

实验 1：利用 Socket，设计和编写一个聊天程序

实验要求

- 1) 使用流式 **Socket**，设计一个两人聊天协议，要求聊天信息带有时间标签。请完整地说明交互消息的类型、语法、语义、时序等具体的消息处理方式。
- 2) 对聊天程序进行设计。给出模块划分说明、模块的功能和模块的流程图。
- 3) 在 **Windows** 系统下，利用 **C/C++**对设计的程序进行实现。程序界面可以采用命令行方式，但需要给出使用方法。编写程序时，只能使用基本的 **Socket** 函数，不允许使用对 **socket** 封装后的类或架构。
- 4) 对实现的程序进行测试。
- 5) 撰写实验报告，并将实验报告和源码提交至本网站。

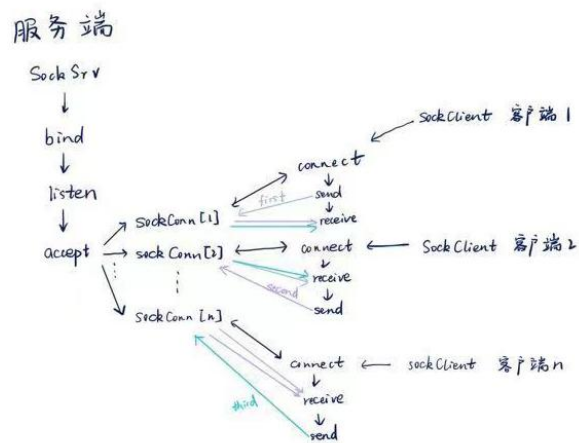
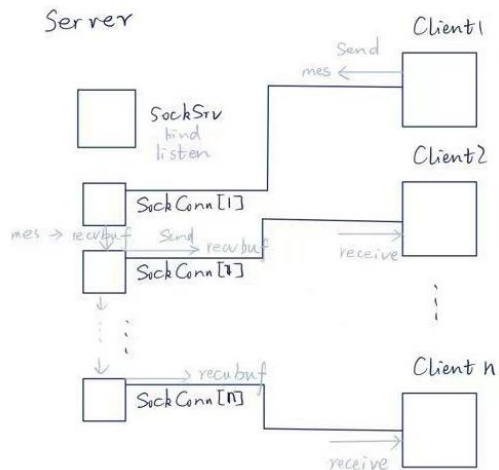
实验内容

协议设计

- **语法**（怎么做，字段分割，哪些字段）
发送的信息支持中英文字符，内容发送格式为：发送时间+“ ”+信息的具体内容
- **语义**（做什么，信息具体含义）
发送方发送“over”时，意味着发送方想发的消息发送完成；
发送方发送“exit”，意味着发送方结束对话。
- **处理**（何时发送，收到消息的动作）
服务器端从发送方接收到消息，处理信息，获得发送信息的最后四个字符，然后发送给所有接收方；
服务器端收到“over”消息时，结束从发送方接收消息，开始从接收方接收消息；
服务器端收到“exit”消息时，关闭连接的套接字；
接收方收到消息时，进行处理，获得接收信息的最后四个字符；
接收方收到普通消息时，正常打印；
接收方收到“over”消息时，不显示消息内容，确定下一个需要发送消息的发送方，若是自己，则开始发送消息功能，否则继续接收消息；

接收方收到“exit”消息时，打印“对方结束对话”，结束对话；

程序设计



程序实现

实现基本功能

完成客户端与服务端的连接，完成客户端之间发送和接受消息。

客户端代码

创建客户端套接字

```

//初始化Socket DLL, 使用2.2版本的socket
WORD wVersionRequested;
WSADATA wsaData;
/* Use the MAKEWORD(lowbyte, highbyte) macro declared in Windef.h */
wVersionRequested = MAKEWORD(2, 2);
WSAStartup(wVersionRequested, &wsaData);

//创建客户端套接字, 使用TCP协议
SOCKET sockClient = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
if (sockClient == INVALID_SOCKET)
{
    cout << "套接字创建失败" << WSAGetLastError() << endl;
    WSACleanup();
    return 1;
}

