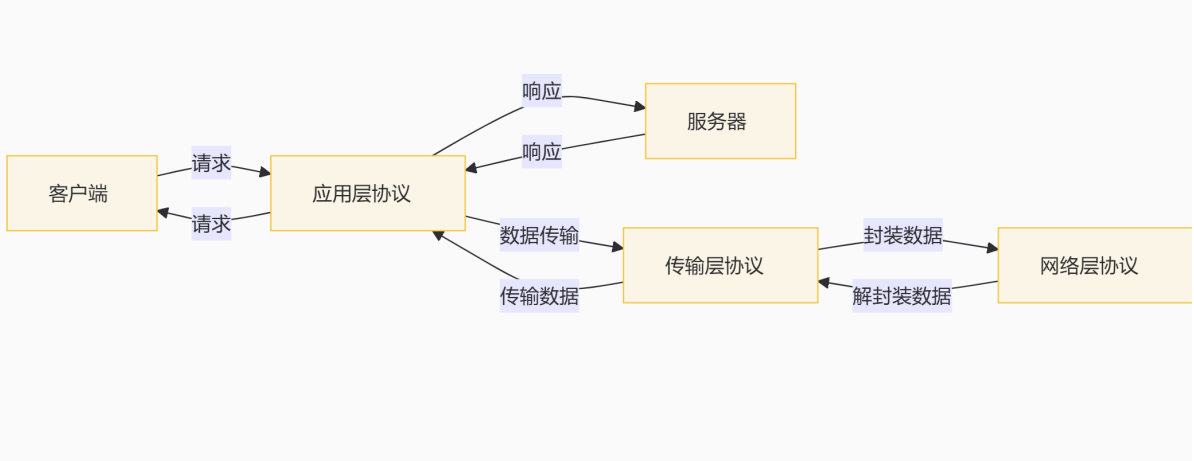
客户端撰写步骤

1、创建socket套接字 socket(AF_INET, SOCK_STREAM, 0)

2、连接服务器 connect()

3、开始通讯

4、关闭连接



2、各模块功能

（1）服务器端代码 server.c

引入头文件和库文件，使程序可以使用Winsock API进行网络编程。

```
#include<stdio.h>
#include<string.h>
#include <stdlib.h>
#include <time.h>
#include<winSock2.h> //windows网络头文件
#pragma comment(lib,"ws2_32.lib") //windows网络库文件
```

定义一个 `MAX_CLIENTS` 常量，指定了服务器最多能支持的客户端数量。同时定义了一个 `ClientInfo` 结构体，用于存储客户端的套接字和 ID，初始化全局变量。

```
#define MAX_CLIENTS 100 // 最大客户端数量

// 全局变量用于存储所有客户端的套接字和ID
struct ClientInfo {
    SOCKET socket;
    int id;
};
// 全局变量用于存储所有客户端的套接字
struct ClientInfo clients[MAX_CLIENTS];
int client_count = 1;
```

broadcast_message为广播消息函数，将消息广播给所有在线的客户端，除了消息的发送者。

```
void broadcast_message(int sender_id, const char* message)
{
    for (int i = 0; i < client_count; ++i)
    {
        if (clients[i].id != sender_id) // 不向发送者转发消息
        {
            send(clients[i].socket, message, (int)strlen(message), 0);
        }
    }
}
```

broadcast_message_to函数广播消息给指定的客户端。

```
// 广播消息给指定客户端
void broadcast_message_to(int client_id, const char* message)
{
    for (int i = 0; i < client_count; ++i)
    {
        if (clients[i].id == client_id)
        {
            send(clients[i].socket, message, strlen(message), 0);
            return; // 找到并发送后立即返回
        }
    }
}
```

