计算机网络第一次实验报告

孙启森 2212422

一、消息协议

1. 消息的语法

- **消息内容**：消息由客户端发送至服务器，或者由服务器转发给其他客户端。
- **消息前缀**：每条消息前面都会加上一个前缀，例如 `[用户%u]` 表示发送者的身份，后面跟着消息内容。
  消息后缀：在最后添加消息发送的时间戳

```
snprintf(sendBuf, BUFFER_SIZE, "[用户%u]  %s  %s", GetCurrentThreadId(), recvBuf, buffer);
```

2.消息的语义

用户的id代表了用户的身份，然后是消息的具体信息，最后是消息的发送时间。

3.消息的处理

客户端在输入要发送的信息后就向服务器端执行发送，而服务器端收到报文后则会打印出来，并将其官博到其他的客户。

4.传输机制

- **TCP**：面向连接，可靠传输，流量控制
- **多线程**：每当有新的客户端连接时，服务器会创建一个新的线程来处理该客户端的请求。

5.会话管理

- **客户端列表**：服务器维护了一个 clientSockets 向量，用于存储所有连接的客户端套接字。
- **连接与断开**：客户输入exit会在客户端进行退出，同时服务器会将其从vector中删除。

```
clientSockets.push_back(sockConn);
clientSockets.erase(remove(clientSockets.begin(), clientSockets.end(), ClientSocket),
clientSockets.end());
```

