

南开大学

《计算机网络》课程实验报告

实验一：利用 Socket 编写一个聊天程序



学 院_____网络空间安全学院_____
专 业_____信息安全_____
学 号_____2112060_____
姓 名_____孙璐_____

一、 实验要求

1. 给出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. 使用流式套接字、采用多线程（或多进程）方式完成程序。
4. 程序应该有基本的对话界面，但可以不是图形界面。程序应该有正常的退出方式。
5. 完成的程序应该支持多人聊天，支持英文和中文聊天。
6. 编写的程序应该结构清晰，具有较好的可读性。
7. 在实验中观察是否有数据丢失，提交程序源码和实验报告。

二、 实验过程

1. 聊天协议的完整说明

（1） 消息类型

聊天协议支持文本消息。每条消息包含一个文本字符串，用于表示聊天内容。

（2） 语法

语法是用户数据与控制信息的结构与格式，以及数据出现的顺序。报文的格式：发送时间戳，发送人姓名，发送的消息内容（[时间戳]用户名： 消息内容），以及其他的提示作用的字符串。退出请求是 `exit_request`。时间戳采用“YYYY-MM-DD HH:MM:SS”格式，以便清晰地显示消息的时间。服务器会根据用户名识别不同的用户，以便将消息发送给正确的目标用户。消息内容是用户输入的文本内容，是实际的聊天消息文本。

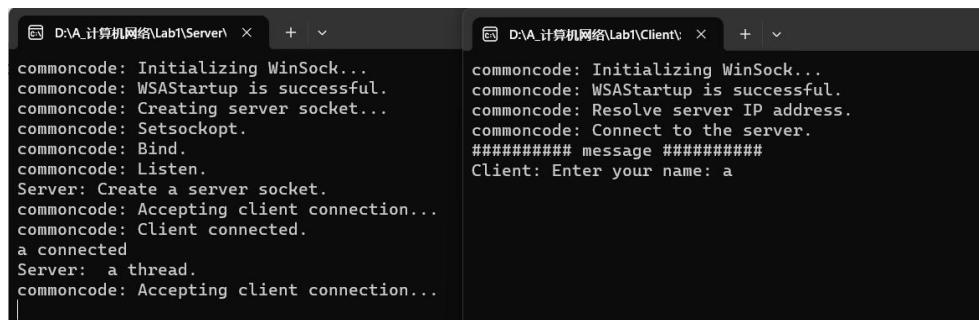
```
struct ChatMessage
{
    char Client_Name[20];
    char time_Stamp[20];
    char MeSsage[256];
    ChatMessage()
    {
        memset(Client_Name, 0, sizeof(Client_Name));
        memset(time_Stamp, 0, sizeof(time_Stamp));
        memset(MeSsage, 0, sizeof(MeSsage));
    }
};
```

（3） 语义

语义是解释控制信息每个部分的含义，它规定了需要发出何种控制信息，以及完成的动作与做出什么样的响应。

(a) 服务器端：

初始化 WinSock 库以准备进行网络通信，输出“commoncode: Initializing WinSock...”，初始完成后输出“commoncode: WSASStartup is successful”。在服务器端，服务器建立 socket, bind 绑定到指定端口之后进入 listen 监听状态，成功完成都会输出相应提示信息（“commoncode: Creating server socket...”，“commoncode: Setsockopt”，“commoncode: Bind”，“commoncode: Listen”，“Server: Create a server socket.”）。server 等待 client 的 connect 连接请求，输出“commoncode: Accepting client connection...”等待期间服务器保持阻塞状态，一旦受到 connect 请求，就会执行 accept，输出“Client connected”的消息，为该客户端分配唯一的标识，方便消息的广播。尝试接收用户名，如果成功接收控制台输出“xxx connected”，为其创建一个线程，创建成功会输出“Server: a thread.”服务器继续等待下一个客户端的连接请求，以支持多个客户端的连接。



```
D:\A_计算机网络\Lab1\Server >
commoncode: Initializing WinSock...
commoncode: WSASStartup is successful.
commoncode: Creating server socket...
commoncode: Setsockopt.
commoncode: Bind.
commoncode: Listen.
Server: Create a server socket.
commoncode: Accepting client connection...
commoncode: Client connected.
a connected
Server: a thread.
commoncode: Accepting client connection...

D:\A_计算机网络\Lab1\Client>
commoncode: Initializing WinSock...
commoncode: WSASStartup is successful.
commoncode: Resolve server IP address.
commoncode: Connect to the server.
##### message #####
Client: Enter your name: a
```

所有用户连接完成后，开始进行收发消息的操作。为了同时支持多个 client 在线，使用 CreateThread 随机创建一个没有被使用过的线程号，为每一个客户端创建了一个线程，用于该客户端的消息接收与广播。在线程中接收 client 发送的报文信息，如果收到了某个 client 发送的消息，字符串拼接在服务器端显示已经收到该信息并将该信息的具体细节显示出来，将其作为发送给其他客户端的报文。（服务器端显示格式：Server:[时间戳] Received message from client: 该客户端名字 message: 该客户端发送的消息）

```
D:\A_计算机网络\Lab1\Server x + -
commoncode: Initializing WinSock...
commoncode: WSStartup is successful.
commoncode: Creating server socket...
commoncode: Setsockopt.
commoncode: Bind.
commoncode: Listen.
Server: Create a server socket.
commoncode: Accepting client connection...
commoncode: Client connected.
vf connected
Server: a thread.
commoncode: Accepting client connection...
commoncode: Client connected.
ve connected
Server: a thread.
commoncode: Accepting client connection...
Server: [ 2023-10-20 16:19:50 ] Received message from client: vf message: we
gebewrb

D:\A_计算机网络\Lab1\Client x + -
commoncode: Initializing WinSock...
commoncode: WSStartup is successful.
commoncode: Resolve server IP address.
commoncode: Connect to the server.
##### message #####
Client: Enter your name: vf
2023-10-20 16:19:50 : Client: Enter your message (type 'exit' if you want to
exit): wegebewrb

D:\A_计算机网络\Lab1\Client2 x + -
commoncode: Initializing WinSock...
commoncode: WSStartup is successful.
commoncode: Resolve server IP address.
commoncode: Connect to the server.
##### message #####
Client: Enter your name: ve
Client: Server: [ 2023-10-20 16:19:50 ] Received message from client: vf mes
sage: wegebewrb [Broadcast]
```

为了让其他的客户端也显示出该客户端的消息，服务器端进行发送操作。此时服务器端作为中转端，识别出具体发送消息的客户端，将之前从该客户端收到的消息广播到其他的客户端，在其他客户端显示消息（显示格式 Client: Server:[时间戳] Received message from client: 该客户端名字 message: 该客户端发送的信息【Broadcast】）。

