



南开大学
Nankai University

南 开 大 学

计 算 机 学 院

计算机网络实验报告

Lab 1 Socket 编程实践

徐俊智

年级：2022 级

专业：计算机科学与技术

指导教师：吴英

2024 年 10 月 17 日

目录

一、 实验目的	1
二、 实验要求	1
三、 聊天协议设计	1
(一) 通信模型	1
(二) 消息格式	2
(三) 消息传输	2
(四) 关闭连接	2
四、 实验内容	2
(一) 服务器端	3
(二) 客户端	4
五、 实验中遇到的问题	4
六、 程序运行演示	6

一、 实验目的

利用 Socket 编写一个聊天程序

二、 实验要求

1. 给出你设计的聊天协议的完整说明。
2. 利用 C 或 C++ 语言，使用基本的 Socket 函数完成程序，不允许使用 CSocket 等封装后的类编写程序。
3. 使用流式 Socket 完成程序。
4. 程序应该有基本的对话界面，但可以不是图形界面，还应该要有正常的退出方式。
5. 程序应该支持多人聊天，支持英文和中文信息。
6. 编写的程序应该结构清晰，具有较好的可读性。
7. 现场演示。
8. 提交程序源码、可执行代码和实验报告。

三、 聊天协议设计

(一) 通信模型

- 服务器启动：创建服务器套接字，给服务器绑定地址和端口，使服务器套接字处于监听状态，循环接收客户端的连接请求。
- 客户端连接：创建客户端套接字，客户端向服务器套接字发起连接请求，只有当服务器完成启动时，客户端才可以与之相连，否则连接失败。
- 服务器接收连接请求：服务器接受客户端的连接请求后，为每个客户端分配唯一的用户 ID。
- 服务器支持接收多个客户端连接请求
 - 服务器主线程广播用户进入聊天消息给所有客户端；
 - 服务器为每个客户端创建一个线程，并且使用新的连接套接字与客户端通信；
 - 服务器的线程处理函数负责将用户 ID 发送给客户端，并循环接收客户端消息。如果客户端发送聊天信息，就打印用户聊天信息并广播聊天消息给所有客户端；如果客户端请求退出，就打印用户退出聊天信息并广播消息给其他客户端；
- 消息传输
 - 客户端主线程发送消息到服务器。
 - 服务器的线程处理函数接收消息后，广播聊天消息给所有客户端。
 - 客户端创建线程来接收并处理服务器消息；
 - 客户端的线程处理函数负责循环接收并处理服务器消息，打印接收到的消息；
- 退出流程

- 关闭服务器，客户端自动断开连接。
- 客户端有三种退出方式，直接关闭窗口，发送“exit”或“quit”消息。
- 服务器的线程处理函数检测到退出请求后，断开与该客户端的连接，并广播退出消息给其他客户端。

(二) 消息格式

- 消息结构：“用户 [ID]: 消息内容”
- 消息类型：
 - 登录消息：“用户 [ID] 加入聊天! ”。用户连接服务器后即登录。服务器向客户端发送用户 ID 作为用户身份标识。
 - 聊天信息：“用户 [ID]: 消息文本”。消息文本支持中文和英文文本。
 - 退出信息：“用户 [ID] 退出聊天! ”。

