

计算机网络实验报告

实验名称：SOCKET编程

学号：2012682

姓名：韩佳迅

计算机网络实验报告

实验名称：SOCKET编程

学号：2012682

姓名：韩佳迅

一、协议设计

协议

服务器

客户端

二、各模块功能与具体实现

(一) 服务器

有限客户端的socket维护问题：

全局变量

find_flag() 函数

main函数

线程函数

(二) 客户端

全局变量

main函数

接收消息的线程函数

三、程序界面与运行

双人聊天

多人聊天

四、实验过程遇到的问题及分析

一、协议设计

套接字：

进程通过套接字发送消息和接收消息

流式套接字：使用TCP协议，支持主机之间面向连接的、顺序的、可靠的、全双工字节流传输

网络协议：

计算机网络中，各个实体之间的数据交换必须遵守事先约定好的规则，这些规则就称为协议。网络协议的组成要素有：

1. 语法：数据与控制信息的结构或格式
2. 语义：需要发出何种控制信息，完成哪些动作以及做出何种响应
3. 时序：事件实现顺序的详细说明

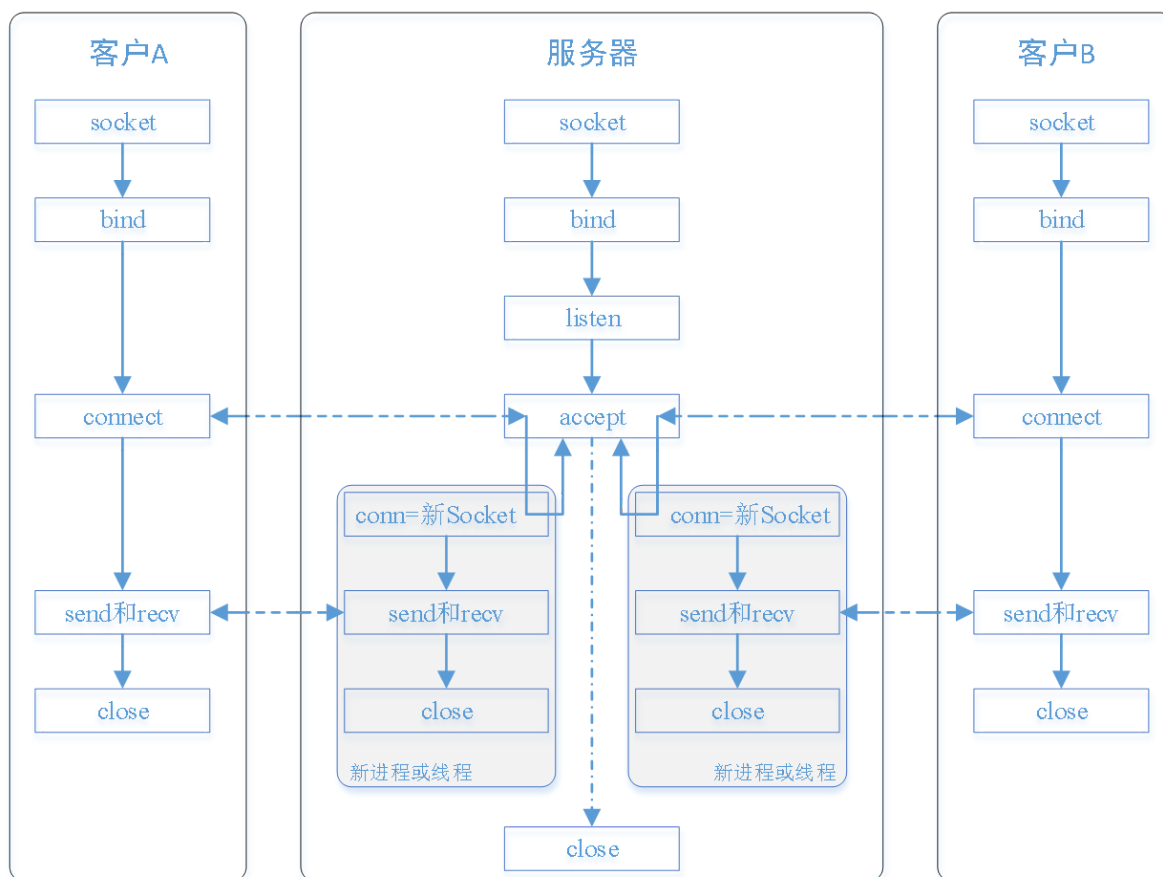
协议

在本程序中，我们进行如下的协议设计：

我们实现了一个多人聊天室（自然也支持双人）

- 采用流式套接字和TCP协议，使用多线程。
- 程序包括客户端和服务端，要求只有服务器端在线时，客户端才可以与之相连，否则连接失败。
- 服务器端支持与多个客户端相连，并设置最大连接数。当超过最大连接数时，服务器不支持连接。
 - 服务器对每个连接的客户端分配一个线程，并用一个新SOCKET与该客户端进行通信；
 - 服务器的每个线程接收它所负责客户端的消息，并将消息打上时间标签再发送给其他客户端并打印到日志信息；
 - 服务器实时检测客户端是否断开连接，若断开连接，则：
 - 及时释放与该客户端的连接资源；
 - 若此时服务器的最大连接数未满，则支持再与新的客户端建连。
- 客户端与服务端建立连接
 - 使用到两个线程：
 - 主线程负责键入要发送的消息并发送（send）给服务器端
 - 再开一个线程接收（recv）从服务器端发来的消息，并打印
 - 客户端实时检测是否与服务器建连，当recv异常，则连接断开，关闭SOCKET
 - 允许客户端自行关闭：当用户输入 **quit** 或 **直接关闭程序**，则连接断开，关闭SOCKET和聊天室
- 收发消息保存在char*类型的缓冲区 RecvBuf 和 SendBuf 里
 - 消息设置有最大长度，不允许输入超过最大长度的字符串
 - 消息以换行符为结束，允许在一条消息中输入多个空格（不以空格为结束）

我们划分两个模块，服务器模块和客户端模块，模块允许的流程如下：



服务器

1. **WSAStartup**: 初始化Socket DLL并协商使用的Socket版本
2. **socket**: 创建服务器端socket
3. **bind**: 初始化服务器IP地址和端口号，并将该地址绑定到服务器socket
4. **listen**: 使socket进入监听状态，等待客户端连接
5. **while (当前连接数 < 最大连接数)**:
 - **accept**: 接受一个特定socket请求等待队列中的连接请求，并将其返回的新的socket赋值给实现设定好的socket队列
 - **CreateThread**: 为每个连接创建线程，每个连接（线程）进入 **handlerRequest** 线程函数
6. **handlerRequest 线程函数**:
 - **while (1)**:
 - **recv**:
 - recv 返回正常:
 - 获取当前时间标签
 - 将收到的消息打上时间标签之后 **send** 给所有客户端
 - recv 返回异常:
 - 若 **WSAGetLastError()** 返回的异常值是10054，表明客户端主动关闭了连接，则直接打印输出连接关闭的信息
 - closesocket (客户端sock)**: 关闭该线程所对应的客户端的socket
7. **closesocket (服务器端sock)**: 最后，关闭服务器的socket

客户端

1. **WSAStartup**: 初始化Socket DLL并协商使用的Socket版本。
2. **socket**: 创建客户端socket。
3. **connect**: 初始化服务器IP地址和端口号, 并将服务器地址和客户端socket建连。
4. **CreateThread for recvThread()**: 创建接收消息的线程
 - **while(1)**:
 - recv**: 接收消息
 - 接收消息正常: 打印消息
 - 接收消息异常: 与服务器连接断开
5. 主线程: 键入并发送消息
 - **while(1)**:
 - cin.getline**: 从命令行键入消息
 - 若输入值是 **quit**, 直接退出循环
 - send**: 发送消息
6. **closesocket**

二、各模块功能与具体实现

(一) 服务器

有限客户端的socket维护问题:

- **clientSockets [MaxClient]**: 在程序的实现时, 我们使用一个 SOCKET 数组 **clientSockets [MaxClient]** 来存放每个 **accept** 返回的套接字。
- **flags[MaxClient]**: 同时设计一个 **flag** 数组, 标记客户端套接字数组的使用情况, 0代表未使用, 1代表正在使用。
- **conn_count**: 当前连接的客户数
- **find_flag()函数**: 返回flags数组中最小的非1下标, 表明数组中下标最小的可用socket。
- 当服务器每 **accept** 一个socket:
 - 将其赋给 **clientSockets [i]**, **i** 为 **find_flag()**函数的返回值
 - 将 **flags[i]** 置1, 表示该socket被使用
 - **conn_count++**
- 每当有一个socket断连:
 - **closesocket**, 关闭该socket
 - 将 **flags[i]** 置0, 表示数组中该socket的位置被空出来
 - **conn_count--**