当客户端想要退出，会向服务器端发送“exit_request”，服务器确认后关闭连接，输出“commoncode: Closing socket...”。

```
D:\A_计算机网络\Lab1\Server x + -
commoncode: Client connected.
e connected
Server: a thread.
commoncode: Accepting client connection...
commoncode: Client connected.
f connected
Server: a thread.
commoncode: Accepting client connection...
Server: [ 2023-10-20 15:21:39 ] Received message from client: d message: ef
Server: [ 2023-10-20 15:21:46 ] Received message from client: f message: er
Server: [ 2023-10-20 15:21:42 ] Received message from client: e message: er
Server: [ 2023-10-20 15:21:56 ] Received message from client: e message: exit_request
Server: Failed to receive message from client e. Error: 10054
commoncode: Closing socket...
Server: [ 2023-10-20 15:21:49 ] Received message from client: d message: ed
Server: [ 2023-10-20 15:21:53 ] Received message from client: f message: ed

Microsoft Visual Studio 调试 x + -
Client: Server: [ 2023-10-20 15:21:46 ] Received message from client: f message: er [Broadcast]
2023-10-20 15:21:42 : Client: Enter your message (type 'exit' if you want to exit): er
2023-10-20 15:21:56 : Client: Enter your message (type 'exit' if you want to exit): exit
Exit request sent.
Client: Server confirmed exit. Closing the connection...
D:\A_计算机网络\Lab1\Client2\Debug\Client2.exe (进程 43904)已退出，代码为 0。
要在调试停止时自动关闭控制台，请启用“工具”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口。 . . .

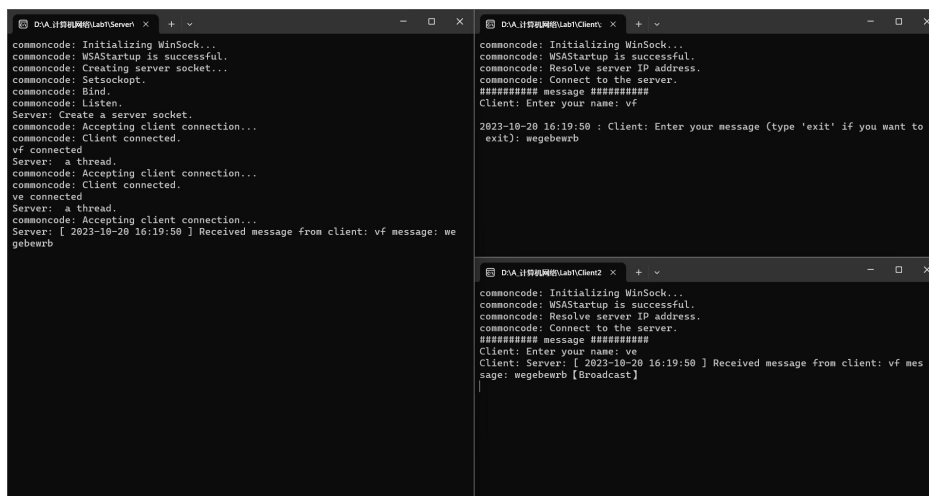
D:\A_计算机网络\Lab1\Client x + -
commoncode: Connect to the server.
##### message #####
Client: Enter your name: d
2023-10-20 15:21:39 : Client: Enter your message (type 'exit' if you want to exit): ef
Client: Server: [ 2023-10-20 15:21:46 ] Received message from client: f message: er [Broadcast]
Client: Server: [ 2023-10-20 15:21:42 ] Received message from client: e message: er [Broadcast]
2023-10-20 15:21:49 : Client: Enter your message (type 'exit' if you want to exit): ed
Client: Server: [ 2023-10-20 15:21:53 ] Received message from client: f message: ed [Broadcast]

D:\A_计算机网络\Lab1\Client2 x + -
commoncode: Connect to the server.
##### message #####
Client: Enter your name: f
Client: Server: [ 2023-10-20 15:21:39 ] Received message from client: d message: ef [Broadcast]
2023-10-20 15:21:46 : Client: Enter your message (type 'exit' if you want to exit): er
Client: Server: [ 2023-10-20 15:21:42 ] Received message from client: e message: er [Broadcast]
Client: Server: [ 2023-10-20 15:21:49 ] Received message from client: d message: ed [Broadcast]
2023-10-20 15:21:53 : Client: Enter your message (type 'exit' if you want to exit): ed
```

(b) 客户端

初始化 WinSock 库以准备进行网络通信，输出“commoncode: Initializing WinSock...”和“WSAStartup is successful”。建立 socket，与服务器端进行连接，解析服务器 IP 地址，解析成功输出“commoncode: Resolve server IP address.”。若成功能连接到客户端，则输出“commoncode: Connect to the

server” 的消息。输入客户端名字，尝试将名字发送到服务器端，如果接收成功的话，会在服务器端输出“xxx connected.”



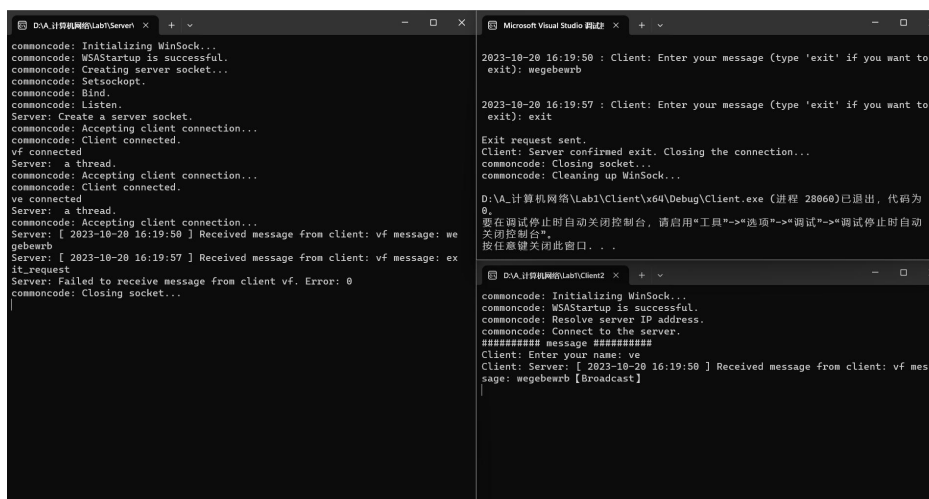
```
commoncode: Initializing WinSock...
commoncode: WSAStartup is successful.
commoncode: Creating server socket...
commoncode: Setsockopt.
commoncode: Bind.
commoncode: Listen.
Server: Create a server socket.
commoncode: Accepting client connection...
commoncode: Client connected.
vf connected
Server: a thread.
commoncode: Accepting client connection...
commoncode: Client connected.
ve connected
Server: a thread.
commoncode: Accepting client connection...
Server: [ 2023-10-20 16:19:50 ] Received message from client: vf message: wegebewrb

commoncode: Initializing WinSock...
commoncode: WSAStartup is successful.
commoncode: Resolve server IP address.
commoncode: Connect to the server.
##### message #####
Client: Enter your name: vf
2023-10-20 16:19:50 : Client: Enter your message (type 'exit' if you want to exit): wegebewrb

commoncode: Initializing WinSock...
commoncode: WSAStartup is successful.
commoncode: Resolve server IP address.
commoncode: Connect to the server.
##### message #####
Client: Enter your name: ve
Client: Server: [ 2023-10-20 16:19:50 ] Received message from client: vf message: wegebewrb [Broadcast]
```

创建接收消息和发送消息的两个线程，recv 接收消息存在 message_buffer 里，接收到了消息就会输出“Client: 接收到的消息 【Broadcast】”

