多人聊天室

姓名：齐明杰　学号：2113997 班级：信安2班

1　聊天室简介

本实验旨在实现一个多人在线聊天室，使用户能够在实时环境中与其他人进行交流。聊天室由服务器端和客户端组成。服务器端负责管理客户端连接，广播消息，以及维护在线用户列表。客户端则负责与用户交互，发送和接收消息。

2　各模块功能

2.1　服务器模块：

定义了服务器类。由于是多人聊天，服务器需要同时记录多个客户的信息，也定义了客户的结构体，如下：

```
// 定义客户结构体
struct Client {
    SOCKET sock;                        // 套接字
    string username;                    // 用户名
    Client(SOCKET sock = INVALID_SOCKET, string username = "$") : sock(sock), username(username) {}
};

// 聊天室服务器类
class ChatRoomServer {
public:
    ChatRoomServer(UINT port = 12720, UINT client = 64);     // 构造函数
    ~ChatRoomServer();                                      // 析构函数
    void Start();                                           // 启动服务器
    void Stop();                                              // 关闭服务器
    void PrintInfo(const string& info);                       // 输出日志

private:
    // 定义服务器相关常量
    UINT MAX_CLIENTS;                                 // 最大客户端数量
    UINT PORT;                                      // 服务器端口
    constexpr static UINT BUFFER_SIZE = 1024;    // 缓冲区大小

    // 定义服务器相关变量
    SOCKET SockServer = INVALID_SOCKET;          // 服务器套接字
    Client* clients;                            // 客户端数组
    HANDLE* hThreads;       // 线程句柄，每个客户端均有一个线程来处理
    UINT hpointer = 0;                              // 线程句柄数组的指针
```

```
    sockaddr_in addrServer;                    // 服务器地址

    // 定义服务器相关函数
    void InitWinSock();                        // 初始化WinSock
    int find_pos();                            // 查找空闲的客户端存放位置
    UINT Online_Count();                       // 获取在线人数
    static DWORD WINAPI ClientHandler(LPVOID pParam);// 每个客户的线程函数
    void BroadcastMessage(const string& msg);    // 将消息广播给所有客户端
    string GetCurrTime();                        // 获取当前时间
};
```

核心函数及其功能:

1. `ChatRoomServer(UINT port = 12720, UINT client = 64)`：**构造函数**

功能：此函数初始化聊天室服务器，设置服务器的端口和最大客户端数量，并为客户端数组和线程句柄数组分配内存。

2. `~ChatRoomServer()`：**析构函数**

功能：在服务器对象生命周期结束时，此函数确保释放所有资源，包括动态分配的内存和线程。

3. `Start()`：**启动服务器**

功能：此函数初始化服务器套接字，绑定到指定端口，并开始监听客户端连接。对于每个连接的客户端，它都会启动一个新线程来处理该客户端的消息。它是服务器的主循环，首先**创建了一个套接字**，并尝试**绑定**到提供的端口。一旦成功，它就开始**监听**来自客户端的连接。对于每个连接，它都会查找一个空闲的位置来存储客户端信息，并**为该客户端启动一个新线程**。

4. `Stop()`：**关闭服务器**

功能：此函数关闭所有客户端连接，终止所有客户端线程，并关闭服务器套接字。确保服务器在适当的时候完全关闭并释放所有资源。

5. `BroadcastMessage(const string& msg)`：**广播消息**

功能：广播是一个聊天室的核心功能，因为它允许一个消息被发送给所有在线的用户。此函数简单地遍历客户端数组，并使用 `send()` 函数将消息发送给每个在线的客户端。

6. `ClientHandler(LPVOID pParam)`：**客户端处理线程**

功能：此函数是多线程的核心，因为每个连接的客户端都有自己的执行线程。线程首先读取客户端的用户名，然后循环接收并处理消息。如果客户端发送了一个 "exit" 消息，或者连接断开，线程就会终止。它循环接收来自客户端的消息，处理这些消息（例如，解析并广播消息），并在客户端断开连接时清理资源。