注: 具体功能的实现和讲解见下面的核心代码和注释。

全局变量

```
const int PORT = 8000; //端口号
#define MaxClient 4 //最大允许客户数=MaxClient-1
#define MaxBufSize 1024 //最大缓冲区大小
int conn_count = 0; //当前连接的客户数
SOCKET clientSockets[MaxClient]; //客户端socket数组
SOCKADDR_IN clientAddrs[MaxClient]; //客户端地址数组
int flags[MaxClient]; //flag数组
```

find_flag() 函数

```
int find_flag(){
    for (int i = 0; i < MaxClient; i++){
        if (flags[i] == 0)
            return i;
    }
    return -1;
}
```

main函数

```
int main()
{
    //=====初始化socket dll=====
    WSADATA wsadata;
    WSStartup(MAKEWORD(2, 2), &wsadata); //MAKEWORD(主版本号, 副版本号)
    if (LOBYTE(wsadata.wVersion) != 2 || HIBYTE(wsadata.wVersion) != 2)
    {
        cout << "初始化Socket DLL: fail! \n" << endl;
        return -1;
    }
    cout << "初始化Socket DLL: success!\n" << endl;

    //=====创建socket=====
    SOCKET serverSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP); //采用TCP协议, 流式套接字
    if (serverSocket == INVALID_SOCKET)
    {
        cout << "创建socket: fail! \n" << endl;
        return -1;
    }
    cout << "创建socket: success! \n" << endl;

    //=====初始化服务器地址 && bind=====
    SOCKADDR_IN serverAddr;
    serverAddr.sin_family = AF_INET; //地址类型
    serverAddr.sin_addr.S_un.S_addr = inet_addr("127.0.0.1"); //地址
    serverAddr.sin_port = htons(PORT); //端口号
    int tem = bind(serverSocket, (LPSOCKADDR)&serverAddr, sizeof(serverAddr));
```

```

if (tem == SOCKET_ERROR)
{
    cout << "bind: fail! \n" << endl;
    return -1;
}
else
{
    cout << "bind to port " << PORT << ": success! \n" << endl;
}

//=====listen=====
tem = listen(serverSocket, MaxClient); // 最大监听队列长度为 MaxClient
if (tem != 0)
{
    cout << "listen: fail! \n" << endl;
    return -1;
}
else
{
    cout << "listen: success! \n" << endl;
}

cout << "Server ready! Waiting for client request..." << endl << endl;

//=====循环接收client=====
while (1)
{
    if (conn_count < MaxClient)
    {
        int len = sizeof(SOCKADDR);
        int kk = find_flag();
        clientSockets[kk] = accept(serverSocket,
(sockaddr*)&clientAddr[kk], &len);
        flags[kk] = 1;
        conn_count++;
        if (clientSockets[kk] == SOCKET_ERROR)
        {
            cout << "wrong client! \n" << endl;
            closesocket(serverSocket);
            WSACleanup();
            return -1;
        }
        time_t ts = time(NULL); //时间标签
        char mytime[32]{ 0 };
        strncpy(mytime, ctime(&ts), sizeof(mytime));
        cout << "client (" << clientSockets[kk] << ") connect, number of
current clients: " << conn_count << ". TIME: " << mytime;
        HANDLE hThread = CreateThread(NULL, NULL, handlerRequest,
(LPVOID)kk, 0, NULL);
        CloseHandle(hThread);
    }
    else
    {
        cout << "Request is full! " << endl;
        return -1;
    }
}

```

```

    }
}
closesocket(serverSocket); //关闭socket
WSACleanup();
return 0;
}