(三) 消息传输

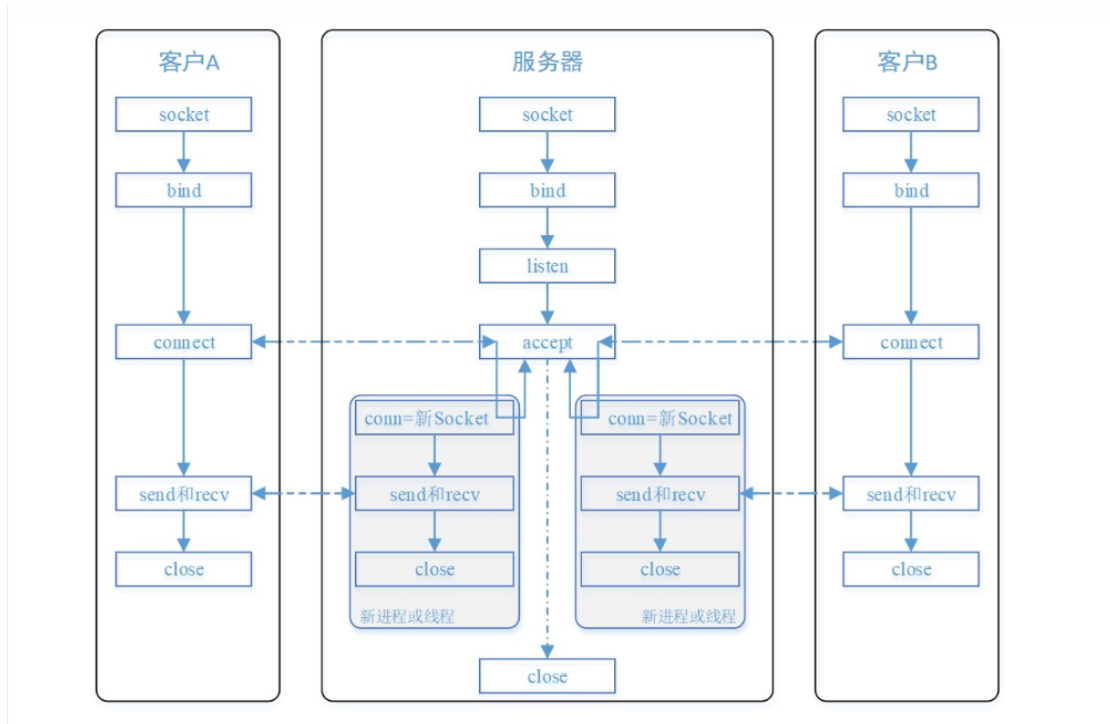
- 客户端主线程通过 send() 函数发送消息到服务器。
- 服务器的线程处理函数通过 recv() 函数接收消息后，通过 send 函数广播聊天消息给所有客户端。
- 客户端的线程处理函数通过 recv() 函数接收服务器消息，打印接收到的消息；

(四) 关闭连接

- 服务器主动关闭连接，通过 TCP 的四次挥手断开连接，客户端自动断开连接。
- 客户端有三种退出方式，直接关闭窗口，发送“exit”或“quit”消息。
- 服务器的线程处理函数检测到退出请求后，断开与该客户端的连接，并广播退出消息给其他客户端。

四、 实验内容

本次实验要实现流式套接字、TCP 协议的多人聊天室，包括服务器端和客户端：



(一) 服务器端

1. WSStartup() 函数初始化 Socket 库，协商使用的 Socket 版本
2. socket() 函数创建服务器 socket
3. bind() 函数给服务器绑定地址和端口
4. listen() 函数使服务器 socket 处于监听状态，准备接受客户端的连接请求
5. while (true) 循环接收客户端的连接请求
 - accept() 函数接受客户端的连接请求
 - acceptThread() 函数为每一个客户端连接创建线程
6. handleAccept() 函数接收并处理客户端消息
 - send() 函数发送用户 ID 给客户端
 - for (const auto& pair : accepts)
 - send() 函数发送用户进入聊天消息给所有客户端
 - while (true)
 - recv() 函数接受客户端消息
 - if (recvBytes <= 0 || message == "exit" || message == "quit")
 - * closesocket() 函数关闭该连接套接字 socket
 - * for (const auto& pair : accepts)
 - send() 函数发送退出消息给其他客户端
 - else
 - for (const auto& pair : accepts)
 - send() 函数发送消息给其他客户端