receive_thread_func为接收线程函数，这个函数在一个单独的线程中运行，处理来自客户端的消息。其中包括处理普通消息、私聊消息和退出消息的功能。

```
DWORD WINAPI receive_thread_func(LPVOID lpThreadParameter)
{
    SOCKET client_socket = ((SOCKET*)lpThreadParameter)[0];
    int client_id = ((SOCKET*)lpThreadParameter)[1]; // 获取客户端ID

    char welcome_message[1024];
    sprintf(welcome_message, "用户%d进入聊天室", client_id);
    printf("%s\n", welcome_message);
    broadcast_message(client_id, welcome_message);
    char name_message[1024];
    sprintf(name_message, "你是用户%d", client_id);
    broadcast_message_to(client_id, name_message);
    while (1)
    {
        char buffer[1024] = { 0 }; // 接收数据
        int ret = recv(client_socket, buffer, 1024, 0); // 接收数据
        if (ret <= 0)
        {
            break;
        }

        // 检测退出信号
        if (strncmp(buffer, "QUIT", 4) == 0)
        {
```

```
            // 构造退出消息并广播
            char quit_message[1024];
            sprintf(quit_message, "用户%d退出聊天室", client_id);
            broadcast_message(client_id, quit_message);
            printf("用户%d退出聊天室 \n", client_id);
            break;
        }

        // 检查是否为私聊消息
        if (strncmp(buffer, "PRIVATE:", 8) == 0)
        {
            // 处理私聊消息
            int target_id;
            char message[1024];
            if (sscanf(buffer + 8, "%d:%[^:]", &target_id, message) == 2)
            {
                // 发送私聊消息给目标用户
                for (int i = 0; i < client_count; ++i)
                {
                    if (clients[i].id == target_id && clients[i].id != client_id)
                    {
                        char private_message[1024];
                        printf("私聊消息来自用户%d发送给用户%d: %s \n", client_id,
target_id, message);
                        sprintf(private_message, "私聊消息来自用户%d: %s",
client_id, message);
                        send(clients[i].socket, private_message,
strlen(private_message), 0);
                    }
                }
            }
            else
            {
                // 如果解析失败，可以打印错误信息或忽略该消息
                printf("私聊消息格式错误。\n");
            }
        }
        else
        {
            // 获取当前时间
            time_t now = time(NULL);
            struct tm tm_info;
            localtime_s(&tm_info, &now);
            char time_str[20];
            strftime(time_str, sizeof(time_str), "%X", &tm_info); // 格式化时间为字
符串

            printf("%d: %s   [%s]\n", client_id, buffer, time_str); // 打印

            // 将接收到的消息广播给其他客户端，并加上时间戳
            char broad_message[1024];
            sprintf(broad_message, "用户%d: %s   [%s]", client_id, buffer,
time_str);
            broadcast_message(client_id, broad_message);
        }
```

```
    }

    // 从客户端数组中移除断开连接的客户端
    for (int i = 0; i < client_count; ++i)
    {
        if (clients[i].socket == client_socket)
        {
            memmove(&clients[i], &clients[i + 1], sizeof(struct ClientInfo) *
(client_count - i - 1));
            --client_count;
            break;
        }
    }

    closesocket(client_socket);
    return 0;
}
```

主函数初始化Winsock，如果初始化失败，会输出错误信息并返回 `-1`。创建一个 IPv4 的 TCP 套接字。如果创建失败，会输出错误信息并返回 `-1`。

```
//windows上开启网络权限
WSADATA wsaData;
if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0)
{
    printf("开启网络权限失败 %d\n", GetLastError());
    return -1;
}
// 1.创建socket套接字
SOCKET listen_socket = socket(AF_INET, SOCK_STREAM, 0);
if (INVALID_SOCKET == listen_socket) //INVALID_SOCKET = -1
{
    printf("创建socket失败 errcode: %d\n",GetLastError());
    return -1;
}
```

给socket绑定端口号，设置监听地址和端口，并将其绑定到之前创建的套接字上。如果绑定失败，会输出错误信息，关闭套接字，并返回 `-1`。

```
//2.给socket绑定端口号
struct sockaddr_in local = { 0 };

local.sin_family = AF_INET;
local.sin_port = htons(8080);//中间设备使用的是大端序（路由器）
//local.sin_addr.s_addr = INADDR_ANY; //服务器 选项 网卡 127.0.0.1（本地环回）只接受哪
个网卡的数据 一般写全0地址表示全部都接受
local.sin_addr.s_addr = inet_addr("0.0.0.0"); //字符串IP地址转换成整数IP

if (-1 == bind(listen_socket, (struct sockaddr*)&local, sizeof(local)))
{
    printf("绑定端口号失败 errcode: %d\n", GetLastError());
    closesocket(listen_socket);
    return -1;
}
```