获取输入的消息之后，会将当前时间和输入的消息分别发送。如果输入的消息是“exit”，会发送特定的消息“exit_request”到服务器端，客户端输出“Exit request send.”服务器接收后将确认接收的消息发送给该客户端，客户端接收到服务器端已接收的消息后，输出“Client: Server confirmed exit.”，然后输出“commoncode: Closing the connection”，“commoncode: Cleaning up WinSock...”以提示关闭套接字和清理 WinSocket 资源。



```
commoncode: Initializing WinSock...
commoncode: WSAStartup is successful.
commoncode: Creating server socket...
commoncode: Setsockopt.
commoncode: Bind.
commoncode: Listen.
Server: Create a server socket.
commoncode: Accepting client connection...
commoncode: Client connected.
vf connected
Server: a thread.
commoncode: Accepting client connection...
commoncode: Client connected.
ve connected
Server: a thread.
commoncode: Accepting client connection...
Server: [ 2023-10-20 16:19:50 ] Received message from client: vf message: wegebewrb
Server: [ 2023-10-20 16:19:57 ] Received message from client: vf message: exit_request
Server: Failed to receive message from client vf. Error: 0
commoncode: Closing socket...

2023-10-20 16:19:50 : Client: Enter your message (type 'exit' if you want to exit): wegebewrb

2023-10-20 16:19:57 : Client: Enter your message (type 'exit' if you want to exit): exit
Exit request sent.
Client: Server confirmed exit. Closing the connection...
commoncode: Closing socket...
commoncode: Cleaning up WinSock...
D:\A_计算机网络\Lab1\Client\x64\Debug\Client.exe (进程 28060)已退出，代码为 0。
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口。

commoncode: Initializing WinSock...
commoncode: WSAStartup is successful.
commoncode: Resolve server IP address.
commoncode: Connect to the server.
##### message #####
Client: Enter your name: ve
Client: Server: [ 2023-10-20 16:19:50 ] Received message from client: vf message: wegebewrb [Broadcast]
```

使用线程同步等待线程退出，最后关闭套接字连接，释放资源。

(4) 时序

时序是对事件发生顺序的详细说明。

(a) 服务器端

启动服务器应用程序，初始化 WinSock 库，创建服务器套接字以侦听客户端连接。进入主循环，等待客户端的连接请求。当有客户端请求连接时，服务器接受连接，并分配唯一标识符（客户端标志）给连接的客户端。为客户端创建一个可用于处理该客户端的消息接收和广播的线程，服务器等待下一个客户端连接请求。

(b) 客户端

启动客户端应用程序，初始化 WinSock 库，创建客户端套接字并连接到服务器。提示用户输入用户名，并将其发送到服务器。创建两个线程：一个线程（Client_Receive）用于接收从服务器发送的消息，另一个线程（Client_Send）用于从用户输入中获取消息并将其发送到服务器。等待从服务器接收到的消息，如果用户输入“exit”，客户端发送退出消息到服务器并关闭套接字。如果用户继续输入消息，客户端将消息发送到服务器，继续等待消息或用户输入。

2. 代码

(1) 多线程

(a) 服务器端

```
//CLIENT newClient;
//newClient.client = acceptClient(serverSocket, &clientAddr, &clientAddrLen);
//接受客户端的连接
clients[num].client = acceptClient(serverSocket, &clientAddr, &clientAddrLen);
clients[num].flag = num;
//接收用户名，检验是否能成功接收
recv(clients[num].client, clients[num].clientname, sizeof(clients[num].clientname), 0); //接收用户名
cout << clients[num].clientname << " connected" << endl;
//receiveMessage(clients[num].client, clients[num].clientname);

//clients.push_back();
DWORD lpThreadId;
HANDLE clientThread;

//创建线程处理消息接收与转发
clientThread = CreateThread(nullptr, 0, (LPTHREAD_START_ROUTINE)handleClient, &clients[num], 0, &lpThreadId);
if (clientThread == nullptr)
{
    cerr << "Server: Failed to create a thread. Error: " << GetCurrentThreadId() << endl;
}
else
{
    cout << "Server: a thread." << endl;
}
// 标记加一
num++;
//lock_guard<mutex> lock(mtx); // 锁定互斥锁，确保线程安全
//clients.push_back(newClient);
```

这段代码在服务器端中创建了一个新的线程（handleClient 函数），用于处理特定客户端的消息接收和广播。服务器端可以同时处理多个客户端的连接，每个客户端都会有一个单独的线程用于消息处理，以实现多人聊天的并发。

```

// 处理单个客户端的线程
DWORD WINAPI handleClient(LPVOID lpParam)
{
    CLIENT* Client = (CLIENT*)lpParam; // 声明并初始化一个 CLIENT 指针
    char timestamp[64] = {}; // 时间戳, 可以是字符串或时间格式
    char content[256]; // 接收的聊天消息内容
    int i = 0;
    char tmp1[] = "[";
    char tmp2[] = " ";
    char tmp3[] = "Received message from client: ";
    char tmp4[] = "message: ";
    char tmp5[] = "Server: ";

    while (true)
    {
        memset(content, 0, sizeof(content));
        //memset(timestamp, 0, sizeof(timestamp));

        //接收数据
        int rcv_time = recv(Client->client_Socket, timestamp, sizeof(timestamp), 0);
        int rcv_con = recv(Client->client_Socket, content, sizeof(content), 0);
        //int rcv_time = receiveMessage(Client->client, timestamp);
        //int ret_con = receiveMessage(Client->client, content);

        //拼接服务器端收到消息的显示格式
        if (rcv_con > 0)
        {
            strcpy_s(Client->content_buf, sizeof(Client->content_buf), tmp5);
            strcat_s(Client->content_buf, sizeof(Client->content_buf), tmp1);
            strcat_s(Client->content_buf, sizeof(Client->content_buf), timestamp);
            strcat_s(Client->content_buf, sizeof(Client->content_buf), tmp2);
            strcat_s(Client->content_buf, sizeof(Client->content_buf), tmp3);
            strcat_s(Client->content_buf, sizeof(Client->content_buf), Client->clientname);
            strcat_s(Client->content_buf, sizeof(Client->content_buf), tmp4);
            strcat_s(Client->content_buf, sizeof(Client->content_buf), content);
            //服务器端的显示
            cout << Client->content_buf << endl;

            //lock_guard<mutex> lock(mutex); //使用互斥锁以确保多个线程安全地操作 clients vector, 保护共享资源 clients

            if (strcmp(content, "exit_request") != 0)
            {
                for (i = 0; i < num; i++)
                {
                    if (i != Client->flag)
                    {
                        // 发送回复消息给客户端
                        SOCKET send_result = send(clients[i].client_Socket, Client->content_buf, strlen(Client->content_buf), 0);
                        //sendMessage(Client->client, Client->buf);
                        if (send_result == SOCKET_ERROR) // 处理发送错误
                        {
                            cerr << "Server: Failed to send message to client " << Client->clientname << ". Error: " << WSAGetLastError() << endl;
                        }
                    }
                }
            }
            else
            {
                for (i = 0; i < num; i++)
                {
                    if (i == Client->flag)
                    {
                        SOCKET send_result = send(clients[i].client_Socket, content, strlen(content), 0);
                        if (send_result == SOCKET_ERROR) // 处理发送错误
                        {
                            cerr << "Server: Failed to send message to client " << Client->clientname << ". Error: " << WSAGetLastError() << endl;
                        }
                    }
                }
            }
            else
            {
                cout << "Server: Failed to receive message from client " << Client->clientname << ". Error: " << WSAGetLastError() << endl;
                break;
            }
        }
    }
}

```

DWORD 是 windows 环境中定义的数据类型, 本质就是 unsigned int 型
 WINAPI 是一个宏, 所代表的符号是 __stdcall, 函数名前加上这个符号表示
 这个函数的调用约定是标准调用约定, windows API 函数采用这种调用约定。