7. closesocket() 函数关闭服务器 socket
8. WSACleanup() 函数释放 socket 库资源

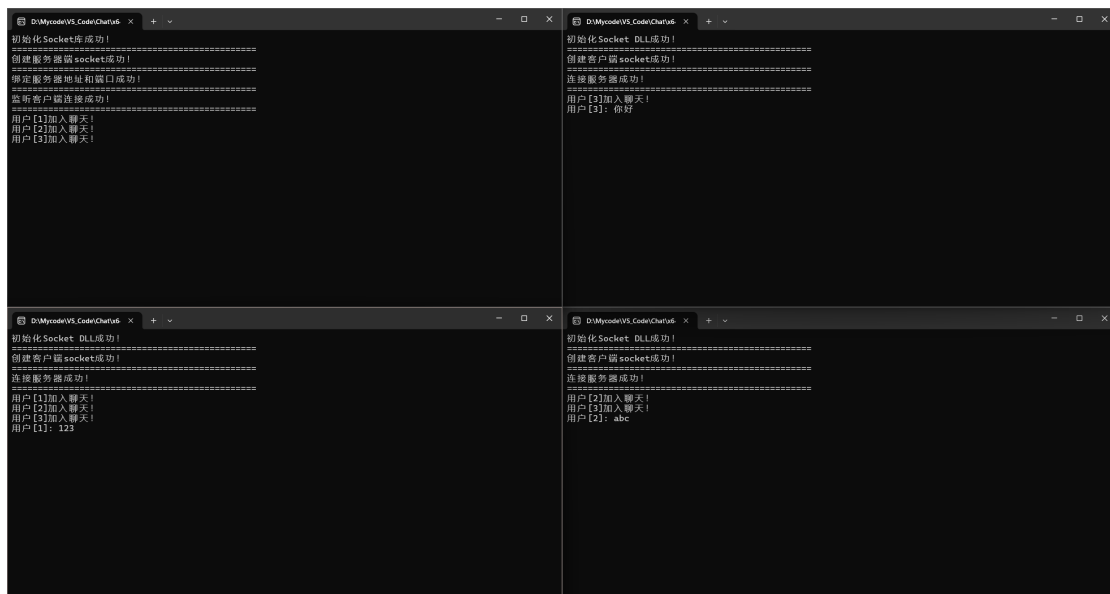
(二) 客户端

1. WSStartup() 函数初始化 Socket 库, 协商使用的 Socket 版本
2. socket() 函数创建客户端 socket
3. connect() 函数向服务器 socket 发起连接请求
4. recv() 函数并保存用户 ID
5. recvThread() 函数为客户端连接创建接收线程
6. while (true) 发送消息到服务器
 - cin.getline() 函数读取用户输入
 - send() 函数发送消息到服务器
7. handleServer() 函数接收并处理客户端消息
 - while (true)
 - recv() 函数接受服务器消息
8. closesocket() 函数关闭客户端 socket
9. WSACleanup() 函数释放 socket 库资源

五、 实验中遇到的问题

在测试的过程中, 我发现一个用户发送消息会覆盖掉其他所有用户还未发送的消息, 虽然用户未发送的消息会通过主线程的 cin.getline 函数存到发送缓冲区 sendBuff, 实际上不影响用户正常的键入和发送, 但对用户体验会造成极大的负面影响。

启动服务器, 运行三个客户端作为例子。客户端输入但不发送消息。



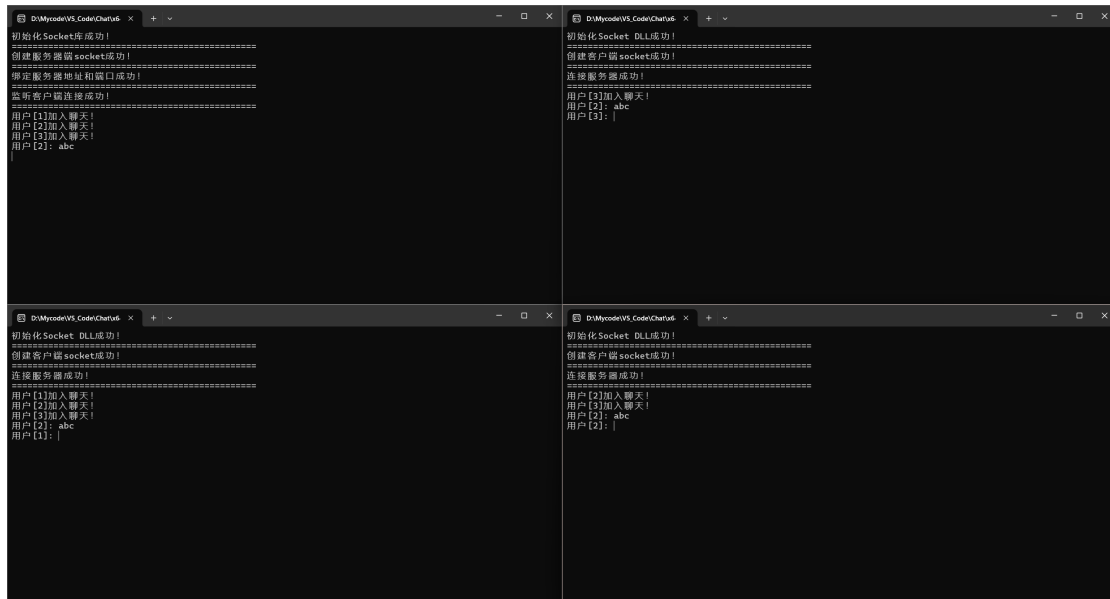
```
初始化Socket库成功!
创建服务器端socket成功!
绑定服务器地址和端口成功!
监听客户端连接成功!
用户 [1] 加入聊天!
用户 [2] 加入聊天!
用户 [3] 加入聊天!

初始化Socket DLL成功!
创建客户端socket成功!
连接服务器成功!
用户 [3] 加入聊天!
用户 [3]: 你好

初始化Socket DLL成功!
创建客户端socket成功!
连接服务器成功!
用户 [1] 加入聊天!
用户 [2] 加入聊天!
用户 [3] 加入聊天!
用户 [1]: 123

初始化Socket DLL成功!
创建客户端socket成功!
连接服务器成功!
用户 [2] 加入聊天!
用户 [3] 加入聊天!
用户 [2]: abc
```