给socket开启监听属性，将套接字设置为监听模式，并允许最多 10 个待连接请求排队。如果开启监听失败，会输出错误信息，关闭套接字，并返回 `-1`。

```
//3. 给socket开启监听属性
if (-1 == listen(listen_socket, 10))
{
    printf("开启监听失败 errcode: %d\n", GetLastError());
    closesocket(listen_socket);
    return -1;
}
```

使用 `accept` 函数等待客户端连接。如果连接失败，则继续循环等待下一个连接。每次有新客户端连接时，为其分配一个唯一的 ID，将客户端的套接字和 ID 存储到全局数组 `clients` 中。为每个新连接创建一个新的线程来处理该客户端的消息。如果创建线程失败，则输出错误信息，释放内存，关闭套接字，并继续循环等待下一个连接。当程序结束时，关闭监听套接字，调用 `WSACleanup` 清理 Winsock 资源。

```
    // 4.等待客户端连接
    //返回的客户端socket才是跟客户端可以通讯的一个socket
    // accept是阻塞函数，等待有客户端连接进来就接受连接，然后返回，否则一直阻塞
    while(1)
    {
        SOCKET client_socket = accept(listen_socket, NULL, NULL);
        if (INVALID_SOCKET == client_socket) //连接失败
        {
            continue;
        }
        int client_id = client_count; // 分配当前客户端的ID

        SOCKET* sockfd = (SOCKET*)malloc(sizeof(SOCKET)*2);
        sockfd[0] = client_socket;
        sockfd[1] = client_id;

        // 将新的客户端套接字添加到数组中
        if (client_count < MAX_CLIENTS)
        {
            clients[client_count].socket = client_socket;
            clients[client_count].id = client_id;
            ++client_count;
        }

        // 创建接收线程
        HANDLE hReceiveThread = CreateThread(NULL, 0, receive_thread_func,
sockfd, 0, NULL);
        if (hReceiveThread == NULL)
        {
            printf("创建线程失败 %lu\n", GetLastError());
            free(sockfd);
            closesocket(client_socket);
            continue;
        }
    }
    closesocket(listen_socket);
    WSACleanup();
    return 0;
```

```
}
```

（2）客户端代码 client.c

引入头文件库文件，定义常量和结构体，初始化全局变量，同服务器端代码。

```
#include<stdio.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>
#include<winSock2.h>
#pragma comment(lib,"ws2_32.lib")

#define MAX_CLIENTS 100 // 最大客户端数量

// 全局变量用于存储所有客户端的套接字和ID
struct ClientInfo {
    SOCKET socket;
    int id;
};

// 初始化全局变量
struct ClientInfo clients[MAX_CLIENTS];
int client_count = 1; // 初始化用户ID
```

recv_thread_func为接收线程函数，这个函数在一个单独的线程中运行，负责接收来自服务器的消息，并将其打印出来。

```
DWORD WINAPI recv_thread_func(LPVOID lpThreadParameter)
{
    int* threadParams = (int*)lpThreadParameter;
    int client_socket = threadParams[0];
    int client_id = threadParams[1];  // 获取客户端ID

    // 输出欢迎信息
    char welcome_message[1024];
    sprintf(welcome_message, "欢迎进入聊天室", client_id);
    printf("%s\n", welcome_message);

    while (1)
    {
        char rbuffer[1024] = { 0 }; // 接受缓冲区
        int ret = recv(client_socket, rbuffer, 1024, 0);
        if (ret <= 0)
        {
            break;
        }
        printf("%s\n", rbuffer);
    }
    printf("Receive thread: socket %llu, disconnect.\n", client_socket);
    return 0;
}
```