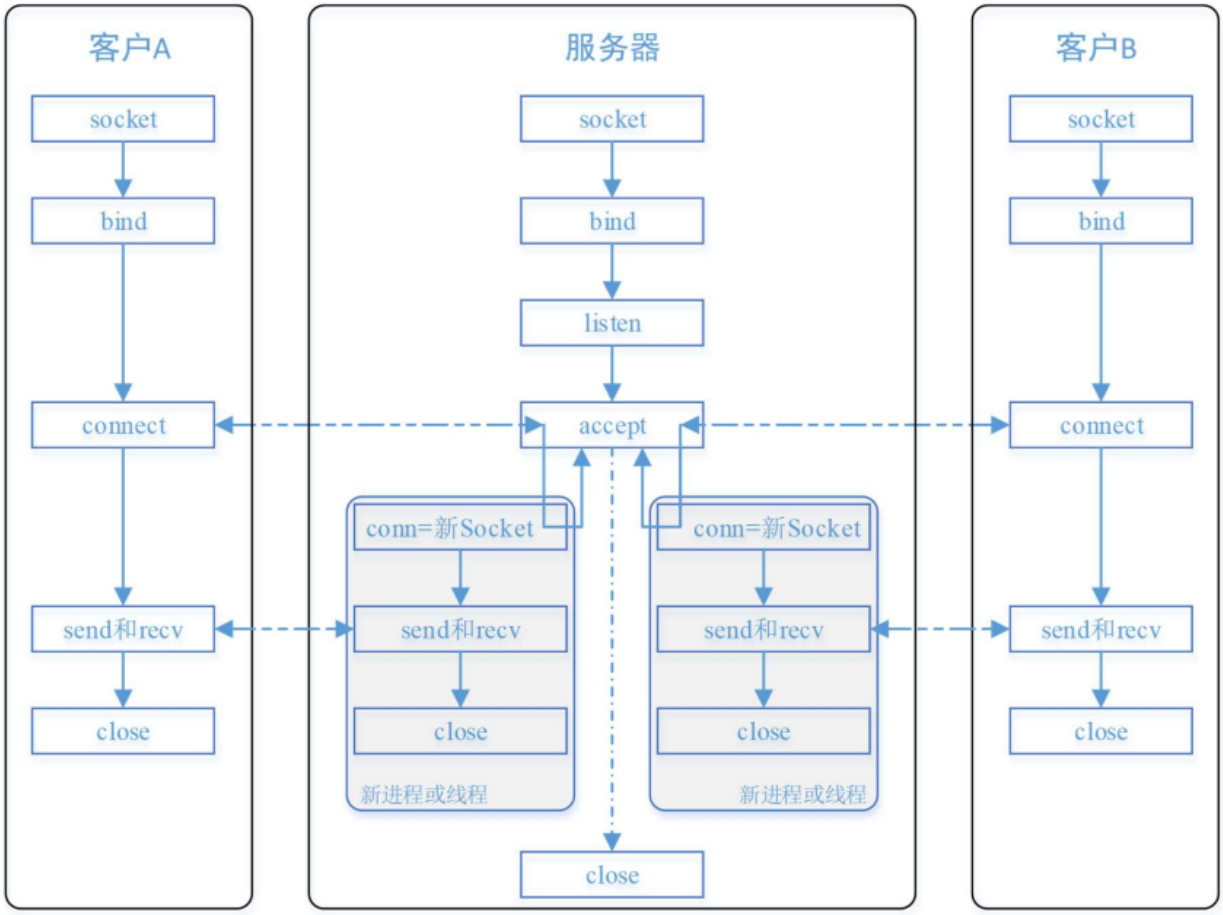
6.消息广播

当服务器收到客户的消息时，会按照格式打印在控制台，同时将其广播到其他客户。

7.错误处理

当接收或发送出现错误时，会进行输出提醒，同时断开连接并进行关闭。

二、功能实现和代码分析



本实验主要基于课件中所给的利用TCP编写应用程序的步骤来完成实验，如上即为所给的原理图。即用户端与服务器端端建立连接，服务器端建立多个线程与用户进行交互。

1.服务器端实现

(1).初始化并绑定socket

```
int result = 0;
	WORD wVersionRequested;
	WSADATA wsaData;
	wVersionRequested = MAKEWORD(2, 2);
	result=WSAStartup(wVersionRequested, &wsaData);
	if (result != 0)
	{
		cout << "出错了";
	}
	else
	{
		cout << "聊天室is加载中。。。。"<<endl;
	}

	SOCKET sockSrv = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
	if (sockSrv == INVALID_SOCKET)
	{
		cout << "出错了" << ",错误请看" << WSAGetLastError();
		WSACleanup();
	}
	sockaddr_in addrSrv;
	addrSrv.sin_family = AF_INET;
	const char* ip = "127.0.0.1";
	if (inet_pton(AF_INET, ip, &addrSrv.sin_addr) <= 0) {
		std::cerr << "inet_pton失败了 " << WSAGetLastError() << std::endl;
		closesocket(sockSrv);
		WSACleanup();
		return 1;
	}
	addrSrv.sin_port = htons(8088);
	bind(sockSrv, (SOCKADDR*)&addrSrv, sizeof(SOCKADDR));
```

在上面的代码中我们初始化Socket DLL，并返回信息判断是否初始化成功，接着初始化socket，采用AF_INET以及流式套接字和IPPROTO_TCP协议，并通过bind将其与对应的地址即127.0.0.1 端口8088绑定。从而完成了准备工作。

(2).监听和连接处理

```
while (1) {
                DWORD dwThreadId;
                int len = sizeof(addrClient);
                SOCKET sockConn = accept(sockSrv, (SOCKADDR*)&addrClient, &len);
                if (clientSockets.size() >= MAX_CLIENTS) {
                        char rejectMsg[BUFFER_SIZE] = "聊天室已满，无法连接。";
                        send(sockConn, rejectMsg, strlen(rejectMsg), 0);
                        closesocket(sockConn);
                        continue;
                }
                clientSockets.push_back(sockConn);
                HANDLE hThread = CreateThread(NULL, NULL, handlerRequest, LPVOID(sockConn), 0,
&dwThreadId);
                CloseHandle(hThread);
        }
```

在这里我们进行持续的监听操作，当未处理的请求排队超过三时就会进行拒绝。之后进入一个无限循环，当接受请求之后会创建一个新的线程，用于对该连接进行处理。同时设置了一个最大连接数，当超过数目时，就会拒绝连接。