用户 2 发送消息，我们可以看见用户 1 和 3 的输入都被覆盖掉了。



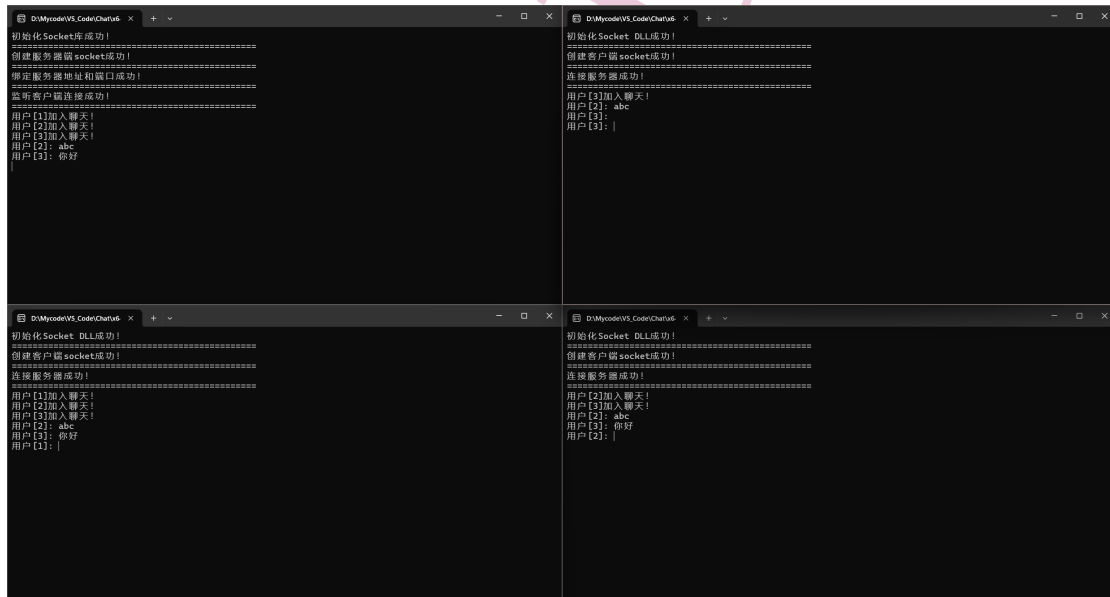
```
初始化Socket DLL成功!
创建服务端socket成功!
绑定服务端地址和端口成功!
监听客户端连接成功!
=====
用户 [1]加入聊天!
用户 [2]加入聊天!
用户 [3]加入聊天!
用户 [2]: abc
|

初始化Socket DLL成功!
创建客户端socket成功!
连接服务端成功!
=====
用户 [2]加入聊天!
用户 [2]: abc
用户 [2]: |

初始化Socket DLL成功!
创建客户端socket成功!
连接服务端成功!
=====
用户 [2]加入聊天!
用户 [2]: abc
用户 [2]: |

初始化Socket DLL成功!
创建客户端socket成功!
连接服务端成功!
=====
用户 [2]加入聊天!
用户 [2]: abc
用户 [2]: |
```