send_thread_func函数为发送线程函数，这个函数在一个单独的线程中运行，负责从标准输入读取用户输入，并将其发送给服务器。

```
DWORD WINAPI send_thread_func(LPVOID lpThreadParameter)
{
    int* threadParams = (int*)lpThreadParameter;
    int client_socket = threadParams[0];
    int client_id = threadParams[1]; // 获取客户端ID

    while (1)
    {
        char sbuffer[1024] = { 0 }; // 发送缓冲区
        //printf("please enter: ");
        fgets(sbuffer, 1024, stdin); // 使用 fgets 来读取带有换行符的输入

        // 移除换行符
        sbuffer[strcspn(sbuffer, "\n")] = 0;

        // 检查用户是否输入了 "quit"
        if (strcmp(sbuffer, "quit") == 0)
        {
            send(client_socket, "QUIT", 4, 0); // 发送退出信号
            printf("您已结束聊天！\n");
            break;
        }

        // 检查是否为私聊消息
        if (strncmp(sbuffer, "PRIVATE:", 8) == 0)
        {
            // 发送私聊消息
            send(client_socket, sbuffer, strlen(sbuffer), 0);
        }
        else
        {
            // 发送用户输入的数据
            send(client_socket, sbuffer, strlen(sbuffer), 0);
        }
    }
    //printf("Send thread: socket %llu, disconnect.\n", client_socket);
    return 0;
}
```

初始化Winsock，创建 IPv4 的 TCP 套接字。如果创建失败，会输出错误信息并返回 `-1`。设置了服务器的地址和端口，并尝试连接服务器。如果连接失败，会输出错误信息，关闭套接字，并返回 `-1`。

```
WSADATA wsaData;
if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0)
{
    printf("开启网络权限失败 %d\n", GetLastError());
    return -1;
}

//1.创建socket套接字
SOCKET client_socket = socket(AF_INET, SOCK_STREAM, 0);
if (INVALID_SOCKET == client_socket) {
```

```
    printf("创建socket失败 \n");
    return -1;
}

//2.连接服务器
struct sockaddr_in target;
target.sin_family = AF_INET;
target.sin_port = htons(8080);
target.sin_addr.s_addr = inet_addr("127.0.0.1");

if (-1 == connect(client_socket, (struct sockaddr*)&target, sizeof(target)))
{
    printf("连接服务器失败 \n");
    closesocket(client_socket);
    return -1;
}
```

为当前客户端分配一个唯一的 ID，为接收线程和发送线程分配参数，并使用 `malloc` 动态分配内存，将当前客户端的信息（套接字和 ID）添加到全局数组 `clients` 中，并递增 `client_count`。

```
int client_id = client_count; // 分配当前客户端的ID

// 创建接收线程的参数
int* recv_threadParams = (int*)malloc(sizeof(int) * 2);
recv_threadParams[0] = (int)client_socket;
recv_threadParams[1] = client_id;

// 创建发送线程的参数
int* send_threadParams = (int*)malloc(sizeof(int) * 2);
send_threadParams[0] = (int)client_socket;
send_threadParams[1] = client_id;

// 将新的客户端套接字添加到数组中
if (client_count < MAX_CLIENTS)
{
    clients[client_count].socket = client_socket;
    clients[client_count].id = client_id;
    ++client_count;
}
```