```

向服务端发出建连请求

```

//addrSrv
SOCKADDR_IN addrSrv;
//设置地址族为IPv4
addrSrv.sin_family = AF_INET;
//设置地址的端口号信息
addrSrv.sin_port = htons(8888);
//127.0.0.1一个特殊的IP地址, 表示是本机的IP地址
addrSrv.sin_addr.S_un.S_addr = inet_addr("127.0.0.1");
//向一个服务端的socket发出建连请求
int conn = connect(sockClient, (SOCKADDR*)&addrSrv, sizeof(SOCKADDR));
if (conn == SOCKET_ERROR) {
    closesocket(sockClient);
    cout << "连接错误: " << WSAGetLastError() << endl;
    WSACleanup();
    return 1;
}
else {
    cout << "连接成功" << endl;
}

```

发送消息

```

int send_result = send(sockClient, "hello", (int)strlen("hello"), 0);
if (send_result == SOCKET_ERROR) {
    cout << "发送失败: " << WSAGetLastError() << endl;
    closesocket(sockClient);
    WSACleanup();
    return 1;
}

cout << "发送消息: " << send_result << endl;

```

接收消息

```

char recvbuf[512];
int recv_result;
do {
    recv_result = recv(sockClient, recvbuf, 512, 0);
    if (recv_result > 0)
        cout << "收到信息: " << recv_result << endl;
    else if (recv_result == 0)
        cout << "连接关闭" << endl;
    else
        cout << "收到失败: " << WSAGetLastError() << endl;
} while (recv_result > 0);

```

服务端代码

创建服务端套接字

```

//初始化Socket DLL, 使用2.2版本的socket
WORD wVersionRequested;
WSADATA wsaData;
/* Use the MAKEWORD(lowbyte, highbyte) macro declared in Windef.h */
wVersionRequested = MAKEWORD(2, 2);
WSAStartup(wVersionRequested, &wsaData);

SOCKET sockSrv = socket(AF_INET, SOCK_STREAM, 0);
if (sockSrv == INVALID_SOCKET)
{
    cout << "套接字创建失败" << WSAGetLastError() << endl;
    WSACleanup();
    return 1;
}

```

绑定

```
//addrSrv
SOCKADDR_IN addrSrv;
//设置地址族为IPv4
addrSrv.sin_family = AF_INET;
//设置地址的端口号信息
addrSrv.sin_port = htons(8888);
addrSrv.sin_addr.S_un.S_addr = INADDR_ANY;

int bind_reult = bind(sockSrv, (SOCKADDR*)&addrSrv, sizeof(SOCKADDR));
if (bind_reult == SOCKET_ERROR) {
    cout << "绑定失败: " << WSAGetLastError() << endl;
    closesocket(sockSrv);
    return 1;
}
```

开始监听

```
//开始监听
int lis = listen(sockSrv, 100);
if (lis == SOCKET_ERROR) {
    cout << "出现错误: " << WSAGetLastError() << endl;
    closesocket(sockSrv);
    WSACleanup();
    return 1;
}
```

连接客户端

```
//等待连接
SOCKADDR_IN addrClient;
int len = sizeof(addrClient);
SOCKET sockConn = accept(sockSrv, (SOCKADDR*)&addrClient, &len);
if (sockConn == INVALID_SOCKET)
{
    cout << "客户端发出请求, 服务器接收请求失败: " << WSAGetLastError() << endl;
    closesocket(sockConn);
    WSACleanup();
    return 1;
}
else {
    cout << "服务器接收请求" << endl;
}
```