用户 3 直接回车，可以发送输入的内容。



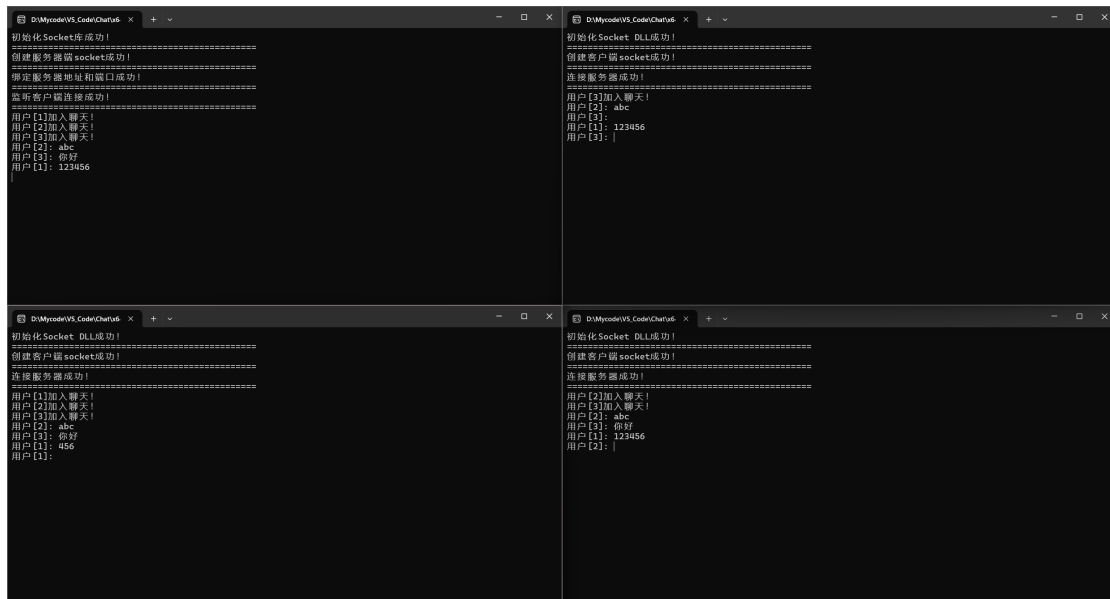
```
初始化Socket DLL成功!
创建服务端socket成功!
绑定服务端地址和端口成功!
监听客户端连接成功!
=====
用户 [1]加入聊天!
用户 [2]加入聊天!
用户 [3]加入聊天!
用户 [2]: abc
用户 [3]: 你好
|

初始化Socket DLL成功!
创建客户端socket成功!
连接服务端成功!
=====
用户 [2]加入聊天!
用户 [2]: abc
用户 [2]: |

初始化Socket DLL成功!
创建客户端socket成功!
连接服务端成功!
=====
用户 [2]加入聊天!
用户 [2]: abc
用户 [2]: |

初始化Socket DLL成功!
创建客户端socket成功!
连接服务端成功!
=====
用户 [2]加入聊天!
用户 [2]: abc
用户 [2]: |
```

用户 1 继续输入，回车后也可以发送输入的内容。



```
初始化Socket库成功!
创建服务器端socket成功!
绑定服务器地址和端口成功!
监听客户端连接成功!
=====
用户[1]加入聊天!
用户[2]加入聊天!
用户[3]加入聊天!
用户[2]: abc
用户[3]: 你好
用户[1]: 123456
=====

初始化Socket DLL成功!
创建客户端socket成功!
连接服务器成功!
=====
用户[3]加入聊天!
用户[2]: abc
用户[3]: 123456
用户[3]: |

初始化Socket DLL成功!
创建客户端socket成功!
连接服务器成功!
=====
用户[2]加入聊天!
用户[3]加入聊天!
用户[2]: abc
用户[3]: 你好
用户[1]: 456
用户[1]: |

初始化Socket DLL成功!
创建客户端socket成功!
连接服务器成功!
=====
用户[2]加入聊天!
用户[3]加入聊天!
用户[2]: abc
用户[3]: 123456
用户[2]: |
```

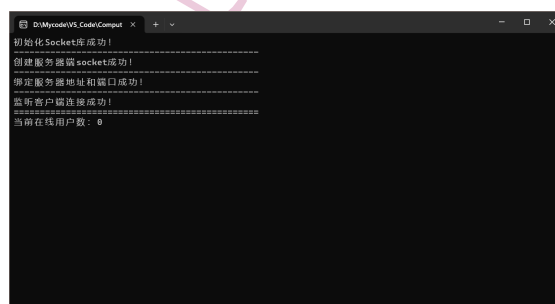
我一开始尝试直接把 sendBuff 的内容存到字符串 userInput 里，在线程处理函数中把提示词”用户 [X]: ” 和字符串 userInput 直接打印出来，发现还是不能解决问题。