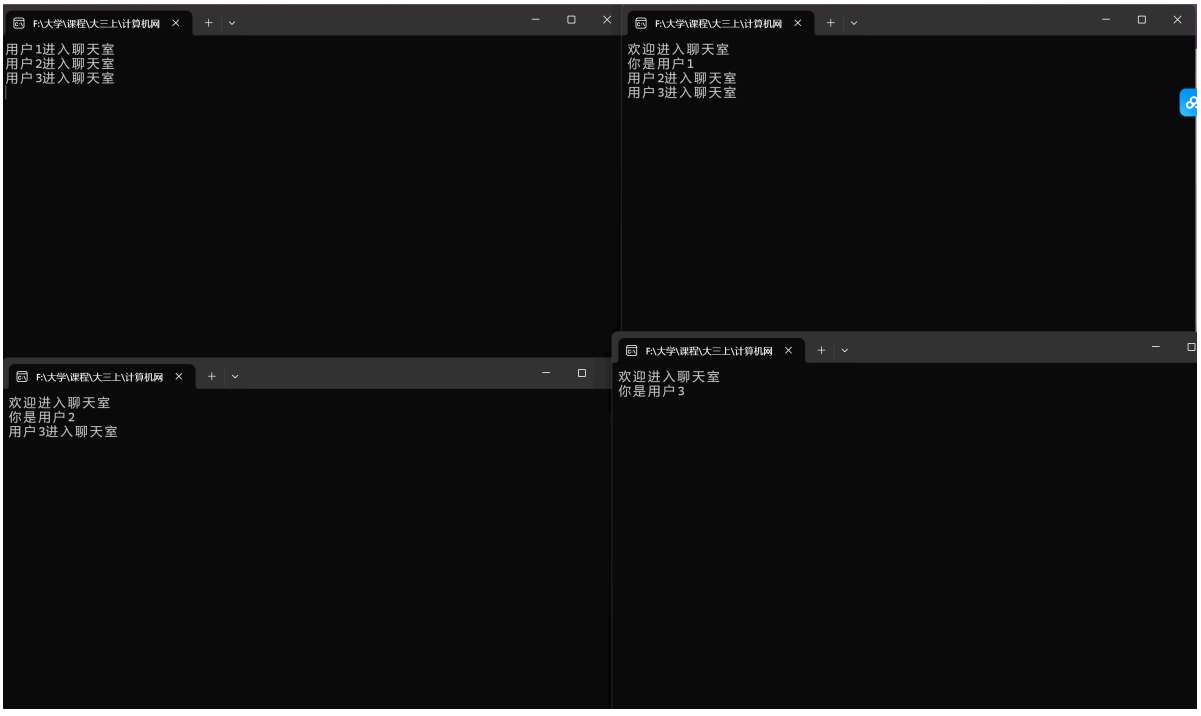
创建两个线程，一个用于接收服务器的消息，另一个用于向服务器发送消息。如果创建线程失败，会输出错误信息，关闭套接字，并返回 `-1`。等待接收线程和发送线程完成。`INFINITE` 表示无限等待，直到线程完成。关闭了接收线程和发送线程的句柄，并关闭了客户端套接字，最后清理了 Winsock 环境。

```
// 创建接收线程
HANDLE hRecvThread = CreateThread(NULL, 0, recv_thread_func, recv_threadParams,
0, NULL);
if (hRecvThread == NULL)
{
    printf("创建接收线程失败  %lu\n", GetLastError());
    closesocket(client_socket);
    return -1;
}

// 创建发送线程
```