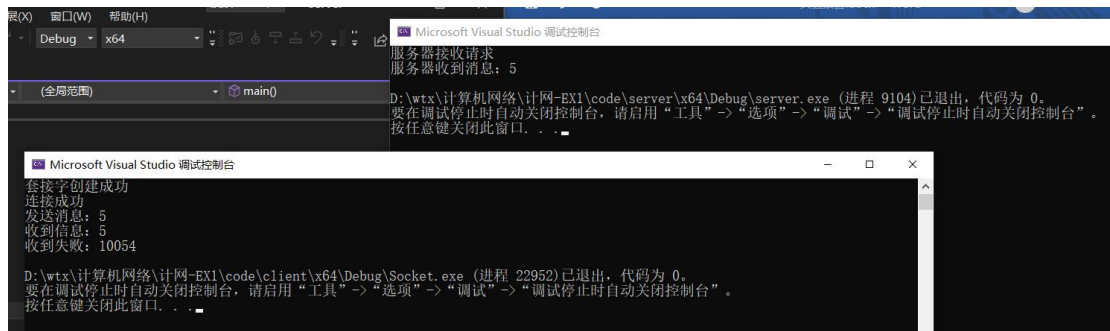
接收消息

```
char recvBuf[512];
int relen = recv(sockConn, recvBuf, 512, 0);
if (relen == SOCKET_ERROR) {
    cout << "服务器没有收到消息: " << WSAGetLastError() << endl;
    closesocket(sockSrv);
    WSACleanup();
    return 1;
}
else {
    cout << "服务器收到消息: " << relen << endl;
}
```

发送消息

```
int slen = send(sockConn, recvBuf, relen, 0);
if (slen == SOCKET_ERROR) {
    cout << "服务器发送消息失败: " << WSAGetLastError() << endl;
    closesocket(sockSrv);
    WSACleanup();
    return 1;
}
```

运行结果



此时服务端和客户端可以进行连接,可以互发消息,收到的消息和发送的消息是一致的,但是客户端发送的消息并不是所希望的“Hello“,而且还未实现多轮对话的功能

完善功能

消息发送正确

```
char recvBuf[512] = { 0 };
int relen = recv(sockConn, recvBuf, 512, 0);
if (relen == SOCKET_ERROR) {
    cout << "服务器没有收到消息:" << WSAGetLastError() << endl;
    closesocket(sockSrv);
    WSACleanup();
    return 1;
}
else {
    cout << "服务器收到消息:" << recvBuf << endl;
}

int slen = send(sockConn, recvBuf, relen, 0);
if (slen == SOCKET_ERROR) {
    cout << "服务器发送消息失败:" << WSAGetLastError() << endl;
    closesocket(sockSrv);
    WSACleanup();
    return 1;
}
else {
    cout << "服务器发送消息:" << recvBuf << endl;
}
```

修改部分

服务端: recvBuf 初始化; 接收消息和发送消息成功时输出 recvBuf

```

char mes[1024] = { "Hello" };
int send_result = send(sockClient, mes, (int)strlen(mes), 0);
if (send_result == SOCKET_ERROR) {
    cout << "发送失败: " << WSAGetLastError() << endl;
    closesocket(sockClient);
    WSACleanup();
    return 1;
}
cout << "发送消息: " << mes << endl;

```

```

char recvbuf[512] = { 0 };
int recv_result;
do {
    recv_result = recv(sockClient, recvbuf, 512, 0);
    if (recv_result > 0)
        cout << "收到信息: " << recvbuf << endl;
    else if (recv_result == 0)
        cout << "连接关闭" << endl;
    else
        cout << "收到失败: " << WSAGetLastError() << endl;
} while (recv_result > 0);

```

客户端：增加 mes 表示发送消息；recvbuf 初始化；接收消息和发送消息成功时输出 recvBuf

Microsoft Visual Studio 调试控制台

```

服务器接收请求
服务器收到消息: Hello
服务器发送消息: Hello

D:\wtx\计算机网络\计网-EX1\code\server\x64\Debug\server.exe (进程 25132)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. . .

```

Microsoft Visual Studio 调试控制台

```

套接字创建成功
连接成功
发送消息: Hello
收到信息: Hello
收到失败: 10054

D:\wtx\计算机网络\计网-EX1\code\client\x64\Debug\Socket.exe (进程 1884)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. . .