(3).线程函数

```
DWORD WINAPI handlerRequest(LPVOID lparam)
{
	SOCKET ClientSocket = (SOCKET)(LPVOID)lparam;
	char sendBuf1[BUFFER_SIZE] = {};
	time_t current_timestamp = time(NULL);
	struct tm local_time;
	localtime_s(&local_time, &current_timestamp);
	char buffer[26];
	asctime_s(buffer, sizeof(buffer), &local_time);
	snprintf(sendBuf1, BUFFER_SIZE, "欢迎[用户%u]加入聊天室  %s", GetCurrentThreadId(), buffer);
	int sendResult = send(ClientSocket, sendBuf1, strlen(sendBuf1), 0);
	if (sendResult == SOCKET_ERROR) {
		printf("Send failed with error: %d\n", WSAGetLastError());
		closesocket(ClientSocket);
		return 1;
	}
	cout << sendBuf1<<endl;

	while (1)
	{
		char sendBuf[BUFFER_SIZE] = {};
		char recvBuf[BUFFER_SIZE] = {};
		int recvResult = recv(ClientSocket, recvBuf, BUFFER_SIZE - 8, 0);
		if (recvResult == SOCKET_ERROR) {
			printf("[用户%u]  %s  %s", GetCurrentThreadId(), "已退出",buffer);
			snprintf(sendBuf, BUFFER_SIZE, "[用户%u]  %s  %s", GetCurrentThreadId(), "已退出",
buffer);
			for (auto& sock : clientSockets)
			{
				if (sock != ClientSocket) // 不向发送者本身发送
				{
					int sresult = send(sock, sendBuf, strlen(sendBuf), 0);
					if (sresult == SOCKET_ERROR) {
						printf("Send failed with error: %d\n", WSAGetLastError());
						continue;
					}
				}
			}
			clientSockets.erase(remove(clientSockets.begin(), clientSockets.end(),
ClientSocket), clientSockets.end());
			break;
		}
		recvBuf[recvResult] = '\0';
		if (strcmp(recvBuf, "exit") == 0)
		{
			printf("[用户%u]  %s  %s", GetCurrentThreadId(), "已退出", buffer);
			snprintf(sendBuf, BUFFER_SIZE, "[用户%u]  %s  %s", GetCurrentThreadId(), "已退出",
buffer);
			for (auto& sock : clientSockets)
			{
				if (sock != ClientSocket) // 不向发送者本身发送
```

```
                                {
                                        int sresult = send(sock, sendBuf, strlen(sendBuf), 0);
                                        if (sresult == SOCKET_ERROR) {
                                                printf("Send failed with error: %d\n", WSAGetLastError());
                                                continue; // 或者采取其他措施
                                        }
                                }
                        }
                        clientSockets.erase(remove(clientSockets.begin(), clientSockets.end(),
ClientSocket), clientSockets.end());
                        break;
                }
                snprintf(sendBuf, BUFFER_SIZE, "[用户%u]  %s  %s", GetCurrentThreadId(), recvBuf,buffer);
                cout << sendBuf<<endl;
                if (recvResult > 0)
                {
                        for (auto& sock : clientSockets)
                        {
                                if (sock != ClientSocket) // 不向发送者本身发送
                                {
                                        int sresult = send(sock, sendBuf, strlen(sendBuf), 0);
                                        if (sresult == SOCKET_ERROR) {
                                                printf("Send failed with error: %d\n", WSAGetLastError());
                                                continue; // 或者采取其他措施
                                        }
                                }
                        }
                }
        }
        closesocket(ClientSocket);

        return 0;
}
```

在线程函数中则相当于服务器对于每个连接的处理，设置了发送以及接收的缓冲区，并设置大小为buffersize在连接之后首先会向客户端发送“欢迎进入”，从而告诉客户端已经成功连接。在这之后为客户分配一个id，接着进入一个无限循环，持续从客户端接受数据，当检查到客户输入“exit”或者断联后，则会广播告知用户推出并关闭线程，同时将其从vector中删除。若平常情况则将接收的信息转换为id信息时间的格式，并将其广播到其他用户。