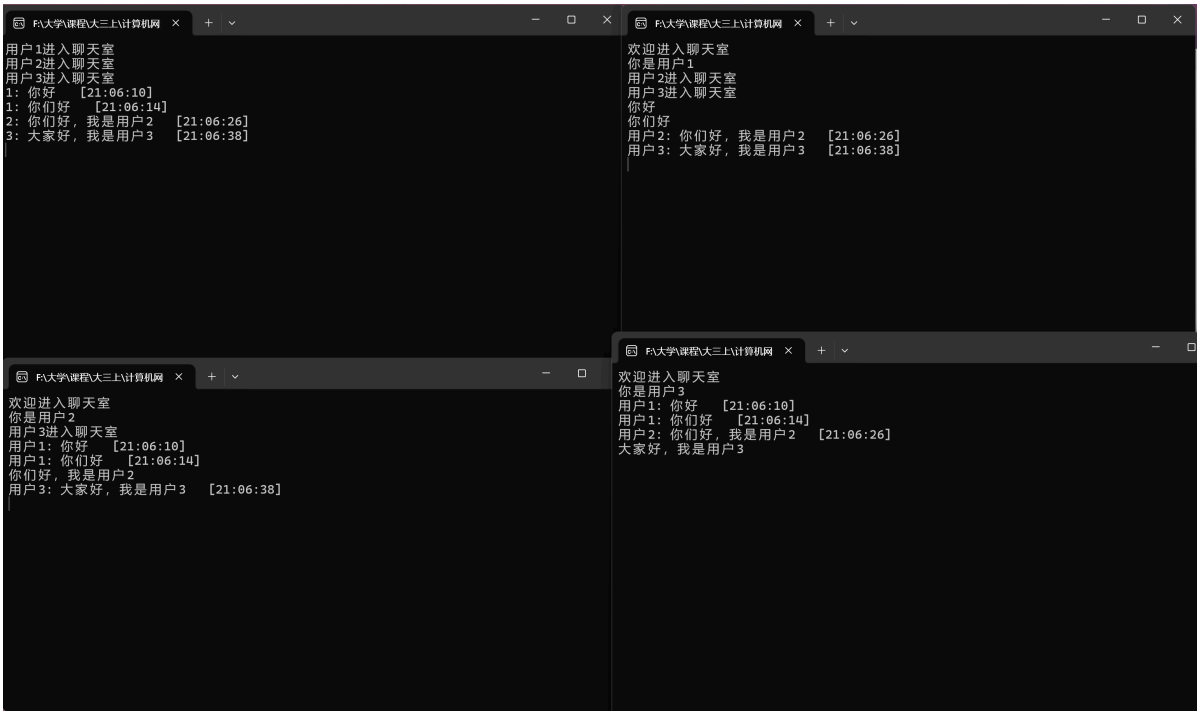
```
HANDLE hSendThread = CreateThread(NULL, 0, send_thread_func, send_threadParams,
0, NULL);
if (hSendThread == NULL)
{
    printf("创建发送线程失败 %lu\n", GetLastError());
    closesocket(client_socket);
    return -1;
}
// 等待两个线程完成
WaitForSingleObject(hRecvThread, INFINITE);
WaitForSingleObject(hSendThread, INFINITE);

// 关闭线程句柄
CloseHandle(hRecvThread);
CloseHandle(hSendThread);

//4.关闭连接
closesocket(client_socket);
WSACleanup();
```

3、程序界面展示及运行

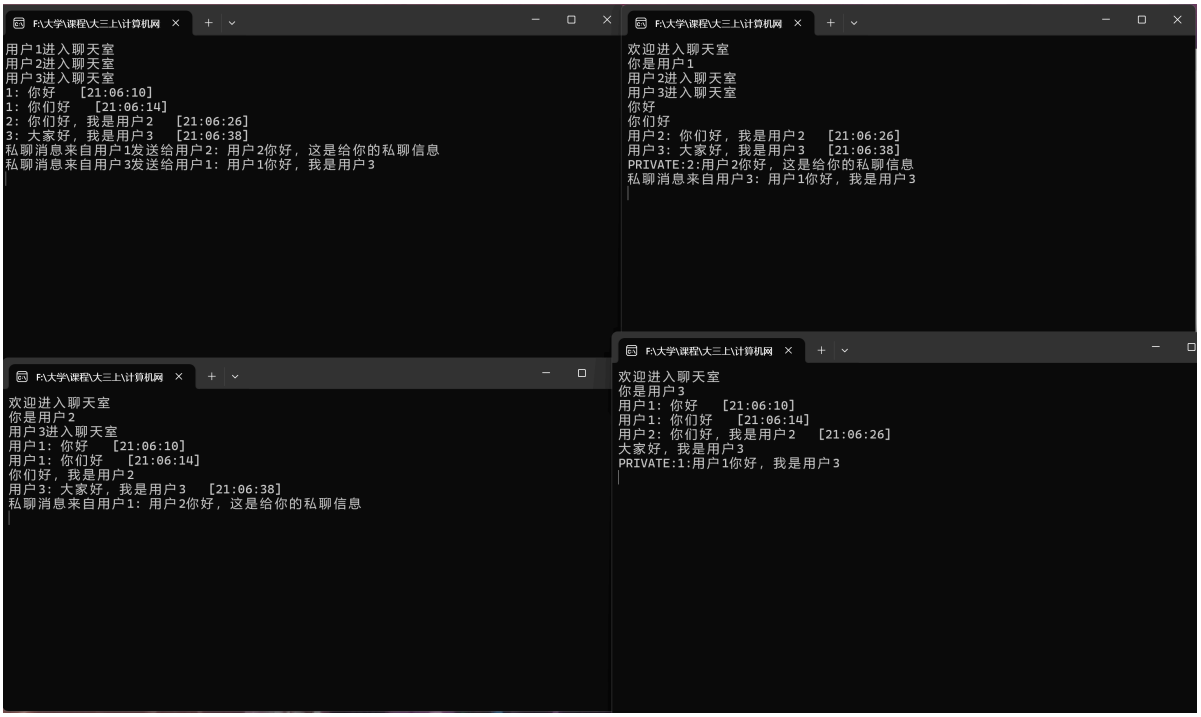
程序先开启服务器端，再开启客户端，否则客户端无法连接到服务端。成功启动时，服务器端会显示“用户x进入聊天室”，客户端显示“欢迎进入聊天室，你是用户x”，并且在新的客户端进入聊天室时，其余客户端也会提示“用户x进入聊天室”。