2.2 客户端模块

本次客户端我采用MFC可视化编程实现。

预览图如下：

在上方输入**服务端IP**，**服务端口**，然后输入自己的**用户名**，点击连接服务器即可连接服务端。

在聊天区会显示服务端发送来的信息，在下面编辑框输入点击发送即可发送信息到服务器，在聊天区显示。

点击退出，即可离开聊天区，此时还在聊天区的客户会收到离开信息。

部分重要函数变量声明如下：

```
afx_msg void OnBnClickedButtonExit();        // 退出按钮
afx_msg void OnBnClickedButtonSend();        // 发送按钮
afx_msg void OnBnClickedButtonConnect();     // 连接按钮
static constexpr UINT BufferSize = 1024;     // 缓冲区大小
virtual void OnClose();          // 重写关闭窗口函数
SOCKET SockClient = INVALID_SOCKET;      // 客户端套接字
HANDLE hThread = NULL;           // 线程句柄
CString UserName;                // 用户名
//打印消息
void PrintMsg(const CString& Name, const CString& strMsg);
//接收消息线程函数
static DWORD WINAPI ReceiveMessages(LPVOID pParam);
LRESULT OnUpdateChatMsg(WPARAM wParam, LPARAM lParam);  // 更新聊天消息
```

其中的几个关键函数：

1. `OnBnClickedButtonExit()`：**退出客户端**按钮点击事件

   功能：当用户点击退出按钮时，此函数会被触发。它会首先确认用户真的想要退出，然后关闭与服务器的套接字连接、终止消息接收线程，释放Winsock资源，并关闭聊天窗口。

2. `OnBnClickedButtonSend()`：**发送消息**按钮点击事件

   功能：此函数处理用户的消息发送请求。它首先检查消息内容的有效性，然后发送消息到服务器。如果发送失败，它会提醒用户，并允许用户重新设置连接的参数。

3. `OnBnClickedButtonConnect()`：**连接服务器**按钮点击事件

   功能：此函数处理用户的连接请求。它首先初始化Winsock、获取IP、端口和用户名，然后尝试与服务器建立连接。一旦连接成功，它会发送用户名给服务器并启动一个新线程来接收服务器的消息。

4. `ReceiveMessages(LPVOID pParam)`：**接收消息线程**函数

   功能：此函数在单独的线程中运行，不断地从服务器接收消息。一旦接收到消息，它会使用 `OnUpdateChatMsg` 函数将消息发送到主线程进行显示。

5. `OnUpdateChatMsg(WPARAM wParam, LPARAM lParam)`：更新聊天消息函数

   功能：这是一个消息处理函数，负责处理从 `ReceiveMessages` 线程发送来的消息，并在聊天窗口中显示它们。

6. `PrintMsg(const CString& Name, const CString& strMsg)`：打印消息函数

   功能：这个函数负责在客户端的用户界面上显示消息。它格式化并将消息追加到聊天窗口中。

7. `OnClose()`：关闭窗口函数

   功能：与 `OnBnClickedButtonExit()` 类似，当用户试图关闭聊天客户端窗口时，此函数被触发。它确保所有的资源都被适当地释放，并通知服务器客户端的退出。

3 协议设计

3.1 消息类型及结构

为确保服务器与客户端之间的通信顺畅且可靠，定义了以下消息协议：

- **Server 发送协议内容**:

在线用户数: users:online_count
有人进来：enter:username:time
有人离开：exit:username:time
聊天：chat:username:content:time

- **Client 发送协议内容**:

用户名选择： name:username
退出： exit:username:time
聊天： chat:username:content:time

- **服务器行为**:

`name`: 服务器解析后，为用户维护一个结构，记录其socket号和用户名。此外，服务器还会广播该用户进入的消息，发送 `enter` 协议。

`exit`: 服务器解析后，知道对应的客户端退出，关闭该套接字，并广播这个消息。

`chat`: 服务器接收到 `chat` 消息后，会直接**转发给所有在线的用户**。

- **客户端行为**:

`users`: 客户端本地解析后，获得在线用户数量，并显示在客户端界面上。

`enter`: 客户端解析消息，知道有新的用户进入聊天室，将这条信息添加到聊天记录中，并在界面上更新。

`exit`: 客户端解析消息，知道有用户离开了聊天室，将这条信息添加到聊天记录中，并在界面上更新。

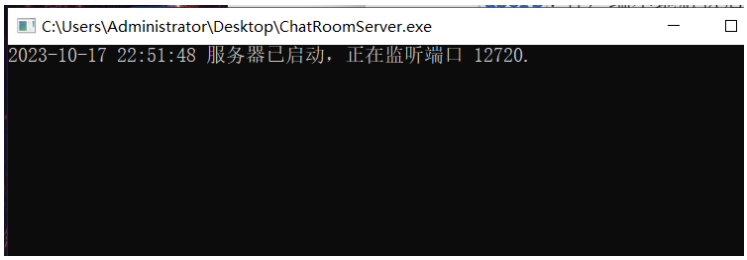
`chat`: 客户端接收到聊天消息后，将其显示在聊天记录中。

4 运行结果

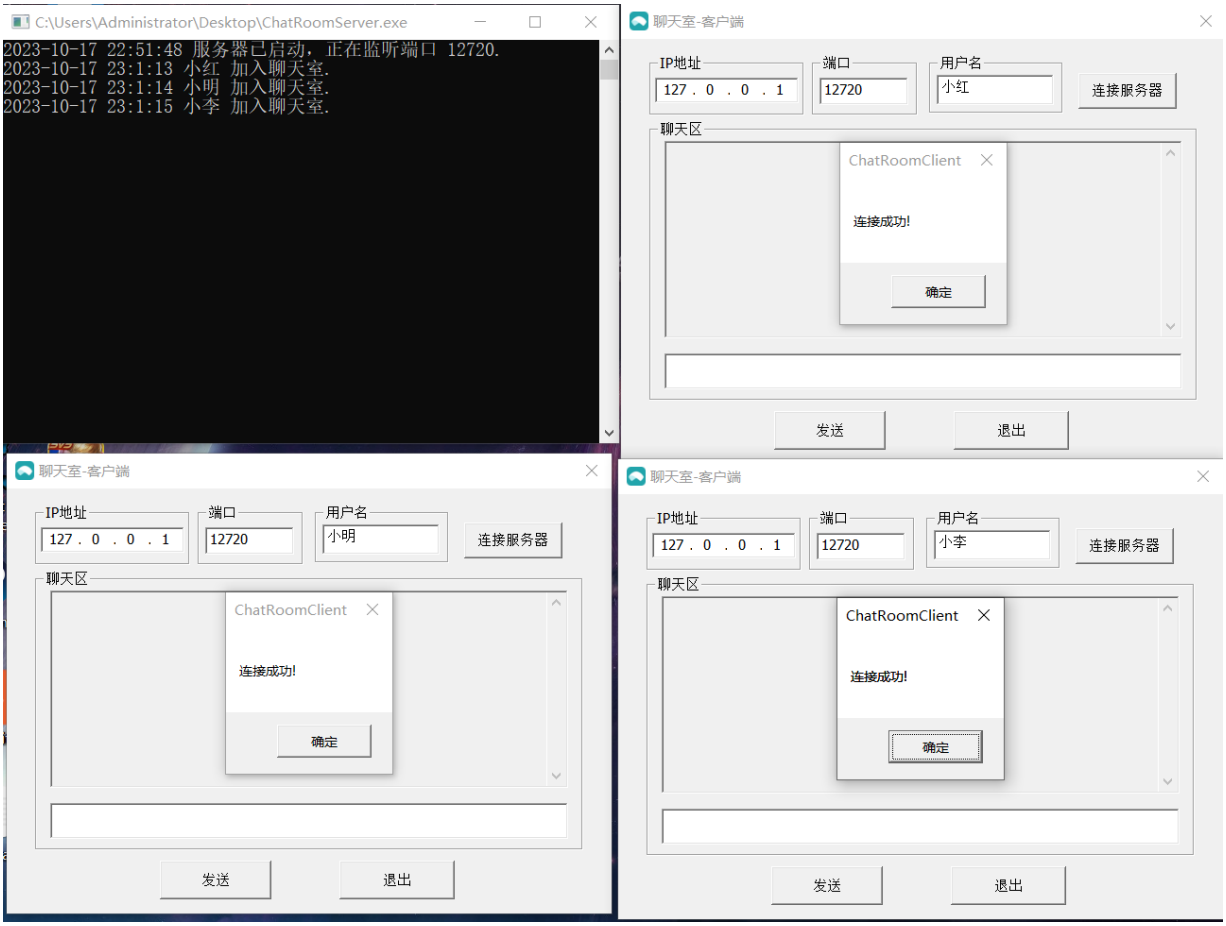
编译后，服务端可执行文件为`ChatRoomServer.exe`，客户端可执行文件为`ChatRoomClient.exe`，客户端可多开。