查阅资料后得知，cin.getline() 函数是一种阻塞的输入方式，必须等待按下 Enter 键才能捕获输入内容。

所以，我引入头文件 #include <windows.h> 用于控制台光标操作。当客户端接收到来自服务器的广播消息时，首先获取当前光标位置，再读取提示词”用户 [X]: ” 到光标之间用户输入但还未发送的内容并存储到 userInput 中，然后用空格清除当前行，打印接收到的消息，换行后重新绘制用户输入。再进行测试，问题解决。

六、 程序运行演示

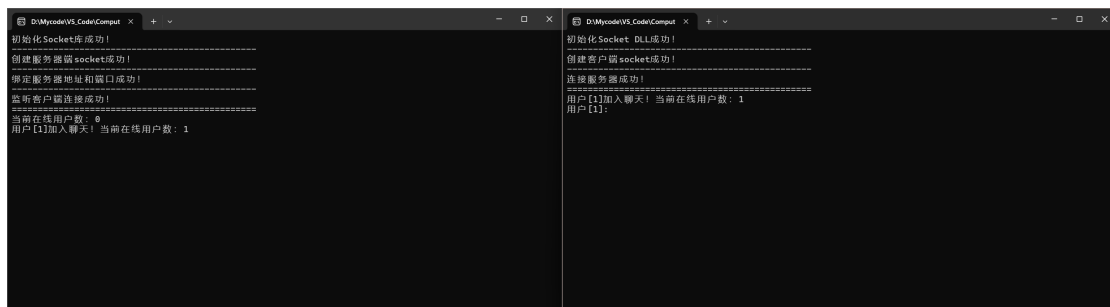
启动服务器，打印系统信息，当前在线用户数：0



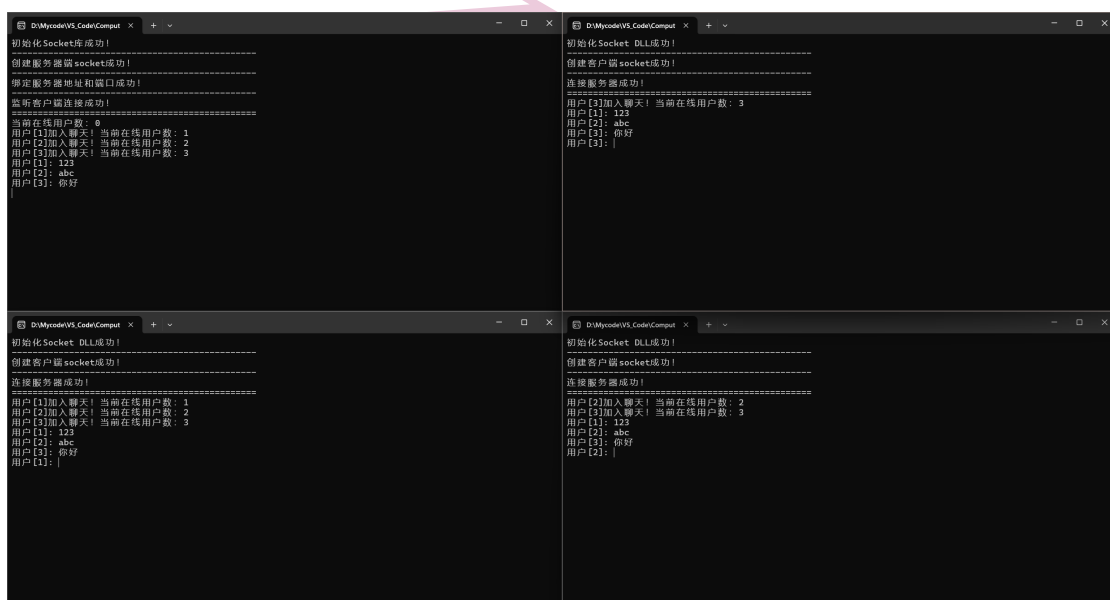
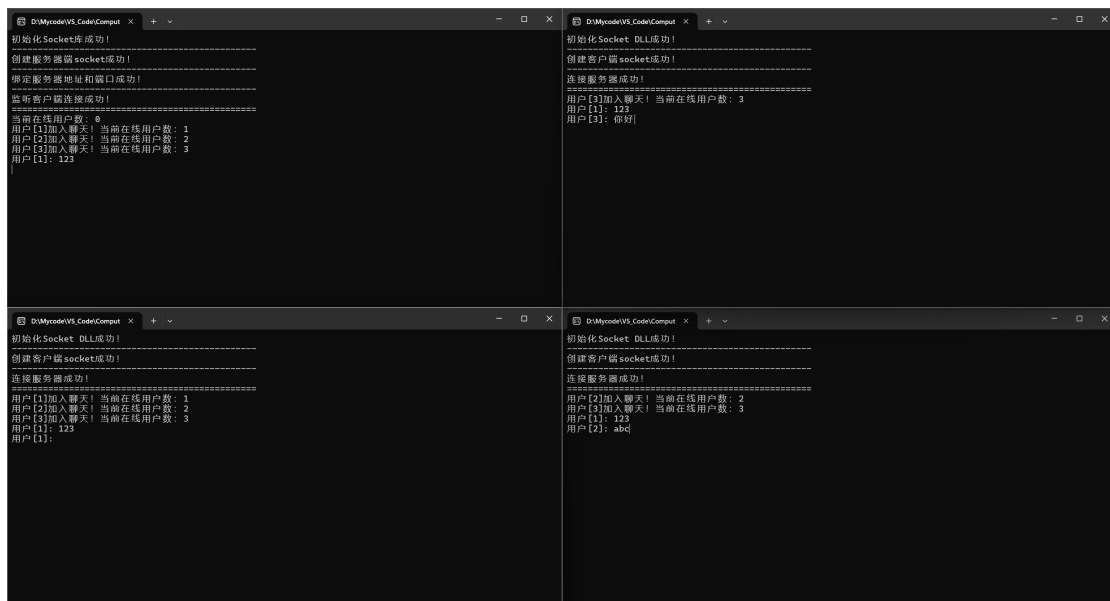
```
初始化Socket库成功!
创建服务器端socket成功!
绑定服务器地址和端口成功!
监听客户端连接成功!
=====
当前在线用户数: 0
```