```

运行结果达到预期目的

多轮对话

```
while (1) {
    while (1) { //从客户端1接收消息, 发送给客户端2
        *recvBuf = '/0';
        recv1_result = recv(sockConn1, recvBuf, 1024, 0);
        if (recv1_result == SOCKET_ERROR) {
            cout << "服务器没有收到客户端1的消息: " << WSAGetLastError() << endl;
            closesocket(sockConn1);
            WSACleanup();
            return 1;
        }
        else {
            cout << "服务器收到客户端1的消息: " << recvBuf << endl;
        }
        send2_result = send(sockConn2, recvBuf, 1024, 0);
        if (send2_result == SOCKET_ERROR) {
            cout << "服务器向客户端2发送消息失败: " << WSAGetLastError() << endl;
            closesocket(sockConn2);
            WSACleanup();
            return 1;
        }
        else {
            cout << "服务器向客户端2发送消息: " << recvBuf << endl;
        }
        if (strcmp(recvBuf, "exit") == 0) {
            closesocket(sockConn1);
            closesocket(sockConn2);
            WSACleanup();
            return 1;
        }
        if (strcmp(recvBuf, "over") == 0) {
            break;
        }
    }
    while (1) { //从客户端2接收消息, 发送给客户端1
```

服务器端接收消息信息如图

```
/*当客户端发送over时, 结束发送消息; 发送exit时, 结束对话*/
while (1) {
    while (1) { //客户端1向2发消息
        cout << "client1: ";
        *mes = '/0';
        cin.getline(mes, sizeof(mes));
        send_result = send(sockClient1, mes, (int)strlen(mes), 0);
        if (send_result == SOCKET_ERROR) {
            cout << "发送消息失败: " << WSAGetLastError() << endl;
            closesocket(sockClient1);
            WSACleanup();
            return 1;
        }
        //cout << "发送消息: " << mes << endl;
        if (strcmp(mes, "exit") == 0) {
            {
                cout << "结束对话" << endl;
                closesocket(sockClient1);
                WSACleanup();
                return 1;
            }
        }
        else if (strcmp(mes, "over") == 0) {
            break;
        }
    }

    while (1) { //客户端1接收2的消息
        *recvbuf = '/0';
        recv_result = recv(sockClient1, recvbuf, 1024, 0);
        if (recv_result > 0)
            cout << "收到信息: " << recvbuf << endl;
        else if (recv_result == 0)
            cout << "连接关闭" << endl;
        else {
            cout << "收到失败: " << WSAGetLastError() << endl;
            closesocket(sockClient1);
            WSACleanup();
            return 1;
        }
        if (strcmp(recvbuf, "exit") == 0) {
            cout << "对方结束对话" << endl;
            closesocket(sockClient1);
            WSACleanup();
            return 1;
        }
        else if (strcmp(recvbuf, "over") == 0) {
            break;
        }
    }
    cout << "client2: " << recvbuf << endl;
}
```