2.客户端

1.初始及发送

```
int main()
{
	int result = 0;
	WORD wVersionRequested;
	WSADATA wsaData;
	wVersionRequested = MAKEWORD(2, 2);
	result=WSAStartup(wVersionRequested, &wsaData);
	if (result != 0)
	{
		cout << "出错了";
	}
	else
	{
		cout << "聊天室加载中"<<endl;
	}
	SOCKET sockClient = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
	sockaddr_in addrSrv;
	addrSrv.sin_family = AF_INET;
	inet_pton(AF_INET, "127.0.0.1", &addrSrv.sin_addr); // 服务器 IP 地址
	addrSrv.sin_port = htons(8088);
	result = connect(sockClient, (SOCKADDR*)&addrSrv, sizeof(addrSrv));
	if (result == SOCKET_ERROR)
	{
		cout << "出错了: 连接失败，错误码: " << WSAGetLastError() << endl;
		closesocket(sockClient);
		WSACleanup();
		return 1;
	}
	cout << "连接成功，让我们开始聊天吧" << endl;
	HANDLE hThread = CreateThread(NULL, NULL, handlerRequest, LPVOID(sockClient), 0, 0);
	char sendbuf[BUFFER_SIZE] = {};

	while (1)
	{
		cin.getline(sendbuf, BUFFER_SIZE);
		if (strcmp(sendbuf, "exit") == 0)
		{
			send(sockClient, sendbuf, strlen(sendbuf), 0);
			break;
		}
		int sendresult = send(sockClient, sendbuf, strlen(sendbuf), 0);
		if (sendresult == SOCKET_ERROR)
		{
			cout << "出错了: 发送数据失败，错误码: " << WSAGetLastError() << endl;
			break;
		}
	}
	closesocket(sockClient);
```

```
        WSACleanup();
        return 0;
}
```

在客户端中与服务器初始操作一样，首先初始化Socket DLL，接着创建新的socket并绑定到服务器地址，再将客户端socket与服务器端连接。连接成功后进入一个无限循环，输入要发送的信息，接着进行发送。在这里我们设置输入exit时就结束循环，接着执行一系列的关闭操作。