LPVOID 是一个没有类型的指针, 也就是可以将任意类型的指针赋值给
 LPVOID 类型的变量 (一般作为参数传递), 然后在使用的时候在转换回来。

(b) 客户端

```
HANDLE ThreadReceive, ThreadSend;

ThreadReceive = CreateThread(NULL, 0, Client_Receive, (LPVOID)clientSocket, 0, NULL);
ThreadSend = CreateThread(NULL, 0, Client_Send, (LPVOID)clientSocket, 0, NULL);

// 使用线程同步等待线程退出
WaitForSingleObject(ThreadReceive, INFINITE);
WaitForSingleObject(ThreadSend, INFINITE);

//进程关闭, 主函数返回
if (WaitForSingleObject(ThreadReceive, INFINITE) == WAIT_OBJECT_0 ||
    WaitForSingleObject(ThreadSend, INFINITE) == WAIT_OBJECT_0)
{
    CloseHandle(ThreadReceive);
    CloseHandle(ThreadSend);
    return 0;
}
```

在客户端代码中，创建了两个线程，一个用于接收从服务器发送的消息，另一个用于从用户输入中获取消息并将其发送到服务器。这允许客户端同时进行消息接收和消息发送操作。

```
// 客户端线程, 用于接收消息
DWORD WINAPI Client_Receive(LPVOID lparam)
{
    SOCKET Client = (SOCKET) (LPVOID) lparam;
    char messagebuffer[256] = { 0 };
    int recv_buffer = 0;

    while (true)
    {
        memset(messagebuffer, 0, sizeof(messagebuffer));
        //int rec=receiveMessage(Client, messagebuffer);
        recv_buffer = recv(Client, messagebuffer, sizeof(messagebuffer), 0);

        if (recv_buffer < 0)
        {
            cout << "Client: Failed to receive the message. Error: " << WSAGetLastError() << endl;
            break;
        }
    }
}
```

```
    else
    {
        if (strcmp(messagebuffer, "exit request") == 0)
        {
            cout << "Client: Server confirmed exit. Closing the connection..." << endl;
            closeSocket(Client);
            cleanupWinSock();
            break;
            return 0;
        }

        else
        {
            cout << "Client: " << messagebuffer << "【Broadcast】" << endl;
        }
    }
}

// 关闭客户端套接字和线程退出
//closeSocket(Client);
return 0;
```



```

DWORD WINAPI Client_Send(LPVOID lparam)
{
    SOCKET clientSocket = (SOCKET) (LPVOID) lparam;
    char message[256] = { 0 };
    char SendTime[64] = {};
    int send_time = 0;
    int send_message = 0;
    int send_exit = 0;
    char Exit[15];
    strcpy_s(Exit, sizeof(Exit), "exit_request");

    while (true && (exitRequested == false))
    {
        memset(message, 0, sizeof(message));
        memset(clientname, 0, sizeof(clientname));
        memset(SendTime, 0, sizeof(SendTime));

        string time = getTime(); // 获取当前时间
        strcpy_s(SendTime, sizeof(SendTime), time.c_str());

        cin.ignore((numeric_limits<streamsize>::max)(), '\n');

        cout << SendTime << " " << clientname << ": ";
        cout << "Client: Enter your message (type 'exit' if you want to exit): ";
        cin.getline(message, sizeof(message));

        cin.ignore((numeric_limits<streamsize>::max)(), '\n');

        if (strcmp(message, "exit") == 0)
        {
            send_time = send(clientSocket, SendTime, strlen(SendTime), 0);
            // send_message = send(clientSocket, message, strlen(message), 0);
            // 发送特殊的退出请求消息给服务器
            send_exit = send(clientSocket, Exit, strlen(Exit), 0);
            if (send_exit == SOCKET_ERROR)
            {
                cerr << "Client: Failed to send exit request. Error: " << WSAGetLastError() << endl;
                // 这里可以考虑进行错误处理, 例如重新发送或关闭连接
            }
            else
            {
                cout << "Exit request sent." << endl;

                // send(clientSocket, "exit_request", strlen("exit_request"), 0);
                exitRequested = true;
                return 0;
            }
        }
        else
        {
            send_time = send(clientSocket, SendTime, strlen(SendTime), 0);
            send_message = send(clientSocket, message, strlen(message), 0);
        }
    }

    return 0;
}

```

多线程的使用使得服务器端和客户端能够并发地处理消息, 允许多个用户同时连接和聊天, 而不会阻塞主线程。

```
HANDLE CreateThread(  
    LPSECURITY_ATTRIBUTES lpThreadAttributes,  
    DWORD dwStackSize,  
    LPTHREAD_START_ROUTINE lpStartAddress,  
    LPVOID lpParameter,  
    DWORD dwCreationFlags,  
    LPDWORD lpThreadId  
);
```

lpThreadAttributes: 指向 SECURITY_ATTRIBUTES 的指针，用于定义新线程的安全属性，一般设置成 NULL；

dwStackSize: 分配以字节数表示的线程堆栈的大小，默认值是 0；

lpStartAddress: 指向一个线程函数地址。每个线程都有自己的线程函数，线程函数是线程具体的执行代码；

lpParameter: 传递给线程函数的参数；

dwCreationFlags: 表示创建线程的运行状态，其中 CREATE_SUSPEND 表示挂起当前创建的线程，而 0 表示立即执行当前创建的进程；

lpThreadId: 返回新创建的线程的 ID 编号；

```
DWORD WaitForSingleObject(  
    HANDLE hHandle,  
    DWORD dwMilliseconds  
);
```

hHandle: 指定对象或时间的句柄；

dwMilliseconds: 等待时间，以毫秒为单位，当超过等待时间时，此函数返回。如果参数设置为 0，则该函数立即返回；如果设置成 INFINITE，则该函数直到有信号才返回。