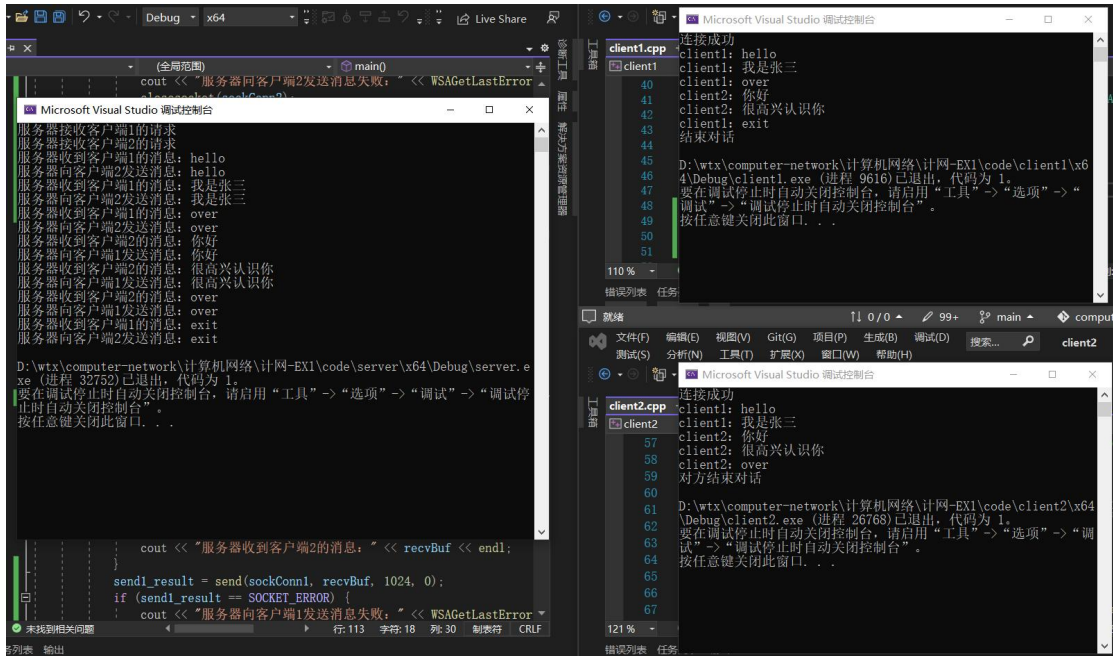
客户端 1 收发消息代码如图，此时默认客户端 1 先发消息，客户端 2 后发消息，客户端 2 相关代码即将 client1 和 client2 关键词调换。并且将接收、发送代码顺序调换。

运行后发现遇到问题：客户端 1 输出 over 后客户端 2 不能进行输出

原因：char 型数组内容清空时代码

```
*recvBuf = '\0';
```

使用 memset 函数更改后成功，此时运行结果如下图



增加功能：群聊

按照连接服务端的顺序进行标记 id，根据 id 以自小到的顺序依次发言。

服务端代码修改

```
//实现群聊
//连接客户端（最高100个）
int amount = 0;
cin >> amount;
SOCKADDR_IN addrClient[100];
SOCKET sockConn[100];

for (int i = 1; i <= amount; i++) {
    int len = sizeof(addrClient[i]);
    sockConn[i] = accept(sockSrv, (SOCKADDR*)&addrClient[i], &len);
    if (sockConn[i] == INVALID_SOCKET)
    {
        cout << "客户端" << i << "发出请求，服务器接收请求失败：" << WSAGetLastError() << endl;
        closesocket(sockConn[i]);
        WSACleanup();
        return 1;
    }
    else {
        cout << "服务器接收客户端" << i << "的请求" << endl;
    }
}
```

以数组形式存储与客户端连接产生的套接字


```

//向所有客户端传达有多少人参与群聊以及各自的序号
for (int i = 1; i <= amount; i++) {
    char buf[2];
    buf[0] = (char)i;
    buf[1] = (char)amount;
    int send_r = send(sockConn[i], buf, 1024, 0);
    if (send_r == SOCKET_ERROR) {
        cout << "服务器向客户端" << i << "发送消息失败: " << WSAGetLastError() << endl;
        closesocket(sockConn[i]);
        WSACleanup();
        return 1;
    }
}