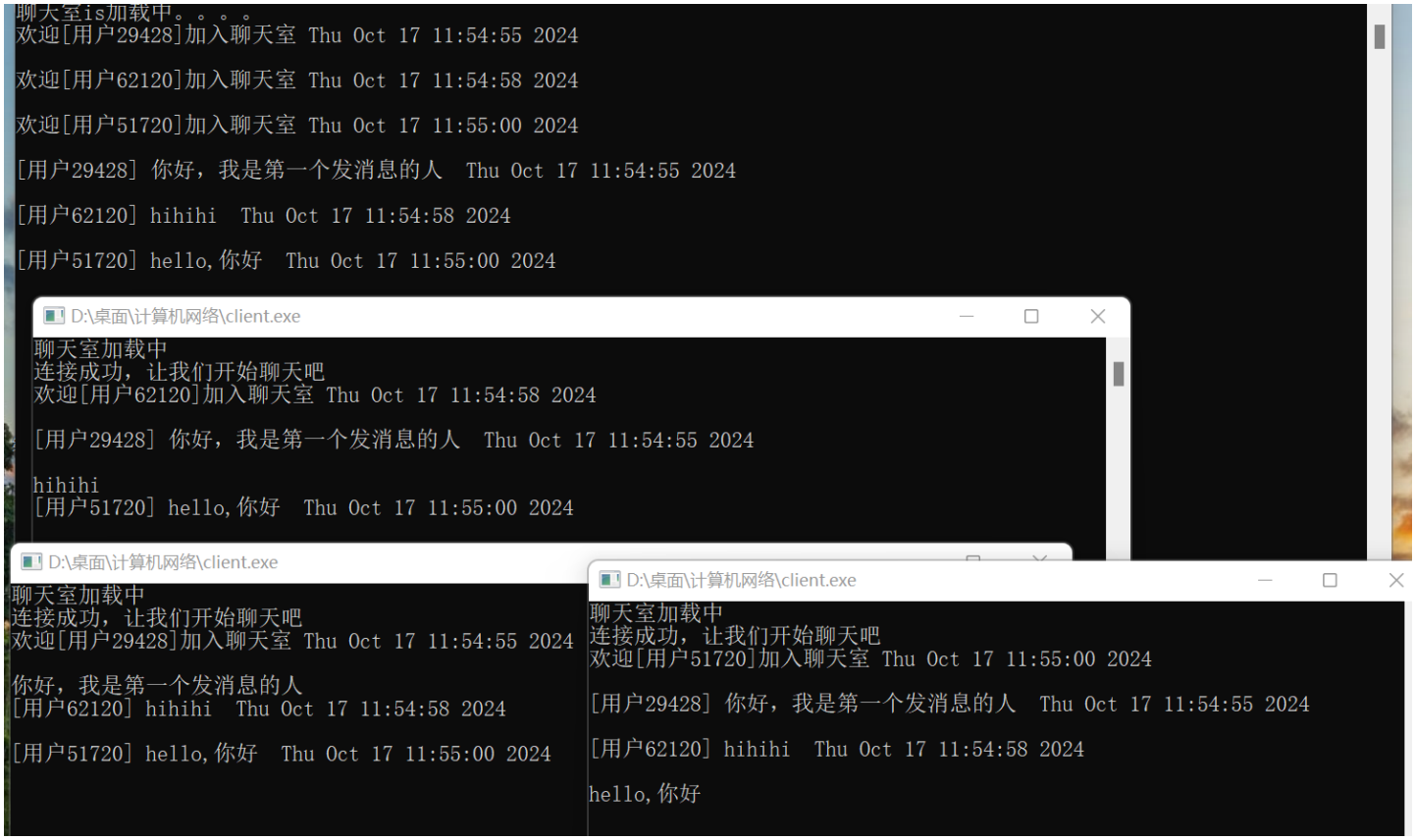
2.接收信息

```
DWORD WINAPI handlerRequest(LPVOID lparam)
{
        SOCKET ClientSocket = (SOCKET)(LPVOID)lparam;
        char recvBuf[BUFFER_SIZE]={};
        while (1)
        {
                int recresult = recv(ClientSocket, recvBuf, BUFFER_SIZE-8, 0);
                if (recresult > 0)
                {
                        recvBuf[recresult] = '\0';
                        cout << recvBuf << endl;
                }
                else if (recresult == 0)
                {
                        cout << "聊天室已关闭" << endl;
                        break;
                }
                else
                {
                        cout << "聊天室已被关闭" << endl;
                        break;
                }
        }
        closesocket(ClientSocket);
        return 0;
}
```