按以下步骤依次进行:

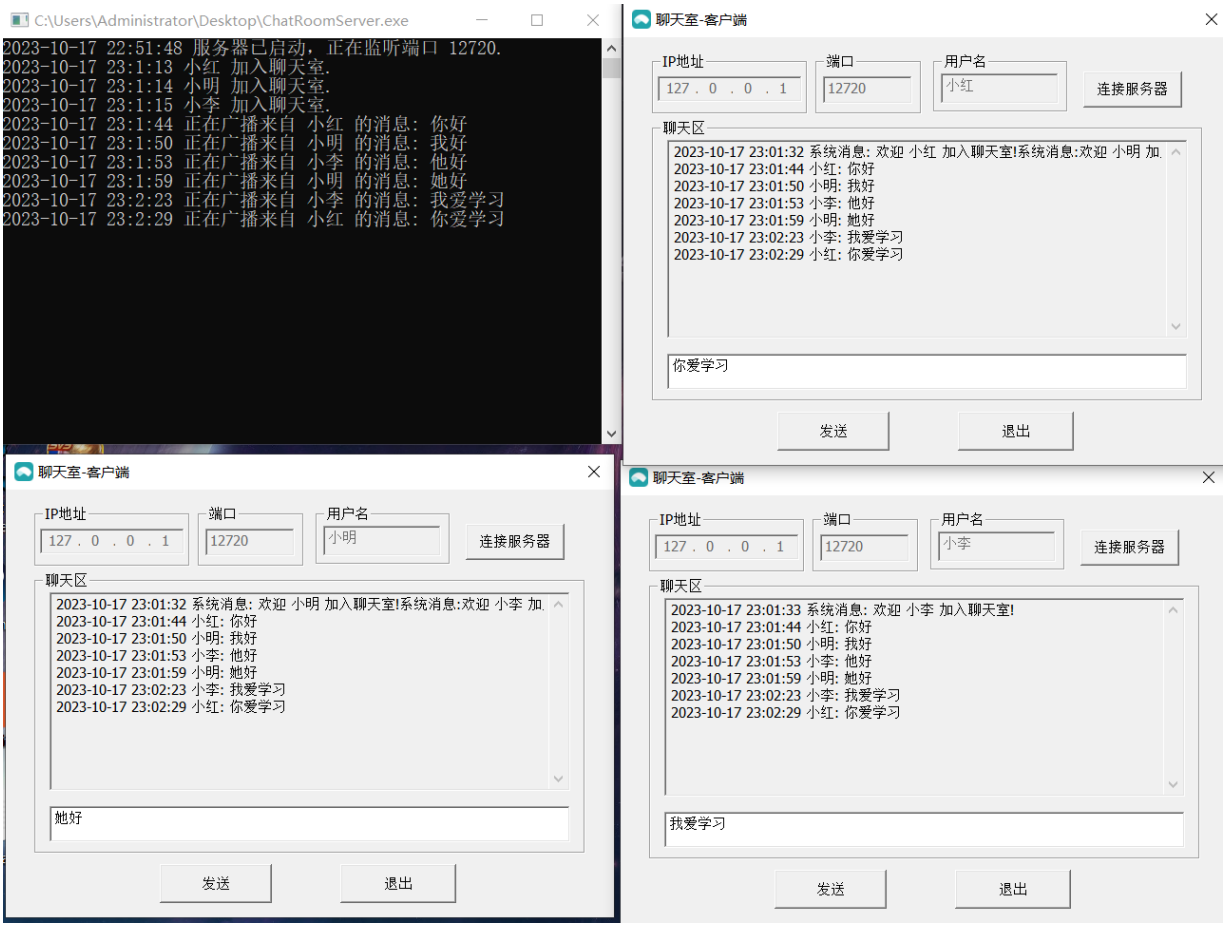
1. 打开服务端



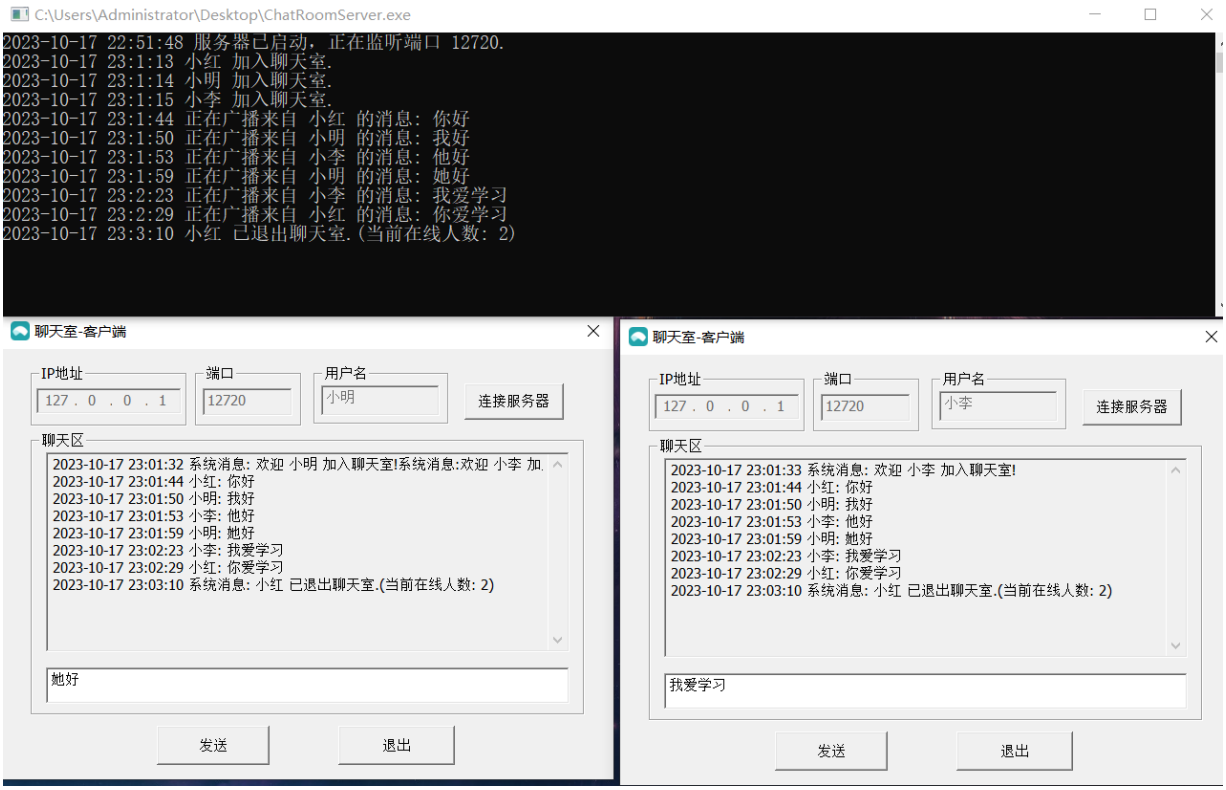
2. 打开若干个客户端(本次打开3个)，依次连接服务端

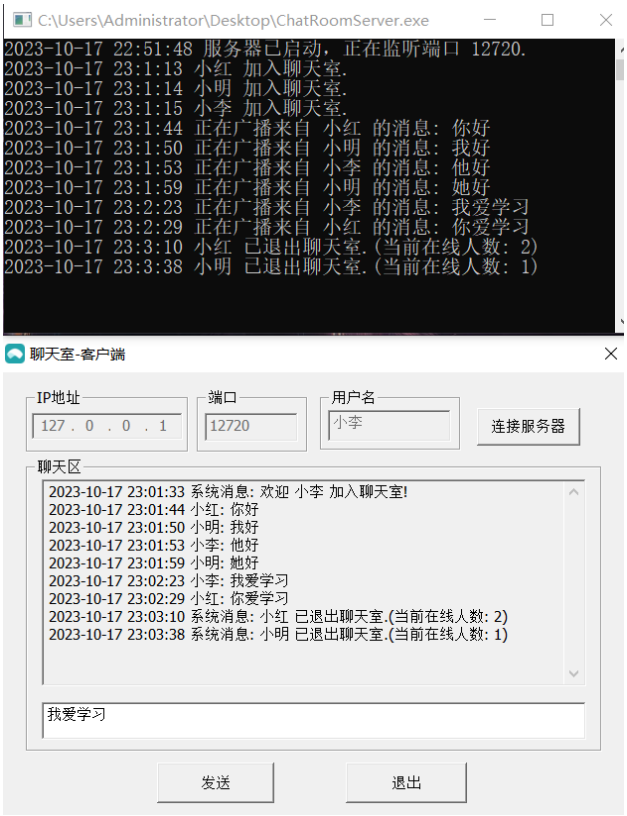


3. 客户端无顺序自由发言



4. 客户端退出



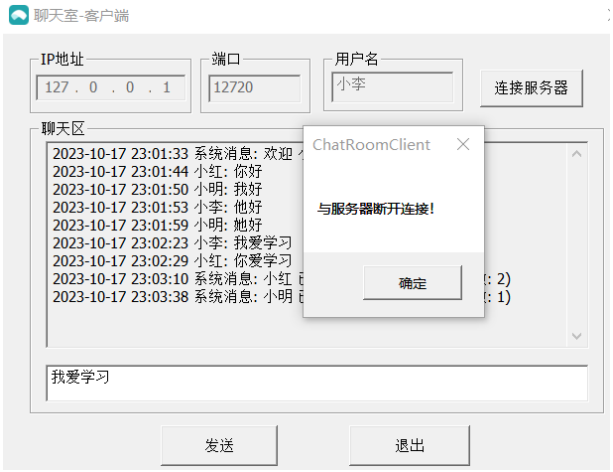


5. 服务端退出