```

序号依据连接服务端顺序产生

```

//处理消息
char recvBuf[1024] = { 0 };
int recv_result = 0;
int send_result = 0;
int i = 1;
while (1) {
    memset(recvBuf, '\0', sizeof(recvBuf));
    recv_result = recv(sockConn[i], recvBuf, 1024, 0);
    if (recv_result == SOCKET_ERROR) {
        cout << "服务器没有收到客户端" << i << "的消息: " << WSAGetLastError() << endl;
        closesocket(sockConn[i]);
        WSACleanup();
        return 1;
    }
    else {
        cout << "服务器收到客户端" << i << "的消息: " << recvBuf << endl;
    }
    for (int j = 1; j <= amount; j++) {
        if (j != i) {
            send_result = send(sockConn[j], recvBuf, 1024, 0);
            if (send_result == SOCKET_ERROR) {
                cout << "服务器向客户端" << j << "发送消息失败: " << WSAGetLastError() << endl;
                closesocket(sockConn[j]);
                WSACleanup();
                return 1;
            }
            else {
                cout << "服务器向客户端" << j << "发送消息: " << recvBuf << endl;
            }
        }
    }
    if (strcmp(recvBuf, "over") == 0) {
        if (i != amount) {
            i++;
        }
        else {
            i = 1;
        }
    }
    if (strcmp(recvBuf, "exit") == 0) {
        for (int j = 1; j <= amount; j++) {
            closesocket(sockConn[j]);
        }
        WSACleanup();
        return 1;
    }
}

```

不再有从具体某个客户端接收消息以及向具体某个客户端发送消息。服务端从任意一个客户端接收到消息，转发给其余所有客户端。

客户端代码修改

客户端实际上不区分客户端 1、2、3，即可以 client3 程序先连接服务端，被认作是客户端 1。

```

recv_result = recv(sockClient, buf, 1024, 0);
int id = (int)buf[0];
int amount = (int)buf[1];
cout << id << endl;
cout << amount << endl;

```

连接后由服务端对于客户端进行 id 赋值，并告知连接的总客户端数

```

while (1) { //客户端发消息
    cout << "client1: ";
    memset(mes, '\0', sizeof(mes));
    cin.getline(mes, sizeof(mes));
    send_result = send(sockClient, mes, (int)strlen(mes), 0);
    if (send_result == SOCKET_ERROR) {
        cout << "发送消息失败: " << WSAGetLastError() << endl;
        closesocket(sockClient);
        WSACleanup();
        return 1;
    }

    //cout << "发送消息: " << mes << endl;
    if (strcmp(mes, "exit") == 0) {
        cout << "结束群聊" << endl;
        closesocket(sockClient);
        WSACleanup();
        return 1;
    }

    else if (strcmp(mes, "over") == 0) {
        i++;
        break;
    }
}

while (1) { //客户端接收消息
    memset(recvbuf, '\0', sizeof(recvbuf));
    recv_result = recv(sockClient, recvbuf, 1024, 0);
    if (recv_result > 0) {
        cout << "client" << i << ": " << recvbuf << endl;
    }

    //cout << "收到信息: " << recvbuf << endl;
    else if (recv_result == 0) {
        cout << "连接关闭" << endl;
    }

    else {
        cout << "收到失败: " << WSAGetLastError() << endl;
        closesocket(sockClient);
        WSACleanup();
        return 1;
    }

    if (strcmp(recvbuf, "exit") == 0) {
        cout << "客户端" << i << "结束群聊" << endl;
        closesocket(sockClient);
        WSACleanup();
        return 1;
    }

    else if (strcmp(recvbuf, "over") == 0) {
        if (i != amount) {
            i++;
        }
        else {
            i = 1;
            break;
        }
    }
}
}