因为当接收和发送在一个线程中运行时会产生冲突，因此我们单独创建一个新的线程用于接收，与服务器端同理，当接收断联时就会跳出循环，接着结束线程。

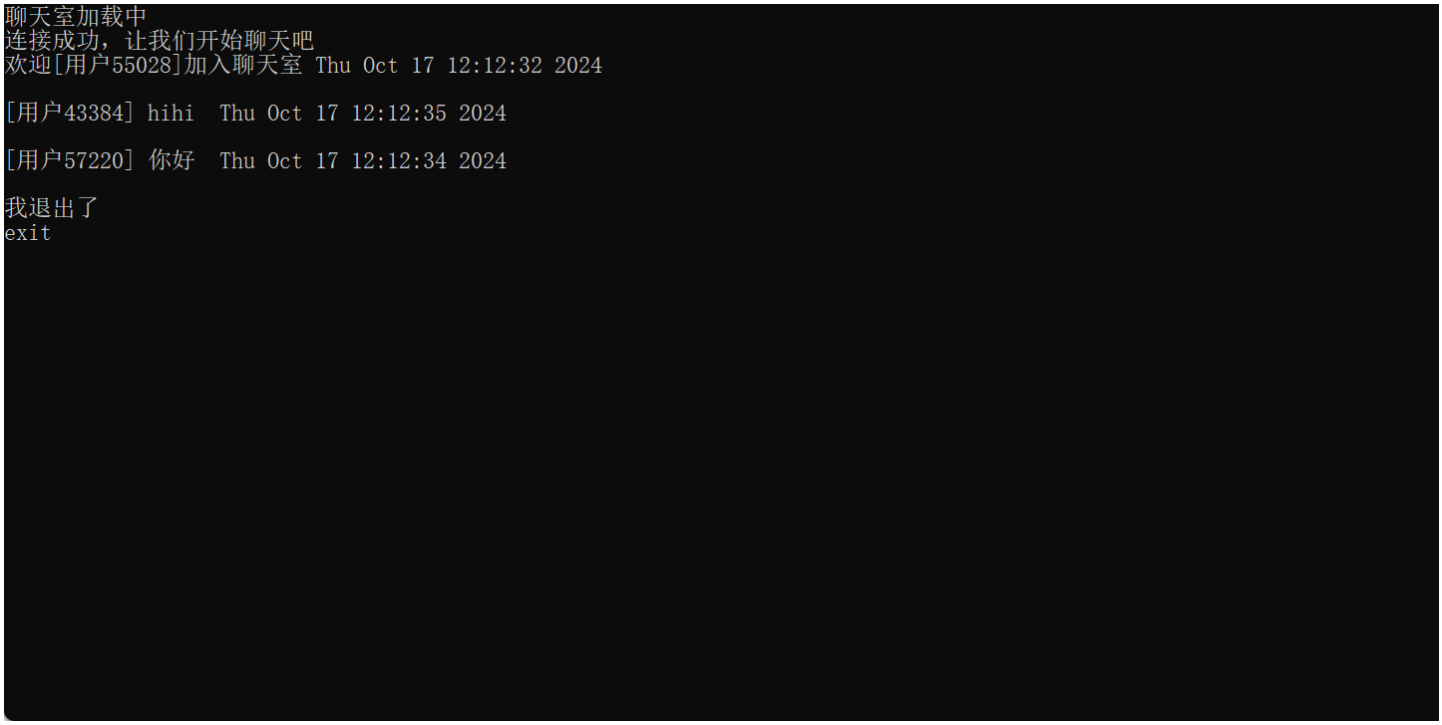
三、效果演示

1.运行演示