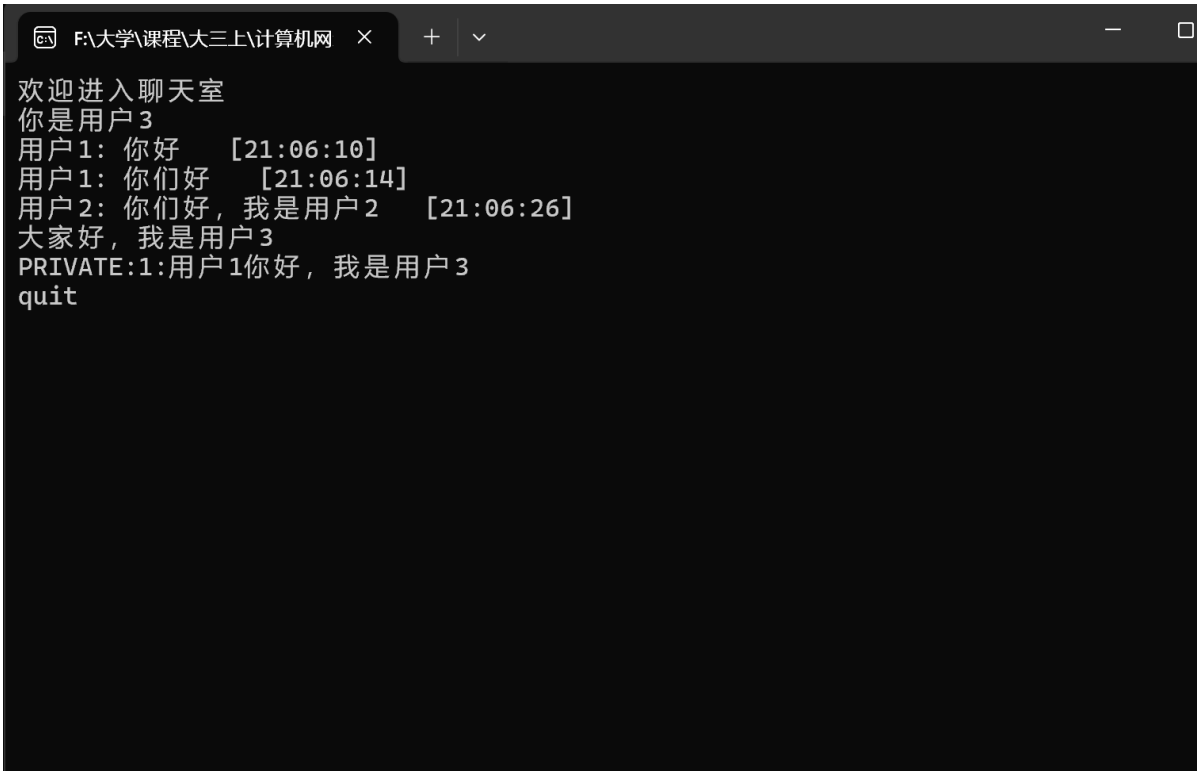
开启多人聊天，当在一个客户端发送信息时，信息会被发送到服务器，然后由服务器广播到其余客户端，显示“用户x：”以及发送的内容和发送时间。