(2) 函数调用过程

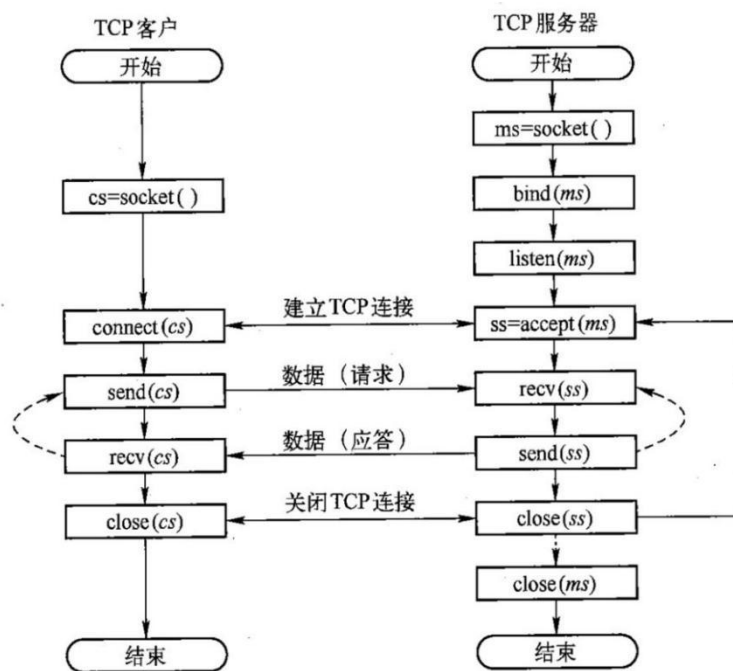
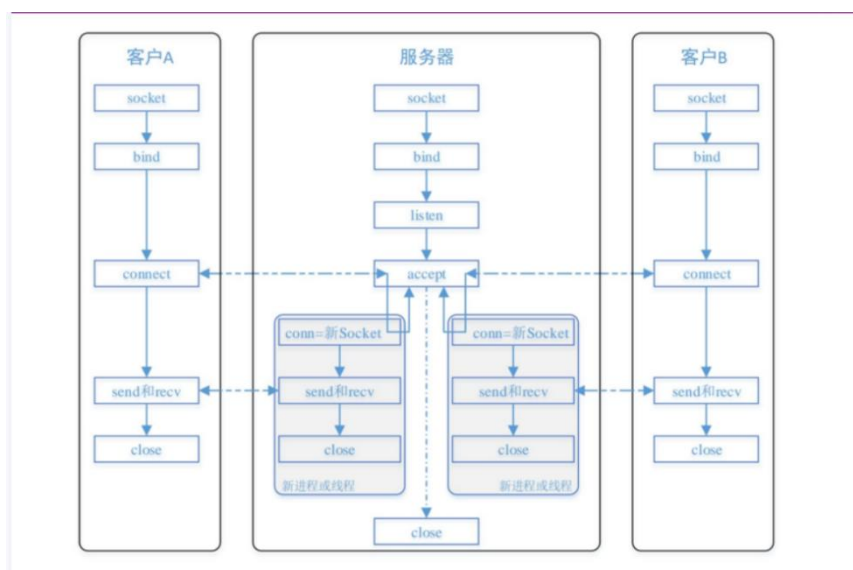


图 2.21 基于 TCP 客户与服务器的典型 Socket API 函数调用过程



① 服务器端

(a) 初始化 WinSock 库

```

// 初始化WinSock库
int initializeWinSock()
{
    cout << "commoncode: Initializing WinSock..." << endl;
    WSADATA wsaData; //wsaData用来存储系统传回的关于WINSOCK的资料.
    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) //MAKEWORD(2, 2)表示使用WINSOCK2版本.
    {
        cout << "commoncode: WSAStartup failed." << endl;
        return -1;
    }
    else
    {
        cout << "commoncode: WSAStartup is successful." << endl;
        return 0;
    }
}

```

WSAStartup();主要就是进行相应的 socket 库绑定。

函数原型: int WSAStartup (WORD wVersionRequested, LPWSADATA lpWSADATA);

使用 Socket 的程序在使用 Socket 之前必须调用 WSAStartup 函数。以后应用程序就可以调用所请求的 Socket 库中的其它 Socket 函数了，然后绑定找到的 Socket 库到该应用程序中。该函数执行成功后返回 0。

如果成功初始化，输出“commoncode: WSAStartup is successful.”，如果不成功，会输出“commoncode: WSAStartup failed.”

(b) 创建套接字

```

// 创建服务器套接字
SOCKET createServerSocket(int port)
{
    cout << "commoncode: Creating server socket..." << endl;
    // 创建一个套接字，使用IPv4地址族(AF_INET)，流式套接字类型(SOCK_STREAM)，和TCP协议(IPPROTO_TCP)
    SOCKET serverSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (serverSocket == INVALID_SOCKET)
    {
        cerr << "commoncode: Create socket failed. Error: " << WSAGetLastError() << endl;
        return -1; // 返回-1表示创建套接字失败
    }
}

```

Socket 的第一个参数是 domain，表示套接字的域名，代表套接字的地址族，这里使用 IPv4。

如果创建成功，输出“commoncode: Creating server socket...”，如果失败，则输出“commoncode: Create socket failed.Error: 上一次错误的代码”。

域名	地址族
AF_UNIX,AF_LOCAL	用于本地通信
AF_INET,PF_INET	IPv4, Internet协议
AF_INET6	IPv6,Internet协议
AF_IPX	Novell网络协议

流式套接字（SOCK_STREAM）：提供了面向连接的、可靠的、数据无错并且无重复的数据发送服务，而且发送的数据时按顺序被接受的。所有利用该套接字进行传递的数据均被视为连续的字节流且无长度限制。这对数据的稳定性、正确性和发送/接受顺序要求严格的应用十分适用，TCP 协议使用该接口，但其对线路的占用率相对比较高。远程登录（TELNET）、文件传输协议（FTP）等使用了流式套接字。

```
//创建一个Socket来监听客户端的连接请求。
int serverPort = 5500;
int serverSocket = createServerSocket(serverPort);

if (serverSocket < 0)
{
    cerr << "Server: Failed to create a server socket. Error: " << WSAGetLastError() << endl;
    cleanupWinSock();
    return 1;
}
else
{
    cerr << "Server: Create a server socket. " << endl;
}
```

在服务器端，如果成功创建套接字，输出“Server: Create a server socket.”，如果失败，则输出“Server: Failed to create a server socket. Error: 上一次错误的代码”

(c) 绑定地址

```
// 配置服务器地址结构
sockaddr_in serverAddr; //通用套接字地址
serverAddr.sin_family = AF_INET; // 地址族, IPv4
serverAddr.sin_port = htons(port); //端口号, 需要使用htons函数将端口号从主机字节序转换为网络字节序
serverAddr.sin_addr.s_addr = INADDR_ANY; //ip地址, INADDR_ANY 表示套接字可以接受来自本机上的所有网络接口（即所有可用的 IP 地址）的连接请求。
```

```
// 设置SO_REUSEADDR选项, 用于任意类型、任意状态套接口的设置选项值, 以允许套接字地址被重用
int reuse = 1;
if (setsockopt(serverSocket, SOL_SOCKET, SO_REUSEADDR, (const char*)&reuse, sizeof(int)) == SOCKET_ERROR)
{
    cerr << "commoncode: Setsockopt failed. Error: " << WSAGetLastError() << endl;
    closesocket(serverSocket);
    return -4; // 返回-4表示设置选项失败
}
else
{
    cout << "commoncode: Setsockopt." << endl;
}

//将套接字绑定到指定的端口和IP地址
if (bind(serverSocket, (struct sockaddr*)&serverAddr, sizeof(serverAddr)) == SOCKET_ERROR)
{
    cerr << "commoncode: Bind failed. Error: " << WSAGetLastError() << endl;
    closesocket(serverSocket);
    return -2;
}
else
{
    cout << "commoncode: Bind." << endl;
}
```

使用 bind() 将套接字绑定到指定的协议族。

```
int bind(int sockfd, const struct sockaddr *myaddr, socklen_t addrlen);
```

返回值：0 表示成功，-1 表示失败，errno 记录错误代码。

如果成功绑定，输出“commoncode: Bind. ”，如果未能成功绑定，输出“commoncode: Bind failed. Error: 上一次错误的代码”

(d) 监听

```
// 开始监听连接请求，最多允许10个等待连接
if (listen(serverSocket, 10) == SOCKET_ERROR)
{
    cerr << "commoncode: Listen failed. Error: " << WSAGetLastError() << endl;
    closesocket(serverSocket);
    return -3; // 返回-3表示监听失败
}
else
{
    cout << "commoncode: Listen." << endl;
}

return serverSocket; // 返回创建的服务器套接字
```

int listen(int sockfd, int backlog);

返回值：0 成功，-1 失败。errno 记录错误代码

sockfd 表示用于监听的套接字，backlog 连接队列的长度。是指完成 TCP 三次握手之后已经成功建立 TCP 连接的队列长度。服务器执行 accept () 操作时，从该队列中取下一个连接进行后续处理。