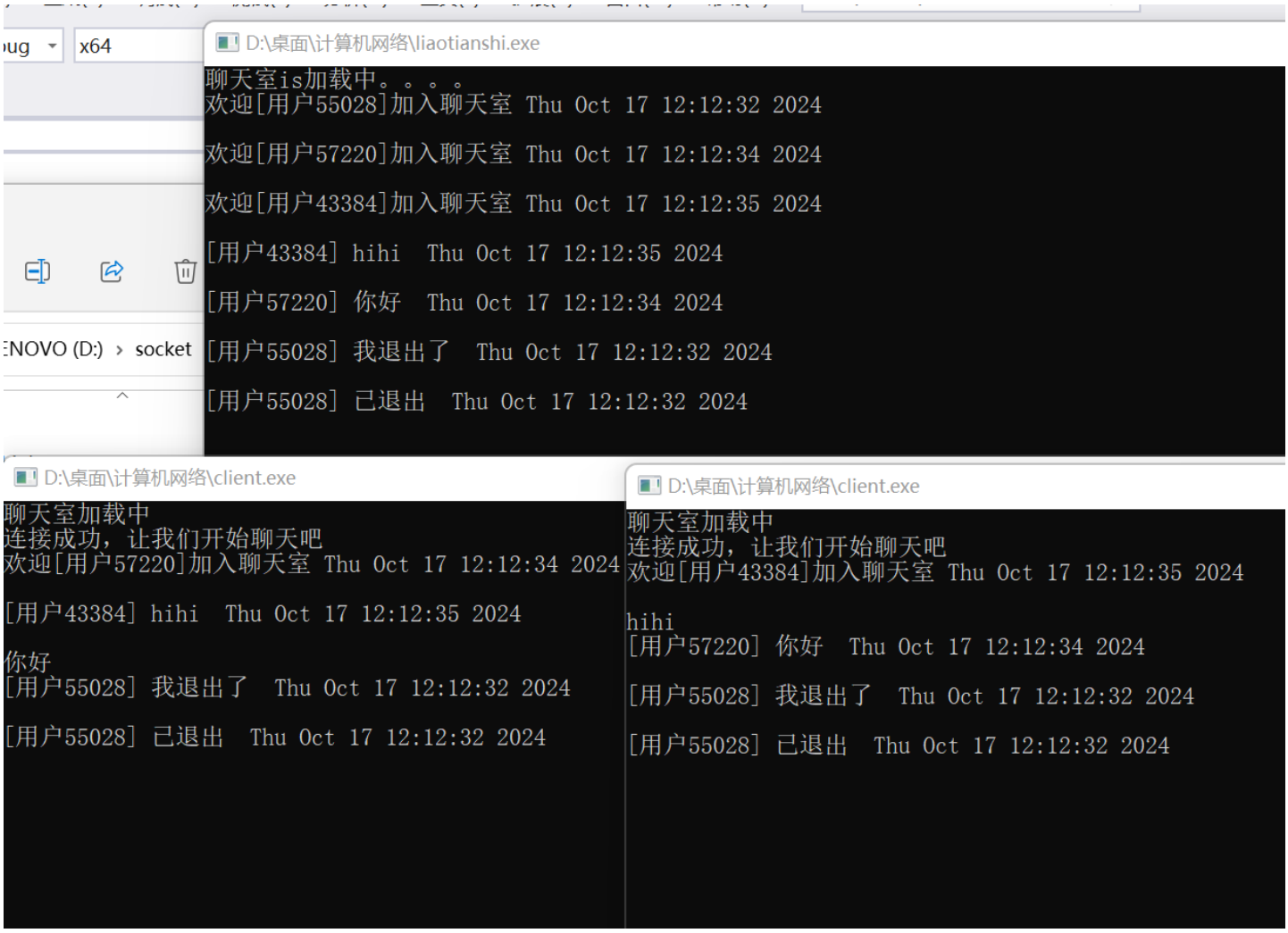


我们这里以三个人聊天为例，可以看到当用户进入后会接收到服务器的欢迎信息，同时不同用户发送信息会在服务器端进行格式化即增加用户名和时间戳，然后进行广播，同时可以看出，对于中英文都可以支持。