直接关闭控制台窗口即可。

另：若客户端还没关闭就关闭服务端会怎么样?

答：因为客户端处于一个while(true)的循环，我的应用会立即提示客户端与服务端断开连接，需要重新连接输入IP端口连接服务器，不会卡死：



5 问题与思考

1. **线程安全**：在多线程环境中，对共享资源的并发访问可能导致数据竞争。需要考虑使用互斥锁或其他同步机制来确保线程安全。
2. **资源管理**：服务器为每个连接的客户端创建一个新线程可能会导致资源过度使用。考虑使用线程池来限制并发线程的数量。
3. **消息大小限制**：当前的实现限制了消息的最大大小，这可能不够灵活。考虑实现一个更复杂的消息接收逻辑。
4. **异常处理**：尽管已经处理了一些常见的错误和异常，但仍可能遇到其他问题，如网络中断或服务器崩溃。
5. **用户名唯一性**：当前的实现不检查用户名是否在服务器上已存在，可能会导致混淆。

为了解决上述问题并改进聊天室，可以考虑以下策略：

- 引入更复杂的协议设计，如 JSON 格式的消息，以支持更多的命令和功能。
- 为客户端和服务器之间的通信加密，以增强安全性。
- 添加用户身份验证机制，确保用户名的唯一性和安全性。