```

线程函数

```

DWORD WINAPI handlerRequest(LPVOID lpParam)
{
    int n = (int)lpParam;
    int receByt = 0;
    char RecvBuf[MaxBufSize]; //接收消息的缓冲区
    char SendBuf[MaxBufSize]; //发送消息的缓冲区

    while (1)
    {
        //接收来自该线程所对应socket的消息
        receByt = recv(clientSockets[n], RecvBuf, sizeof(RecvBuf), 0);

        if (receByt > 0)
        {
            time_t ts = time(NULL); //时间标签
            char mytime[32]{ 0 };
            strncpy(mytime, ctime(&ts), sizeof(mytime));
            cout << "client" << clientSockets[n] << ": " << RecvBuf << "    --"
<< mytime;

            sprintf(SendBuf, "[%d]: %s\n    --s", clientSockets[n],
RecvBuf, mytime);

            //===== 向其他socket都发送消息 =====
            for (int i = 0; i < MaxClient; i++)
            {
                if (flags[i] == 1)
                {
                    send(clientSockets[i], SendBuf, sizeof(SendBuf), 0);
                }
            }
        }
        else
        {
            //===== 关闭连接, 释放资源 =====
            if (WSAGetLastError() == 10054) //客户端主动关闭连接
            {
                time_t ts = time(NULL);
                char mytime[32]{ 0 };
                strncpy(mytime, ctime(&ts), sizeof(mytime));
                cout << "client " << clientSockets[n] << " exit, number of
current clients: " << conn_count - 1 << ". TIME: "<<mytime;
                closesocket(clientSockets[n]);
            }
        }
    }
}

```

```

        flags[n] = 0;
        conn_count--;
        return 0;
    }
    cout << "failed to receive,Error:" << WSAGetLastError() << endl;
    return 0;
}
}
}

```

(二) 客户端

客户端我们在主线程之外又开了一个线程，使得一个接收消息，一个发送消息。

全局变量

```

const int PORT = 8000; //端口号
const int BufSize = 1024; //缓冲区大小
SOCKET clientSocket; //客户端socket

```

main函数

```

int main()
{
    //=====初始化socket dll=====
    WSADATA wsadata;
    WSASStartup(MAKEWORD(2, 2), &wsadata);
    if (LOBYTE(wsadata.wVersion) != 2 || HIBYTE(wsadata.wVersion) != 2)
    {
        cout << "初始化Socket DLL: fail! \n" << endl;
        return -1;
    }
    cout << "初始化Socket DLL: success!\n" << endl;

    //=====创建客户端的socket=====
    clientSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP); //面向网路的流式套接字，
    第三个参数代表自动选择协议
    if (clientSocket == INVALID_SOCKET)
    {
        cout << "创建socket: fail! \n" << endl;
        return -1;
    }
    cout << "创建socket: success! \n" << endl;

    //=====服务器地址=====
    SOCKADDR_IN servAddr;
    servAddr.sin_family = AF_INET; //地址类型
    servAddr.sin_addr.S_un.S_addr = inet_addr("127.0.0.1"); //地址
    servAddr.sin_port = htons(PORT); //端口号

    //===== connect =====

```



```

    int r = connect(clientSocket, (SOCKADDR*)&servAddr, sizeof(SOCKADDR)) ==
SOCKET_ERROR;
    if (r===-1)
    {
        cout << "connect fail: " << WSAGetLastError() << endl;
    }
    else
    {
        cout << "connect: success! " << endl;
    }
    cout << "Now you can start chatting!" << endl;

    //===== 创建接受消息线程 =====
    CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)recvThread, NULL, 0, 0);

    //===== 主线程用于键入消息并发送消息 =====
    while (1)
    {
        char buf[BufSize] = { 0 };
        cin.getline(buf, sizeof(buf));
        if (strcmp(buf, "quit") == 0)//若输入“quit”，则退出
            break;
        send(clientSocket, buf, sizeof(buf), 0);
    }
    closesocket(clientSocket);
    WSACleanup();
    return 0;
}