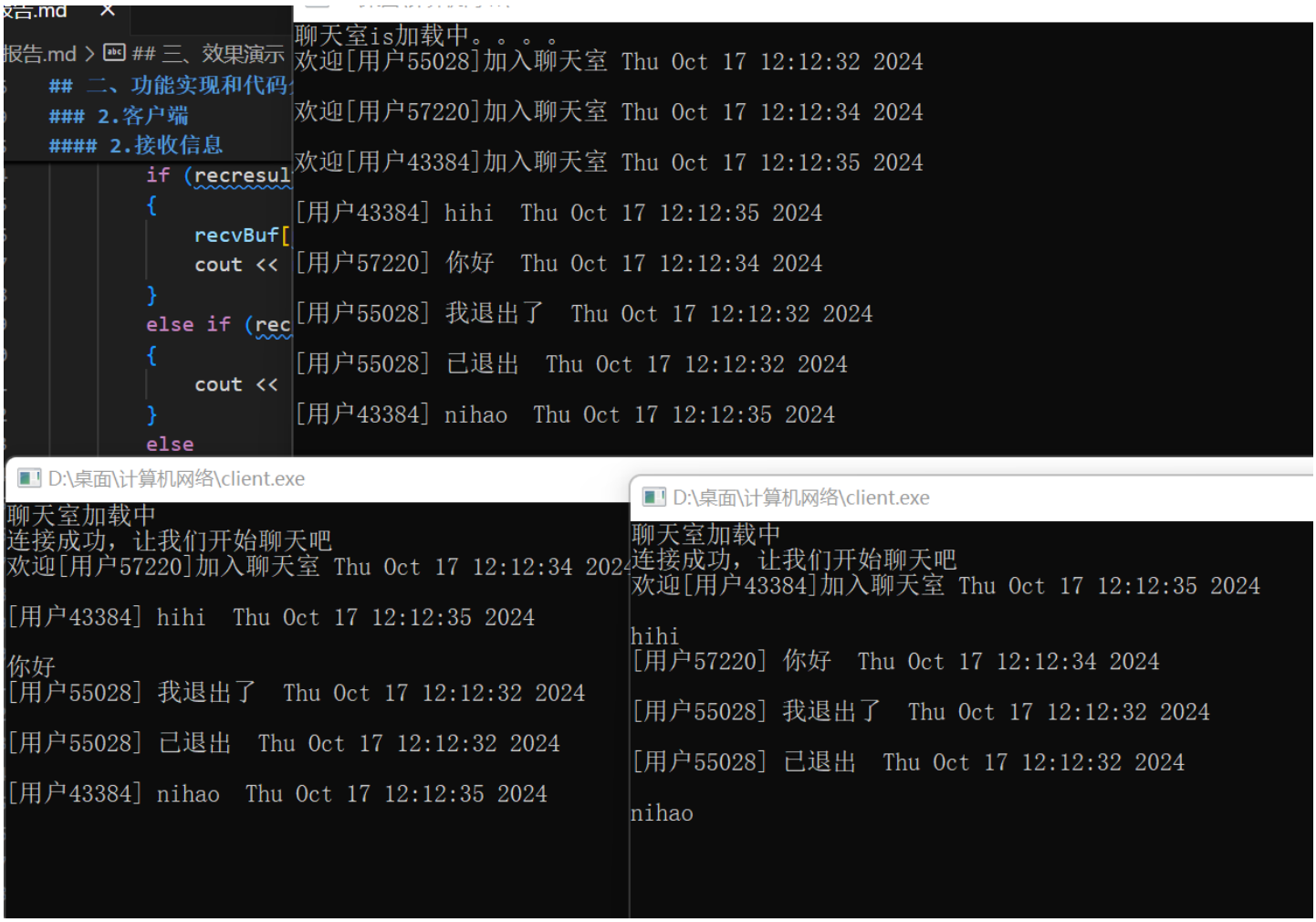
2.退出演示



可以看到输入了exit



接着该exe就结束了运行，同时服务器以及其他用户也收到了该用户退出的消息。



余下的用户则可以继续聊天，没有影响。

当服务器端发生异常而关闭时，则会告知用户聊天室已经被关闭

3.连接超出

可以看到当第四个尝试开启是，就被拒绝。

四、过程中的问题

1.用户端接收和发送的问题

在开始时，将接收和发送都放在主线程进行操作，但在实验过程中发现，当接收消息后如果没有进行输入就无法接收后续的信息。然后将接收和发送分两个循环进行，这样就会持续接收，进不去发送。
所以我们采用创建一个新的线程来专门用于接收信息，这样一个线程用来接收，一个线程用于发送，很好地解决了冲突问题。

2.send 和 rec的问题

在刚开始发送时，因为send会发送输入的字符串的大小，但接收时rec则更大，因此在输出时会将不满的地方填充为烫烫烫，因此我们在rec的字符结尾增加\0，从而解决烫烫烫的问题。

3.当服务器端关闭

在服务器端关闭时，发现客户端持续输出error，这是因为异常时没有执行break，因此持续地循环。发现这一点后将其他异常与结束同步地进行了完善。