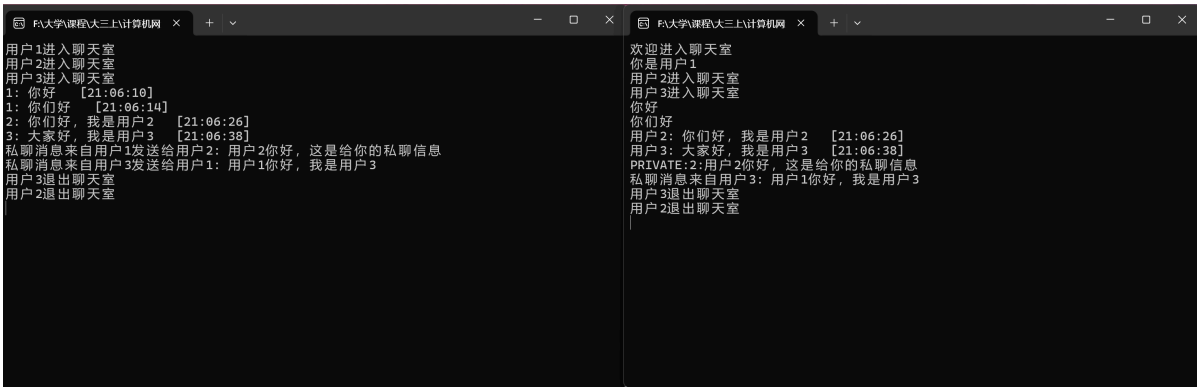
同时实现了私聊功能，当在客户端x输入“PRIVATE：id： "时，表示该客户端要发送私聊信息到用户id，此时服务器端显示“私聊信息来自用户x发送给用户id：”以及发送内容，在对应的客户端id内显示“私聊消息来自用户x：”以及发送内容，其余客户端不会收到私聊信息。



输出quit退出程序



在对应用户退出程序后，服务器端和其余客户端均会收到“用户x退出聊天室”。



4、实验过程中遇到的问题及分析

（1）在初步编写实验时，该程序在等待服务器响应时不能同时处理多个客户端，不能执行多个任务。我通过创建线程使程序能够在后台同时处理多个请求，创建线程可以让一部分程序专注于接收和显示服务器的信息，另一部分则处理用户的输入，可以保持用户界面的响应性。同时创建线程可以使接收数据和发送数据独立于主程序运行，不会阻塞其他操作。同时用户可以即时看到从服务器接收到的消息，提高了实时性。

（2）在添加用户client_id时，我起初会输出如-858993460这样的数，分析可知可能是因为 `client_socket` 被错误地转换为 `int` 类型，并且这个值被用作了 `client_id`。由于 `client_socket` 是一个 `SOCKET` 类型的变量，它可能包含一个负值，当它被转换为 `int` 类型时，这个负值可能会被截断，导致一个错误的正数值。为了解决这个问题，我修改SOCKET为int，并确保client_id是个正确非负的整数，并且在创建线程时传递正确的参数，并在线程完成后正确地等待和关闭线程。

（3）在进行私聊部分程序设计时，我的PRIVATE:client_id:后面输入的字符串无法被解析，经检查可知我应该使用 `sscanf` 函数解析私聊消息中的 `target_id` 和 `message`。`sscanf` 函数的返回值表示成功解析了多少个字段，如果返回值为 2，则说明成功解析了 `target_id` 和 `message`。如果 `sscanf` 返回值为 2，则发送私聊消息给目标用户；如果返回值不为 2，则打印错误信息或忽略该消息。通过以上修改后，服务端能够正确解析私聊消息中的 `target_id` 和 `message`，并仅发送给指定的目标用户。