```

接收消息的线程函数

```

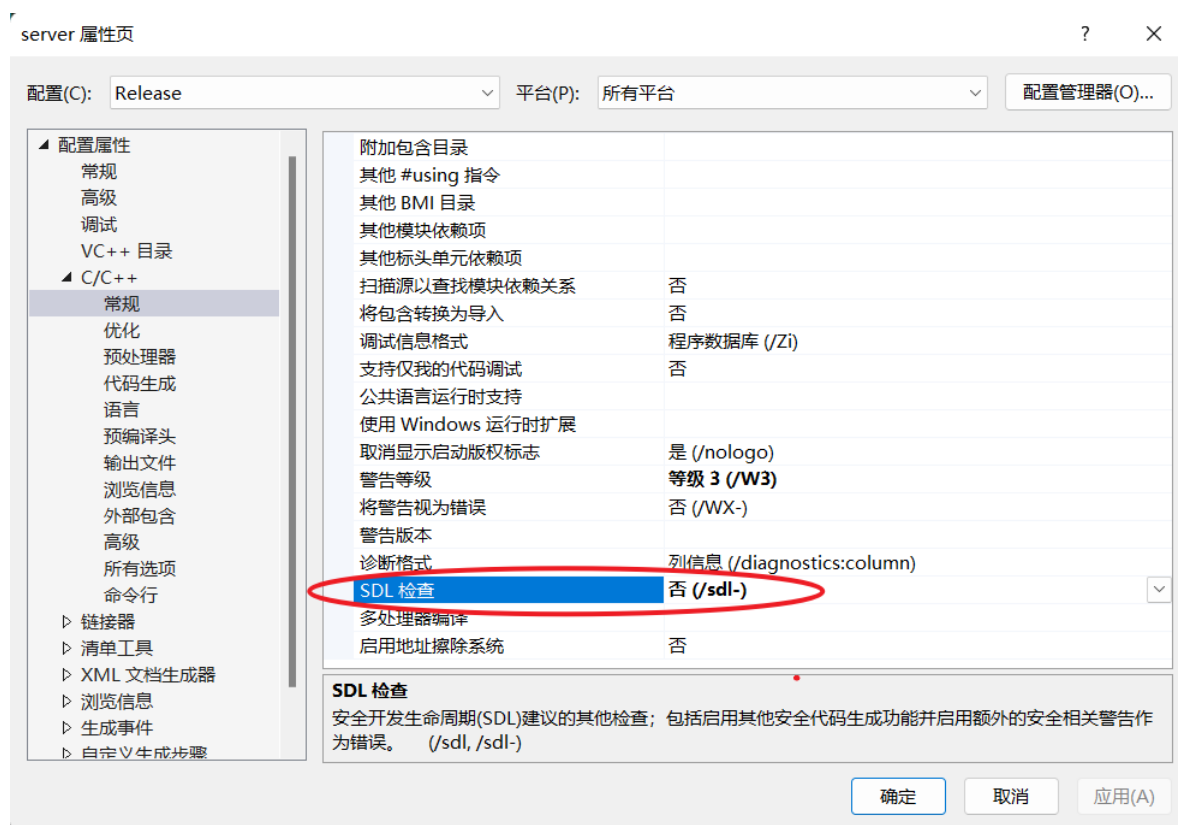
DWORD WINAPI recvThread()
{
    while (1)
    {
        char buffer[BufSize] = { 0 };//接收消息的缓冲区
        int r = recv(clientSocket, buffer, sizeof(buffer), 0);//nrecv是接收到的字节
数

        if (r > 0)//如果接收到的字符数大于0
        {
            cout << buffer << endl;
        }
        else if (r < 0)//如果接收到的字符数小于0就说明断开连接
        {
            cout << "loss connect!" << endl;
            break;
        }
    }
    return 0;
}

```

三、程序界面与运行

在visual studio编译时，需要将SDL检查置为否



双人聊天

- 首先，打开服务器程序server：
 - 在服务器端依次打印出操作日志，例如初始化Socket DLL、创建socket、bind等。

```
D:\learning\my_大三上\计算机网络\实验\Lab1\server\Release\server.exe
初始化Socket DLL: success!
创建socket: success!
bind to port 8000: success!
listen: success!
Server ready! Waiting for client request...
```

- 其次，打开两个客户端程序client：
 - 服务器端：分别打印出两个client的连接信息、当前连接数、时间。
 - 客户端：依次打印出操作日志，并可以开始输入聊天信息了。

The image shows three terminal windows. The top window is the server, titled 'D:\learning\my_大三上\计算机网络\实验\Lab1\server\Release\server.exe'. It displays the following output: '初始化Socket DLL: success!', '创建socket: success!', 'bind to port 8000: success!', 'listen: success!', 'Server ready! Waiting for client request...'. It then shows two clients connecting: 'client (304) connect, number of current clients: 1. TIME: Wed Oct 19 21:19:34 2022' and 'client (324) connect, number of current clients: 2. TIME: Wed Oct 19 21:19:40 2022'. The bottom-left window is a client titled 'D:\learning\my_大三上\计算机网络\实验\Lab1\client\Release\cli...', showing '初始化Socket DLL: success!', '创建socket: success!', 'connect: success!', and 'Now you can start chatting!'. The bottom-right window is another client with the same output.