```

客户端 1（即最先连接的客户端）先发送消息，再接收消息。所以当它发送“over”消息后，跳出发送消息的循环，进入到接收消息的循环中。每接收到一次“over”，发送消息的客户端 id 为 i+1，直到发送消息的客户端 id 为 amount 时，下一次发送消息的客户端为客户端 1，进入发送消息的循环。

```

while (1) { //客户端先接收消息
    memset(recvbuf, '\0', sizeof(recvbuf));
    recv_result = recv(sockClient, recvbuf, 1024, 0);
    if (recv_result > 0) {
        cout << "client" << i << ": " << recvbuf << endl;
    }
    //cout << "收到信息: " << recvbuf << endl;
    else if (recv_result == 0)
        cout << "连接关闭" << endl;
    else {
        cout << "收到失败: " << WSAGetLastError() << endl;
        closesocket(sockClient);
        WSACleanup();
        return 1;
    }
    if (strcmp(recvbuf, "exit") == 0) {
        cout << "客户端" << i << "结束群聊" << endl;
        closesocket(sockClient);
        WSACleanup();
        return 1;
    }
    else if (strcmp(recvbuf, "over") == 0) {
        if (i == id - 1) {
            i = id;
            break;
        }
        else {
            if (i == amount) {
                i = 1;
            }
            else {
                i++;
            }
        }
    }
}

while (1) { //客户端再发送消息
    cout << "client" << id << ": ";
    memset(mes, '\0', sizeof(mes));
    cin.getline(mes, sizeof(mes));
    send_result = send(sockClient, mes, (int)strlen(mes), 0);
    if (send_result == SOCKET_ERROR) {
        cout << "发送消息失败: " << WSAGetLastError() << endl;
        closesocket(sockClient);
        WSACleanup();
        return 1;
    }
    //cout << "发送消息: " << mes << endl;
    if (strcmp(mes, "exit") == 0)
    {
        cout << "结束群聊" << endl;
        closesocket(sockClient);
        WSACleanup();
        return 1;
    }
    else if (strcmp(mes, "over") == 0) {
        if (i == amount) {
            i = 1;
        }
        else {
            i++;
        }
        break;
    }
}
}

```

其余客户端，先接收消息，轮到自己时发送消息。所以当它收到“over”消息时，需要进行判断，如果下一个发消息的客户端是自己，那么跳出接收消息的循环，进入到发送消息的循环。

带时间发送消息

客户端发送信息代码

```
memset(mes, '\0', sizeof(mes));
time_t t;
char buf[128];
memset(buf, 0, sizeof(buf));
struct tm* tmp;
t = time(NULL);
tmp = localtime(&t);
strftime(buf, sizeof(buf), "%Y-%m-%d %H:%M:%S", tmp);
cin.getline(mes, sizeof(mes));
strcat(buf, " ");
strcat(buf, mes);
send_result = send(sockClient, buf, (int)strlen(buf), 0);
```

发送到服务端的消息格式为：时间消息+ “ ” +发送消息内容

客户端接收信息代码

```
memset(recvbuf, '\0', sizeof(recvbuf));
memset(solve, '\0', sizeof(solve));
recv_result = recv(sockClient, recvbuf, 1024, 0);
string buf_dosoming(recvbuf);
strcpy(solve, (buf_dosoming.substr(buf_dosoming.length() - 4, buf_dosoming.length() - 1).c_str()));
```

处理得到接收消息的最后四个字符组成的字符串 solve，用 solve 判断是否到下一个客户端进行发送或者是否结束群聊。

服务端接收消息代码

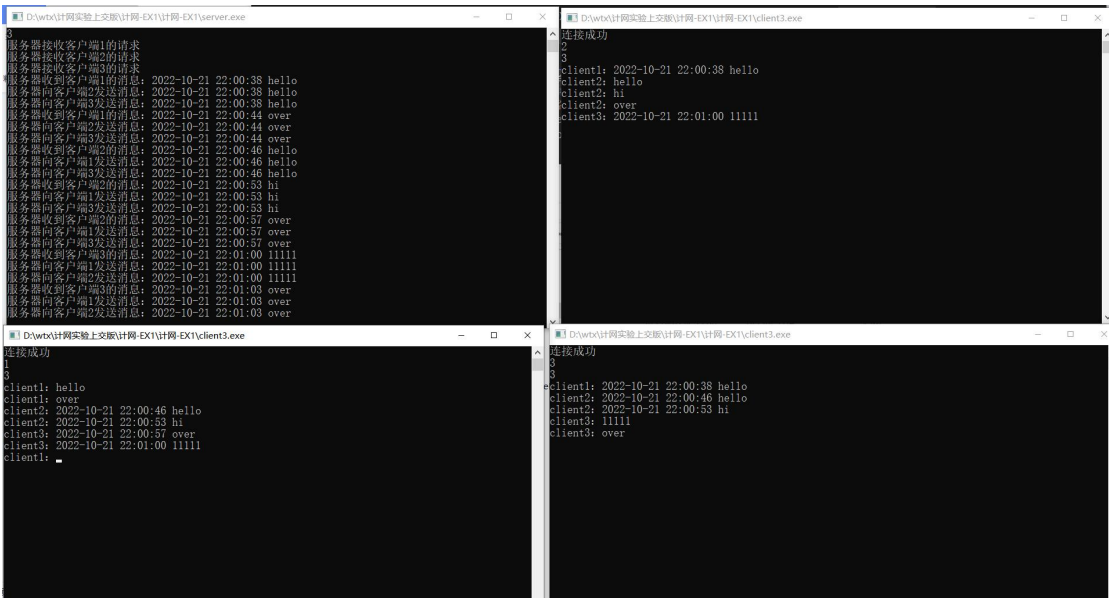
```
else {
    cout << "服务器收到客户端" << i << "的消息: " << recvBuf << endl;
    string buf_dosoming(recvBuf);
    strcpy(t, (buf_dosoming.substr(buf_dosoming.length() - 4, buf_dosoming.length() - 1).c_str()));
}
```