如果成功监听，输出“commoncode: Listen. ”，如果未能成功绑定，输出“commoncode: Listen failed. Error: 上一次错误的代码”

(e) 接受客户端连接

```
//等待客户端的连接
sockaddr_in clientAddr;
int clientAddrLen = sizeof(clientAddr);
```

```
// 接受客户端连接
SOCKET acceptClient(SOCKET serverSocket, sockaddr_in* clientAddr, int* clientAddrLen)
{
    cout << "commoncode: Accepting client connection..." << endl;
    // 使用 accept 函数接受客户端的连接请求，返回一个新的套接字用于与客户端通信
    SOCKET clientSocket = accept(serverSocket, (struct sockaddr*)clientAddr, clientAddrLen);
    // 检查是否接受连接请求失败
    if (clientSocket == SOCKET_ERROR)
    {
        cerr << "commoncode: Accept failed. Error: " << WSAGetLastError() << endl;
        return -1; // 返回-1表示接受连接失败
    }
    cout << "commoncode: Client connected." << endl;
    return clientSocket; // 返回与客户端通信的套接字
}
```

int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);

accept 将客户端的信息绑定到一个 socket 上，也就是给客户端创建一个 socket，通过返回值返回给我们客户端的 socket。一次只能创建一个，有几个客户端链接，就要调用几次。

(f) 接收信息

```
// 从套接字接收消息
int receiveMessage(SOCKET socket, char* message)
{
    cout << "commoncode: Receiving message..." << endl;
    int bytesRead = recv(socket, message, sizeof(message), 0);
    if (bytesRead == SOCKET_ERROR)
    {
        cerr << "commoncode: Error receiving message. Error code: " << WSAGetLastError() << endl;
    }
    return bytesRead;
}
```

int recv(int sockfd, void *buff, size_t nbytes, int flags);

sockfd 接收端套接字描述符，buff 用来存放 recv 函数接收到的数据的缓冲区，nbytes 指明 buff 的长度，flags 一般置为 0。失败时，返回值小于 0；超时或对端主动关闭，返回值等于 0；成功时，返回值是返回接收数据的长度。

recv 先等待 sockfd 的发送缓冲区的数据被协议传送完毕，如果协议在传送 sock 的发送缓冲区中的数据时出现网络错误，那么 recv 函数返回 SOCKET_ERROR。当协议把数据接收完毕，recv 函数就把 sockfd 的接收缓冲区中的数据 copy 到 buff 中(协议接收到的数据可能大于 buff 的长度，所以在这种情况下要调用几次 recv 函数才能把 sockfd 的接收缓冲区中的数据 copy 完。recv 函数仅仅是 copy 数据，真正的接收数据是协议来完成的)。

如果未能接收到消息，输出 “commoncode: Error receiving message. Error code: 上一次错误的代码”

(g) 发送消息

```
// 发送消息到套接字
int sendMessage(SOCKET socket, const char* message)
{
    cout << "commoncode: Sending message..." << endl;
    int bytesSent = send(socket, message, sizeof(message), 0);
    if (bytesSent == SOCKET_ERROR)
    {
        cerr << "commoncode: Error sending message. Error code: " << WSAGetLastError() << endl;
    }
    return bytesSent;
}
```

int send(int sockfd, const void *buff, size_t nbytes, int flags);

sockfd 是指定发送端套接字描述符，buff 存放要发送数据的缓冲区，nbytes 是实际要改善的数据的字节数，flags 一般设置为 0。 当调用该函数时，send() 先

比较待发送数据的长度和套接字的发送缓冲区的长度：当待拷贝数据的长度大于发送缓冲区的长度时，该函数返回 `SOCKET_ERROR`；当待拷贝数据的长度小于或等于发送缓冲区的长度时，那么 `send` 先检查协议是否正在发送套接字的发送缓冲区中的数据：如果是就等待协议把数据发送完，再进行拷贝；如果协议还没有开始发送套接字的发送缓冲区中的数据或者该发送缓冲区中没有数据，那么 `send` 就比较该发送缓冲区中的剩余空间和待拷贝数据的长度：如果待拷贝数据的长度大于剩余空间的大小，`send` 就一直等待协议把该发送缓冲区中的数据发完；如果待拷贝数据的长度小于剩余空间大小，`send` 就仅仅把 `buf` 中的数据拷贝到剩余空间中。（注意：并不是 `send` 把该套接字的发送缓冲区中数据传到连接的另一端，而是协议传的，`send` 仅仅是把数据拷贝到该发送缓冲区的剩余空间里面。）如果 `send` 函数拷贝成功，就返回实际拷贝的字节数；如果拷贝的过程中出现错误，`send` 就返回 `SOCKET_ERROR`；如果 `send` 在等待协议传送数据时网络断开的话，那么 `send` 函数也返回 `SOCKET_ERROR`。

如果未能接发送消息，输出 “commoncode: Error sending message. Error code: 上一次错误的代码”

(h) 关闭套接字，清理 WinSocket 资源

```
// 清理WinSock库资源
void cleanupWinSock()
{
    cout << "commoncode: Cleaning up WinSock..." << endl;
    WSACleanup();
}

// 关闭套接字
void closeSocket(SOCKET socket)
{
    cout << "commoncode: Closing socket..." << endl;
    closesocket(socket);
}
```

输出 “commoncode: Cleaning up WinSock...” 和 “commoncode: Closing socket...”

② 客户端

(a) 初始化 WinSock 库

```

// 初始化WinSock库
int initializeWinSock()
{
    cout << "commoncode: Initializing WinSock..." << endl;
    WSADATA wsaData; //wsaData用来存储系统传回的关于WINSOCK的资料.
    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) //MAKEWORD(2, 2)表示使用WINSOCK2版本.
    {
        cout << "commoncode: WSAStartup failed." << endl;
        return -1;
    }
    else
    {
        cout << "commoncode: WSAStartup is successful." << endl;
        return 0;
    }
}

```

如果成功初始化，输出“commoncode: WSAStartup is successful.”，如果不成
功，会输出“commoncode: WSAStartup failed.”

(b) 创建套接字

```

//创建Socket: 在客户端创建一个Socket来连接服务器。
SOCKET clientSocket = createClientSocket("127.0.0.1", 5500);
cout << "##### message #####" << endl;

```

```

// 创建客户端套接字
SOCKET createClientSocket(const char* serverIP, int serverPort)
{
    SOCKET clientSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (clientSocket == INVALID_SOCKET)
    {
        cerr << "commoncode: Failed to create a client socket. Error: " << WSAGetLastError() << endl;
        cleanupWinSock();
        return -2;
    }

    sockaddr_in serverAddr;
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(serverPort);

    // 解析服务器 IP 地址
    if (inet_pton(AF_INET, serverIP, &serverAddr.sin_addr) <= 0)
    {
        cerr << "commoncode: Failed to resolve server IP address. Error: " << WSAGetLastError() << endl;
        closeSocket(clientSocket);
        cleanupWinSock();
        return -3;
    }
    else
    {
        cout << "commoncode: Resolve server IP address." << endl;
    }
}

```

如果创建套接字失败，则输出“commoncode: Failed to create a client socket.
Error: 上次的错误代码”

```
int inet_pton(int family, const char *strptr, void *addrptr);
```