- 双方进行聊天
 - 服务器端：日志记录两人的聊天内容和时间
 - 客户端：
 - 直接敲入消息，按下回车即可发送
 - 接收并显示所有客户端的消息以及消息时间

The image shows three terminal windows. The top window is the server, titled 'D:\learning\my_大三上\计算机网络\实验\Lab1\server\Release\server.exe'. It shows the same initialization output as before, followed by chat messages: 'client304: hello --Wed Oct 19 21:24:05 2022', 'client324: how are you? --Wed Oct 19 21:24:11 2022', 'client304: I'm fine,thank you! --Wed Oct 19 21:24:23 2022', and 'client324: haha --Wed Oct 19 21:24:28 2022'. The bottom-left window is a client titled 'D:\learning\my_大三上\计算机网络\实验\Lab1\client\Release\cli...', showing 'connect: success!', 'Now you can start chatting!', and then the received messages: 'hello', '[#304]: hello --Wed Oct 19 21:24:05 2022', '[#324]: how are you? --Wed Oct 19 21:24:11 2022', 'I'm fine,thank you!', '[#304]: I'm fine,thank you! --Wed Oct 19 21:24:23 2022', and '[#324]: haha --Wed Oct 19 21:24:28 2022'. The bottom-right window is another client with the same output.

- 退出聊天——输入 *quit* 或 直接关闭窗口

- 在服务器端打印客户端退出日志

```
D:\learning\my_大三上\计算机网络\实验\Lab1\server\Release\server.exe
初始化Socket DLL: success!
创建socket: success!
bind to port 8000: success!
listen: success!
Server ready! Waiting for client request...
client (304) connect, number of current clients: 1. TIME: Wed Oct 19 21:19:34 2022
client (324) connect, number of current clients: 2. TIME: Wed Oct 19 21:19:40 2022
client304: hello --Wed Oct 19 21:24:05 2022
client324: how are you? --Wed Oct 19 21:24:11 2022
client304: I'm fine,thank you! --Wed Oct 19 21:24:23 2022
client324: haha --Wed Oct 19 21:24:28 2022
client304: quitfad --Wed Oct 19 21:27:56 2022
client 304 exit, number of current clients: 1. TIME: Wed Oct 19 21:28:02 2022
```

- 关闭服务器
 - 客户端显示连接失败

```
选择 D:\learning\my_大三上\计算机网络\实验\Lab1\client\Releas...
[304]: I'm fine,thank you!
--Wed Oct 19 21:24:23 2022
haha
[324]: haha
--Wed Oct 19 21:24:28 2022
[304]: quitfad
--Wed Oct 19 21:27:56 2022
loss connect!
```

多人聊天

- 我们设定 最大连接数 = `MaxClient - 1` = $4 - 1 = 3$
- **最大连接数测试:**

首先, 尝试开大于3个client: 服务器端不支持, 并退出。

- **群聊功能测试:**

我们开3个client (后续可以自行调整 `MaxClient` 更改最大连接, 使得能够开辟更多client)

每个客户都可以发送消息, 并且收到所有用户的消息

The image shows three terminal windows. The top window is the server's console, displaying the following output:

```
D:\learning\my_大三上\计算机网络\实验\Lab1\server\Release\server.exe
初始化Socket DLL: success!
创建socket: success!
bind to port 8000: success!
listen: success!
Server ready! Waiting for client request...

client (220) connect, number of current clients: 1. TIME: Wed Oct 19 21:38:51 2022
client (308) connect, number of current clients: 2. TIME: Wed Oct 19 21:38:55 2022
client (324) connect, number of current clients: 3. TIME: Wed Oct 19 21:39:00 2022
client220: 我是client1 --Wed Oct 19 21:39:32 2022
client308: 我是client2 --Wed Oct 19 21:39:38 2022
client324: 我是client3 --Wed Oct 19 21:39:45 2022
```

The bottom two windows are client consoles. The left one is for client1, showing:

```
D:\learning\my_大三上\计算机网络\实验\Lab1\client...
初始化Socket DLL: success!
创建socket: success!
connect: success!
Now you can start chatting!

我是client1
[#220]: 我是client1 --Wed Oct 19 21:39:32 2022

[#308]: 我是client2 --Wed Oct 19 21:39:38 2022

[#324]: 我是client3 --Wed Oct 19 21:39:45 2022
```

The right one is for client2, showing:

```
D:\learning\my_大三上\计算机网络\实验\Lab1\client...
初始化Socket DLL: success!
创建socket: success!
connect: success!
Now you can start chatting!

[#220]: 我是client1 --Wed Oct 19 21:39:32 2022

我是client2
[#308]: 我是client2 --Wed Oct 19 21:39:38 2022

[#324]: 我是client3 --Wed Oct 19 21:39:45 2022
```

• 用户退出测试:

让第二个client退出聊天:

1. 服务器端: 打印“client 308 exit”的消息, 当前连接数降低为2
2. 客户端: 其余客户的聊天不受影响

The image shows the same three terminal windows after client 308 has exited. The server console now shows:

```
D:\learning\my_大三上\计算机网络\实验\Lab1\server\Release\server.exe
bind to port 8000: success!
listen: success!
Server ready! Waiting for client request...

client (220) connect, number of current clients: 1. TIME: Wed Oct 19 21:38:51 2022
client (308) connect, number of current clients: 2. TIME: Wed Oct 19 21:38:55 2022
client (324) connect, number of current clients: 3. TIME: Wed Oct 19 21:39:00 2022
client220: 我是client1 --Wed Oct 19 21:39:32 2022
client308: 我是client2 --Wed Oct 19 21:39:38 2022
client324: 我是client3 --Wed Oct 19 21:39:45 2022
client308: 我要退出了, byebye --Wed Oct 19 21:43:23 2022
client 308 exit, number of current clients: 2. TIME: Wed Oct 19 21:43:29 2022
client220: client3, 我们继续聊! --Wed Oct 19 21:44:10 2022
client324: 好呀好呀! --Wed Oct 19 21:44:15 2022
```

The client1 console shows:

```
D:\learning\my_大三上\计算机网络\实验\Lab1\client...
Now you can start chatting!

我是client1
[#220]: 我是client1 --Wed Oct 19 21:39:32 2022

[#308]: 我是client2 --Wed Oct 19 21:39:38 2022

[#324]: 我是client3 --Wed Oct 19 21:39:45 2022

[#308]: 我要退出了, byebye~ --Wed Oct 19 21:43:23 2022

client3, 我们继续聊!
[#220]: client3, 我们继续聊! --Wed Oct 19 21:44:10 2022

[#324]: 好呀好呀! --Wed Oct 19 21:44:15 2022
```

The client2 console shows:

```
D:\learning\my_大三上\计算机网络\实验\Lab1\client...
Now you can start chatting!

[#220]: 我是client1 --Wed Oct 19 21:39:32 2022

[#308]: 我是client2 --Wed Oct 19 21:39:38 2022

我是client3
[#324]: 我是client3 --Wed Oct 19 21:39:45 2022

[#308]: 我要退出了, byebye~ --Wed Oct 19 21:43:23 2022

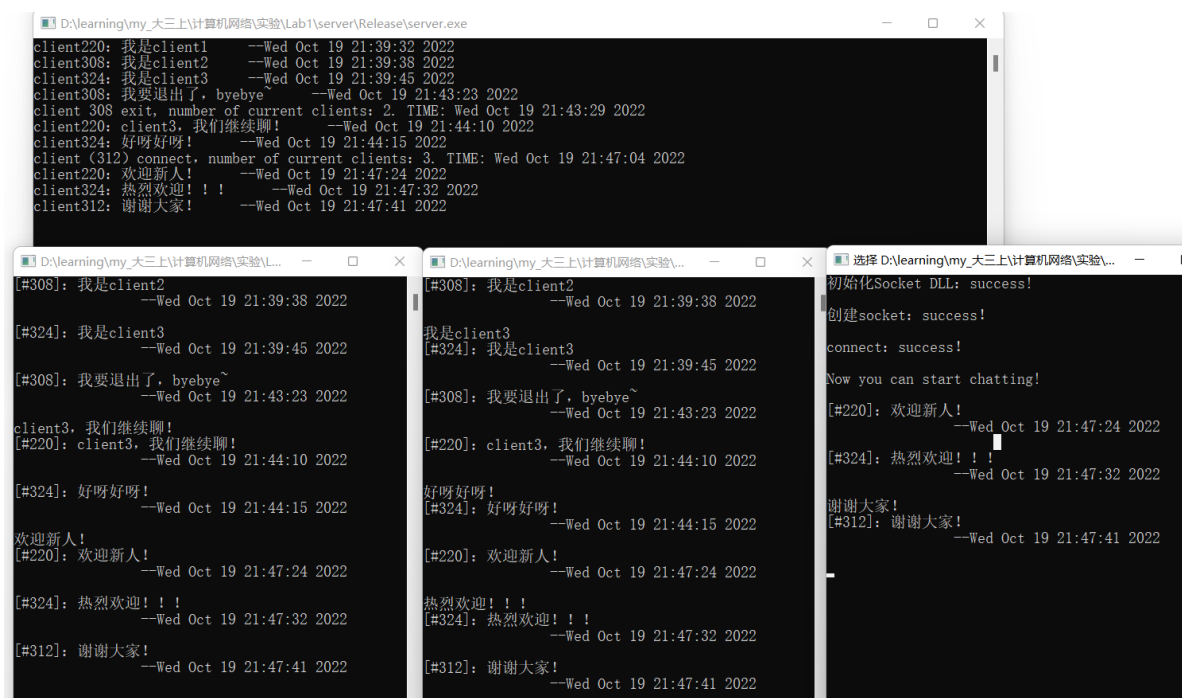
[#220]: client3, 我们继续聊! --Wed Oct 19 21:44:10 2022

好呀好呀!
[#324]: 好呀好呀! --Wed Oct 19 21:44:15 2022
```