处理得到接收消息的最后四个字符组成的字符串 solve，用 solve 判断是否到下一个客户端进行发送或者是否关闭套接字。

运行运行与结果

首先运行 server.exe，输入需要连接的客户端数 amount。

接着运行 amount 个 client.exe，按照连接顺序按规则进行收发消息。



```
D:\web\计网实验上机\计网-EX1\计网-EX1\server.exe
3
服务器接收客户端1的请求
服务器接收客户端2的请求
服务器接收客户端3的请求
服务器收到客户端1的消息: 2022-10-21 22:00:38 hello
服务器向客户端2发送消息: 2022-10-21 22:00:38 hello
服务器向客户端3发送消息: 2022-10-21 22:00:38 hello
服务器收到客户端1的消息: 2022-10-21 22:00:44 over
服务器向客户端2发送消息: 2022-10-21 22:00:44 over
服务器向客户端3发送消息: 2022-10-21 22:00:44 over
服务器收到客户端2的消息: 2022-10-21 22:00:46 hello
服务器向客户端1发送消息: 2022-10-21 22:00:46 hello
服务器向客户端3发送消息: 2022-10-21 22:00:46 hello
服务器收到客户端2的消息: 2022-10-21 22:00:53 hi
服务器向客户端1发送消息: 2022-10-21 22:00:53 hi
服务器向客户端3发送消息: 2022-10-21 22:00:53 hi
服务器收到客户端3发送消息: 2022-10-21 22:00:57 over
服务器向客户端1发送消息: 2022-10-21 22:00:57 over
服务器向客户端3发送消息: 2022-10-21 22:00:57 over
服务器收到客户端3的消息: 2022-10-21 22:01:00 11111
服务器向客户端1发送消息: 2022-10-21 22:01:00 11111
服务器向客户端2发送消息: 2022-10-21 22:01:00 11111
服务器收到客户端3的消息: 2022-10-21 22:01:03 over
服务器向客户端1发送消息: 2022-10-21 22:01:03 over
服务器向客户端2发送消息: 2022-10-21 22:01:03 over

D:\web\计网实验上机\计网-EX1\计网-EX1\client3.exe
连接成功
3
client1: 2022-10-21 22:00:38 hello
client2: hello
client2: hi
client2: over
client3: 2022-10-21 22:01:00 11111

D:\web\计网实验上机\计网-EX1\计网-EX1\client3.exe
连接成功
3
client1: hello
client2: over
client2: 2022-10-21 22:00:46 hello
client2: 2022-10-21 22:00:53 hi
client3: 2022-10-21 22:00:57 over
client3: 2022-10-21 22:01:00 11111
client1: _

D:\web\计网实验上机\计网-EX1\计网-EX1\client3.exe
连接成功
3
client1: 2022-10-21 22:00:38 hello
client2: 2022-10-21 22:00:46 hello
client2: 2022-10-21 22:00:53 hi
client3: 11111
client3: over
```