将点分十进制的 ip 地址转化为用于网络传输的数值格式。该函数将字符串
strptr 转换为 family 地址类型协议簇的网络地址，并存储到 addrptr 中。对于
family 参数，AF_INET 或 AF_INET6 均可（IPv4 和 IPv6）

返回值：若成功则为 1，若输入不是有效的表达式则为 0，若出错则为-1 解析服务器的 IP 地址。

如果解析成功，则输出 “commoncode: Resolve server IP address.”，如果失败，则输出 “commoncode: Failed to resolve server IP address. Error: 上次的错误代码 ”

(c) 连接到服务器

```
// 尝试连接到服务器
if (connect(clientSocket, (struct sockaddr*)&serverAddr, sizeof(serverAddr)) < 0)
{
    cerr << "commoncode: Failed to connect to the server. Error: " << WSAGetLastError() << endl;
    closeSocket(clientSocket);
    cleanupWinSock();
    return -4;
}
else
{
    cout << "commoncode: Connect to the server." << endl;
}

return clientSocket;
```

int connect(int sockfd, const struct sockaddr *servaddr, socklen_t addrlen)

返回：若成功则为 0，若出错则为-1。sockfd 是由 socket 函数返回的套接字描述符，servaddr、sizeof (serveraddr) 分别是一个指向套接字地址结构的指针和该结构的大小。套接字地址结构必须含有服务器的 IP 地址和端口号。

如果连接成功，输出 “commoncode: Connect to the server.”，连接失败，输出 “commoncode: Failed to connect to the server. Error: 上一次的错误代码”

(d) 发送消息

```
// 发送消息到套接字
int sendMessage(SOCKET socket, const char* message)
{
    cout << "commoncode: Sending message..." << endl;
    int bytesSent = send(socket, message, sizeof(message), 0);
    if (bytesSent == SOCKET_ERROR)
    {
        cerr << "commoncode: Error sending message. Error code: " << WSAGetLastError() << endl;
    }
    return bytesSent;
}
```

(e) 接收消息

```

// 从套接字接收消息
int receiveMessage(SOCKET socket, char* message)
{
    cout << "commoncode: Receiving message..." << endl;
    int bytesRead = recv(socket, message, sizeof(message), 0);
    if (bytesRead == SOCKET_ERROR)
    {
        cerr << "commoncode: Error receiving message. Error code: " << WSAGetLastError() << endl;
    }
    return bytesRead;
}

```

(f) 关闭套接字，清理 WinSocket 资源

```

// 清理WinSock库资源
void cleanupWinSock()
{
    cout << "commoncode: Cleaning up WinSock..." << endl;
    WSACleanup();
}

// 关闭套接字
void closeSocket(SOCKET socket)
{
    cout << "commoncode: Closing socket..." << endl;
    closesocket(socket);
}

```

三、 运行说明

(1) 正常情况

以 3 个客户端为例实现多人聊天室。以下是正常聊天的情况。

服务器为各个客户端创建线程，一个客户端输入非“exit”的消息之后，服务器接收并广播给其他客户端，为区别是广播的输出还是输入的输出，广播输出的字符串最后多了“【Broadcast】”，以此作为区分的标志。

```

D:\A_计算机网络的Lab\Server x + -
commoncode: Setsockopt.
commoncode: Bind.
commoncode: Listen.
Server: Create a server socket.
commoncode: Accepting client connection...
commoncode: Client connected.
a connected
Server: a thread.
commoncode: Accepting client connection...
commoncode: Client connected.
b connected
Server: a thread.
commoncode: Accepting client connection...
commoncode: Client connected.
c connected
Server: a thread.
commoncode: Accepting client connection...

D:\A_计算机网络的Lab\Client2 x + -
commoncode: Initializing WinSock...
commoncode: WSASStartup is successful.
commoncode: Resolve server IP address.
commoncode: Connect to the server.
##### message #####
Client: Enter your name: b

D:\A_计算机网络的Lab\Client x + -
commoncode: Initializing WinSock...
commoncode: WSASStartup is successful.
commoncode: Resolve server IP address.
commoncode: Connect to the server.
##### message #####
Client: Enter your name: a

D:\A_计算机网络的Lab\Client3 x + -
commoncode: Initializing WinSock...
commoncode: WSASStartup is successful.
commoncode: Resolve server IP address.
commoncode: Connect to the server.
##### message #####
Client: Enter your name: c

```

```
D:\A\计算机网络的Lab\Server x + - □ ×
a connected
Server: a thread.
commoncode: Accepting client connection...
commoncode: Client connected.
b connected
Server: a thread.
commoncode: Accepting client connection...
commoncode: Client connected.
c connected
Server: a thread.
commoncode: Accepting client connection...
Server: [ 2023-10-20 16:34:50 ] Received message from client: b message: csv
dsvfbgneyn
Server: [ 2023-10-20 16:34:53 ] Received message from client: c message: dvf
wbthyrnuin6ngfv wqe
Server: [ 2023-10-20 16:34:46 ] Received message from client: a message: ewr
vtyh786kiurhbgvfc2e

D:\A\计算机网络的Lab\Client2 x + - □ ×
commoncode: Initializing WinSock...
commoncode: WSASStartup is successful.
commoncode: Resolve server IP address.
commoncode: Connect to the server.
##### message #####
Client: Enter your name: b
2023-10-20 16:34:50 : Client: Enter your message (type 'exit' if you want to
exit): csdsvfbgneyn
Client: Server: [ 2023-10-20 16:34:53 ] Received message from client: c mess
age: dvfwbthyrnuin6ngfv wqe [Broadcast]
Client: Server: [ 2023-10-20 16:34:46 ] Received message from client: a mess
age: ewrvtyh786kiurhbgvfc2e [Broadcast]

D:\A\计算机网络的Lab\Client x + - □ ×
commoncode: Initializing WinSock...
commoncode: WSASStartup is successful.
commoncode: Resolve server IP address.
commoncode: Connect to the server.
##### message #####
Client: Enter your name: a
Client: Server: [ 2023-10-20 16:34:50 ] Received message from client: b mess
age: csdsvfbgneyn [Broadcast]
Client: Server: [ 2023-10-20 16:34:53 ] Received message from client: c mess
age: dvfwbthyrnuin6ngfv wqe [Broadcast]
2023-10-20 16:34:46 : Client: Enter your message (type 'exit' if you want to
exit): ewrvtyh786kiurhbgvfc2e

D:\A\计算机网络的Lab\Client3 x + - □ ×
commoncode: Initializing WinSock...
commoncode: WSASStartup is successful.
commoncode: Resolve server IP address.
commoncode: Connect to the server.
##### message #####
Client: Enter your name: c
Client: Server: [ 2023-10-20 16:34:50 ] Received message from client: b mess
age: csdsvfbgneyn [Broadcast]
2023-10-20 16:34:53 : Client: Enter your message (type 'exit' if you want to
exit): dvfwbthyrnuin6ngfv wqe
Client: Server: [ 2023-10-20 16:34:46 ] Received message from client: a mess
age: ewrvtyh786kiurhbgvfc2e [Broadcast]
```