• 新用户加入测试：

再加入一个新用户，使之达到最大连接数：

1. 服务器端：打印新用户connect日志，当前连接数+1
2. 客户端：新用户加入聊天，聊天正常



```
D:\learning\my_大三上计算机网络\实验\Lab1\server\Release\server.exe
client220: 我是client1 --Wed Oct 19 21:39:32 2022
client308: 我是client2 --Wed Oct 19 21:39:38 2022
client324: 我是client3 --Wed Oct 19 21:39:45 2022
client308: 我要退出了, byebye --Wed Oct 19 21:43:23 2022
client 308 exit, number of current clients: 2. TIME: Wed Oct 19 21:43:29 2022
client220: client3, 我们继续聊! --Wed Oct 19 21:44:10 2022
client324: 好呀好呀! --Wed Oct 19 21:44:15 2022
client (312) connect, number of current clients: 3. TIME: Wed Oct 19 21:47:04 2022
client220: 欢迎新人! --Wed Oct 19 21:47:24 2022
client324: 热烈欢迎!!! --Wed Oct 19 21:47:32 2022
client312: 谢谢大家! --Wed Oct 19 21:47:41 2022

D:\learning\my_大三上计算机网络\实验\Lab1\client\Release\client.exe
[#308]: 我是client2 --Wed Oct 19 21:39:38 2022
[#324]: 我是client3 --Wed Oct 19 21:39:45 2022
[#308]: 我要退出了, byebye --Wed Oct 19 21:43:23 2022
client3, 我们继续聊! --Wed Oct 19 21:44:10 2022
[#220]: client3, 我们继续聊! --Wed Oct 19 21:44:10 2022
[#324]: 好呀好呀! --Wed Oct 19 21:44:15 2022
欢迎新人! --Wed Oct 19 21:47:24 2022
[#220]: 欢迎新人! --Wed Oct 19 21:47:24 2022
[#324]: 热烈欢迎!!! --Wed Oct 19 21:47:32 2022
[#312]: 谢谢大家! --Wed Oct 19 21:47:41 2022

选择 D:\learning\my_大三上计算机网络\实验\Lab1\client\Release\client.exe
初始化Socket DLL: success!
创建socket: success!
connect: success!
Now you can start chatting!
[#220]: 欢迎新人! --Wed Oct 19 21:47:24 2022
[#324]: 热烈欢迎!!! --Wed Oct 19 21:47:32 2022
谢谢大家! --Wed Oct 19 21:47:41 2022
[#312]: 谢谢大家! --Wed Oct 19 21:47:41 2022
```

四、实验过程遇到的问题及分析

有限客户端的socket维护问题：

在本程序中，使用了一个socket数组clientSockets、和一个维护数组 flags 来承接从accept返回的新socket。然而，实际上，在一开始的实现版本中，我并没有开辟flag数组，而是使用的是clientSockets数组移位的方式，并将新socket永远赋值给 clientSockets[conn_count]，如下代码：

```
for (int j = n; j < conn_count-1; j++)
{
    clientsockets[j] = clientsockets[j + 1];
    clientaddrs[j] = clientaddrs[j + 1];
}
```

然而，在实际运行时发现，这样的移位会导致如下问题：

当正在连接的一个客户端下线之后，若再有新客户端上线，则原有客户端之间不能正常交流了，而且出现了两个客户端共用一个socket的现象。这应该是因为服务器端的原有socket已经和客户端程序建立，而移动会导致问题。

因此，我增加了一个维护数组，记录 clientSockets 中可用元素的下标，这样在新加用户时，不会动到已经连接好的socket，使得聊天维护正常。