如果不启动服务器，直接打开客户端，会打印”连接服务器失败!” 并直接退出。

启动服务器后，打开客户端，会打印连接成功提示，从服务器端获得用户 ID 和当前在线用户数，并等待用户键入。服务器端会输出客户加入聊天和并更新当前在线用户数。

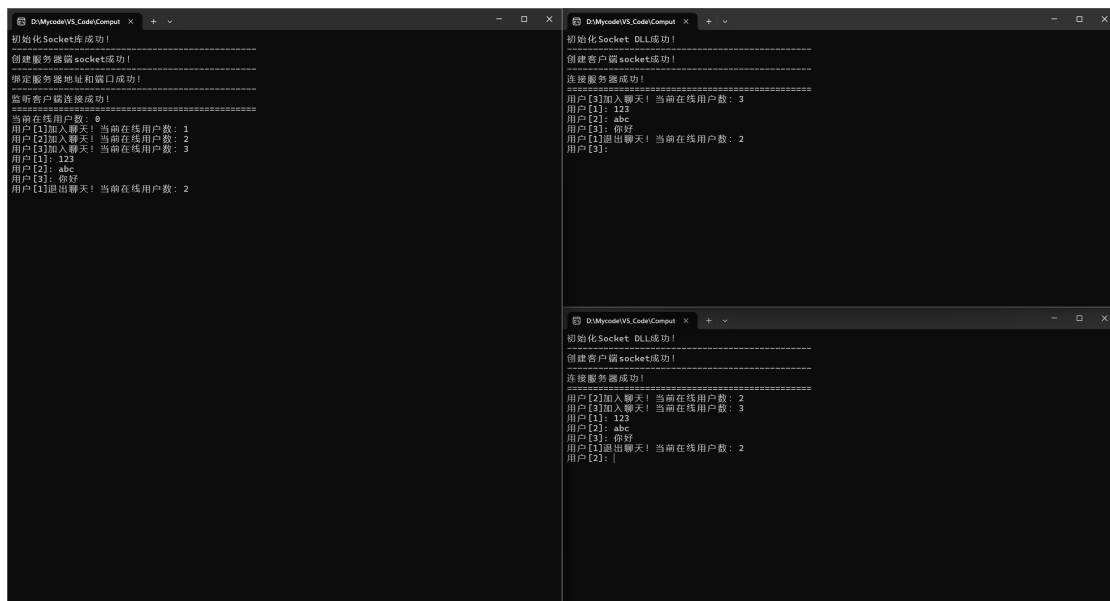


为了测试多人聊天功能，我们运行三个客户端进行测试。



我们可以看见，程序支持英文和中文信息。服务器能够正确的广播消息给其他用户，即实现了多人聊天功能。上一节提及的消息覆盖问题也得到了良好的解决。

继续测试关闭连接功能，首先测试客户端主动关闭连接。



```

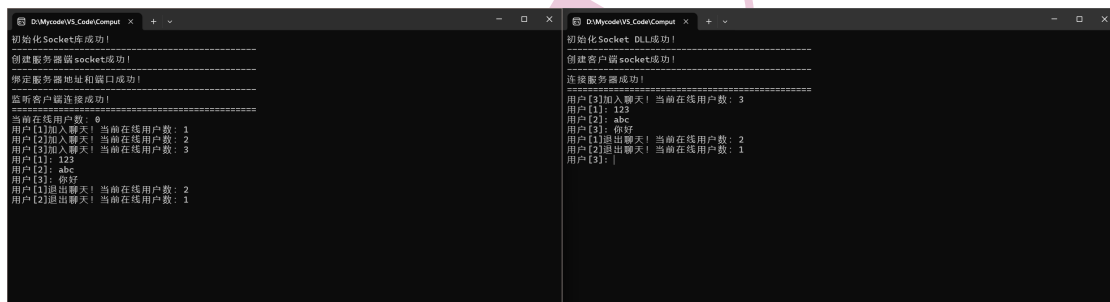
D:\Mycode\VS_Code\Comput x + - x
初始化Socket库成功!
创建服务端socket成功!
绑定服务端地址和端口成功!
监听客户端连接成功!
=====
当前在线用户数: 0
用户 [1] 加入聊天! 当前在线用户数: 1
用户 [2] 加入聊天! 当前在线用户数: 2
用户 [3] 加入聊天! 当前在线用户数: 3
用户 [1]: 123
用户 [2]: abc
用户 [3]: 你好
用户 [1] 退出聊天! 当前在线用户数: 2

D:\Mycode\VS_Code\Comput x + - x
初始化Socket DLL成功!
创建客户端socket成功!
连接服务端成功!
=====
用户 [3] 加入聊天! 当前在线用户数: 3
用户 [1]: 123
用户 [2]: abc
用户 [3]: 你好
用户 [1] 退出聊天! 当前在线用户数: 2
用户 [2]:

D:\Mycode\VS_Code\Comput x + - x
初始化Socket DLL成功!