(2) 当其中一个输出“exit”想要退出，截图中是Client2输入“exit”。

```
D:\A\计算机网络的Lab\Server x + - □ ×
b connected
Server: a thread.
commoncode: Accepting client connection...
commoncode: Client connected.
c connected
Server: a thread.
commoncode: Accepting client connection...
Server: [ 2023-10-20 16:34:50 ] Received message from client: b message: csv
dsvfbgneyn
Server: [ 2023-10-20 16:34:53 ] Received message from client: c message: dvf
wbthyrnuin6ngfv wqe
Server: [ 2023-10-20 16:34:46 ] Received message from client: a message: ewr
vtyh786kiurhbgvfc2e
Server: [ 2023-10-20 16:35:50 ] Received message from client: b message: exi
t_request
Server: Failed to receive message from client b. Error: 0
commoncode: Closing socket...

Microsoft Visual Studio 调试 x + - □ ×
Client: Server: [ 2023-10-20 16:34:53 ] Received message from client: c mess
age: dvfwbthyrnuin6ngfv wqe [Broadcast]
Client: Server: [ 2023-10-20 16:34:46 ] Received message from client: a mess
age: ewrvtyh786kiurhbgvfc2e [Broadcast]
2023-10-20 16:35:50 : Client: Enter your message (type 'exit' if you want to
exit): exit
Exit request sent.
Client: Server confirmed exit. Closing the connection...
commoncode: Closing socket...
commoncode: Cleaning up WinSock...
D:\A\计算机网络的Lab\Client2.exe (进程 24656)已退出，代码
为 0。
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动
关闭控制台”。
按住任意键关闭此窗口。 . . .

D:\A\计算机网络的Lab\Client x + - □ ×
commoncode: Initializing WinSock...
commoncode: WSASStartup is successful.
commoncode: Resolve server IP address.
commoncode: Connect to the server.
##### message #####
Client: Enter your name: a
Client: Server: [ 2023-10-20 16:34:50 ] Received message from client: b mess
age: csdsvfbgneyn [Broadcast]
Client: Server: [ 2023-10-20 16:34:53 ] Received message from client: c mess
age: dvfwbthyrnuin6ngfv wqe [Broadcast]
2023-10-20 16:34:46 : Client: Enter your message (type 'exit' if you want to
exit): ewrvtyh786kiurhbgvfc2e

D:\A\计算机网络的Lab\Client3 x + - □ ×
commoncode: Initializing WinSock...
commoncode: WSASStartup is successful.
commoncode: Resolve server IP address.
commoncode: Connect to the server.
##### message #####
Client: Enter your name: c
Client: Server: [ 2023-10-20 16:34:50 ] Received message from client: b mess
age: csdsvfbgneyn [Broadcast]
2023-10-20 16:34:53 : Client: Enter your message (type 'exit' if you want to
exit): dvfwbthyrnuin6ngfv wqe
Client: Server: [ 2023-10-20 16:34:46 ] Received message from client: a mess
age: ewrvtyh786kiurhbgvfc2e [Broadcast]
```

Client2 将字符串“exit_request”发送给 server，server 接收到这个字符串后，将这个字符串发送到 Client2，接收到这个字符串 Client2 知道服务器已经知道 Client2 要退出了，于是就 CloseSocket(), cleanupWinSock(), 关闭连接。

```
D:\A\计算机网络的Lab\Server x + - □ ×
commoncode: Client connected.
c connected
Server: a thread.
commoncode: Accepting client connection...
Server: [ 2023-10-20 16:34:50 ] Received message from client: b message: csv
dsvfbgneyn
Server: [ 2023-10-20 16:34:53 ] Received message from client: c message: dvf
wbthyrnuin6ngfv wqe
Server: [ 2023-10-20 16:34:46 ] Received message from client: a message: ewr
vtyh786kiurhbgvfc2e
Server: [ 2023-10-20 16:35:50 ] Received message from client: b message: exi
t_request
Server: Failed to receive message from client b. Error: 0
commoncode: Closing socket...
Server: [ 2023-10-20 16:36:02 ] Received message from client: a message: cds
Server: [ 2023-10-20 16:35:57 ] Received message from client: c message: efw
fevgv

Microsoft Visual Studio 调试 x + - □ ×
Client: Server: [ 2023-10-20 16:34:53 ] Received message from client: c mess
age: dvfwbthyrnuin6ngfv wqe [Broadcast]
Client: Server: [ 2023-10-20 16:34:46 ] Received message from client: a mess
age: ewrvtyh786kiurhbgvfc2e [Broadcast]
2023-10-20 16:35:50 : Client: Enter your message (type 'exit' if you want to
exit): exit
Exit request sent.
Client: Server confirmed exit. Closing the connection...
commoncode: Closing socket...
commoncode: Cleaning up WinSock...
D:\A\计算机网络的Lab\Client2.exe (进程 24656)已退出，代码
为 0。
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动
关闭控制台”。
按住任意键关闭此窗口。 . . .

D:\A\计算机网络的Lab\Client x + - □ ×
commoncode: Connect to the server.
##### message #####
Client: Enter your name: a
Client: Server: [ 2023-10-20 16:34:50 ] Received message from client: b mess
age: csdsvfbgneyn [Broadcast]
Client: Server: [ 2023-10-20 16:34:53 ] Received message from client: c mess
age: dvfwbthyrnuin6ngfv wqe [Broadcast]
2023-10-20 16:34:46 : Client: Enter your message (type 'exit' if you want to
exit): ewrvtyh786kiurhbgvfc2e
2023-10-20 16:36:02 : Client: Enter your message (type 'exit' if you want to
exit): cds
Client: Server: [ 2023-10-20 16:35:57 ] Received message from client: c mess
age: efwfevgv [Broadcast]

D:\A\计算机网络的Lab\Client3 x + - □ ×
commoncode: Connect to the server.
##### message #####
Client: Enter your name: c
Client: Server: [ 2023-10-20 16:34:50 ] Received message from client: b mess
age: csdsvfbgneyn [Broadcast]
2023-10-20 16:34:53 : Client: Enter your message (type 'exit' if you want to
exit): dvfwbthyrnuin6ngfv wqe
Client: Server: [ 2023-10-20 16:34:46 ] Received message from client: a mess
age: ewrvtyh786kiurhbgvfc2e [Broadcast]
Client: Server: [ 2023-10-20 16:36:02 ] Received message from client: a mess
age: cds [Broadcast]
2023-10-20 16:35:57 : Client: Enter your message (type 'exit' if you want to
exit): efwfevgv
```

Client2 退出后，其他的两个客户端可以正常发送和接收消息。

四、 实验过程中遇到的问题及分析

1. 该程序最多可容纳 20 人进行多人聊天，并不是可以无限多人进行聊天的聊天室。曾尝试使用 `vector<CLINET>` 和互斥锁机制等代替 `clients[20]`，但输出报错，`vector out of range`。

2. 无法建立连接可能是由于网络配置问题、防火墙设置、端口冲突或服务器未正常运行等原因引起的。需要检查网络连接、端口配置以及服务器是否在运行。

3. 客户端无法完整接收消息，可能是因为接收缓冲区大小不够，导致的接收异常。

4. 在网络通信中，消息可能会丢失或以不同顺序到达接收方，特别是在不可靠的网络环境中。消息丢失可能是由于网络问题引起的，如网络拥塞或丢包。乱序可能是由于消息在网络中经历不同的路径导致的。为解决这些问题，可以引入序列号和确认机制，确保消息的完整性和顺序。

5. 如果想实现多人聊天室的私聊功能，可多接收并发送 `receiver_name` 的字符串，通过 `clientname` 和 `flag` 标记找到这个 `receiver` 客户端线程后，发送消息即可。

6. 可继续增加实现文件传输，数据共享，远程数据访问等功能，需要进一步添加相应的链接使用相关的 API 函数。