创建客户端socket成功!
连接服务端成功!
=====
用户 [2] 加入聊天! 当前在线用户数: 2
用户 [1] 加入聊天! 当前在线用户数: 3
用户 [2]: abc
用户 [1]: 123
用户 [3]: 你好
用户 [1] 退出聊天! 当前在线用户数: 2
用户 [2]:

```

用户 1 直接关闭窗口, 服务器打印了“用户 [1] 退出聊天!”并更新了当前在线用户数。其他用户也收到了服务器的广播消息, 同样打印了“用户 [1] 退出聊天!”并更新了当前在线用户数。



```

D:\Mycode\VS_Code\Comput x + - x
初始化Socket库成功!
创建服务端socket成功!
绑定服务端地址和端口成功!
监听客户端连接成功!
=====
当前在线用户数: 0
用户 [1] 加入聊天! 当前在线用户数: 1
用户 [2] 加入聊天! 当前在线用户数: 2
用户 [3] 加入聊天! 当前在线用户数: 3
用户 [1]: 123
用户 [2]: abc
用户 [3]: 你好
用户 [1] 退出聊天! 当前在线用户数: 2
用户 [2] 退出聊天! 当前在线用户数: 1

D:\Mycode\VS_Code\Comput x + - x
初始化Socket DLL成功!
创建客户端socket成功!
连接服务端成功!
=====
用户 [3] 加入聊天! 当前在线用户数: 3
用户 [1]: 123
用户 [2]: abc
用户 [3]: 你好
用户 [1] 退出聊天! 当前在线用户数: 2
用户 [2] 退出聊天! 当前在线用户数: 1
用户 [3]:

```

用户 2 发送“quit”消息, 效果如上。

再测试服务器主动关闭连接。随着服务器的关闭, 此刻唯一连接着的用户 3 也断开连接。