实验1：利用Socket设计和编写一个聊天程序

📌 姓名：陈睿颖

学号：2013544

专业：计算机科学与技术

1. 实验内容

a. 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。

b. 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。

c. 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。

d. 对实现的程序进行测试。

2. 协议设计

2.1 消息类型

在协议设计中，交互信息为1000个字节以内的字符串信息，在服务器端和客户端分别有相应的char型数组进行存放。根据输入内容，将其作为字符串类型进行接收，并通过后续模块进行发送与接

收。

2.2 消息语法

`char server_buff[1000]`

`char client_buff[1000]`

控制信息结构：通过识别输入信息进行控制的转换。

2.3 消息语义

本程序中的消息分为非空消息和空消息：

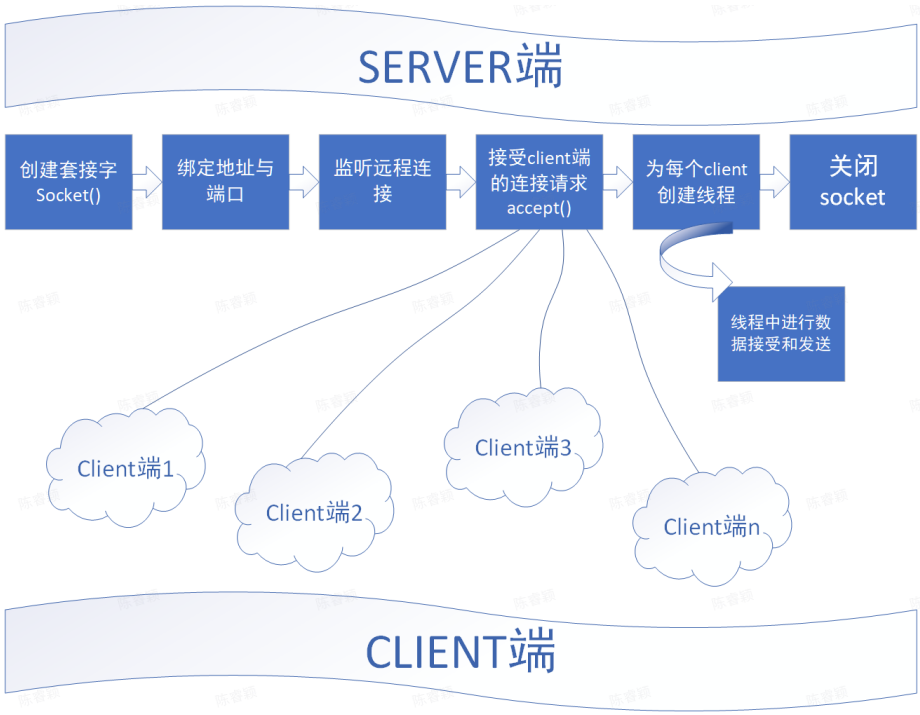
- 非空消息：聊天内容；
- 空消息：在客户的发送数据的线程中设置了一个int变量 `flag` ，如果用户输入 `q` 则将 `flag` 变量置为0，客户端不进行数据的发送并且退出程序，同时服务端显示客户端退出；

2.4 消息时序

1. 打开服务器端exe，绑定端口和IP地址。等待客户端连接；
2. 打开客户端exe，默认绑定和服务器相同的端口和IP地址，进行与服务器的连接。若连接成功则进行聊天，若失败则打印报错信息；
3. 与服务器连接成功的n个客户端收到服务端的欢迎信息并开始进行多线程聊天，聊天内容打印在服务端上；
4. 当有客户端退出聊天程序时，服务器端显示离开用户的详情；

2.5 设计示意图

如图所示：



3. 各模块功能

3.1 server端

3.1.1 获取当前时间的函数

```
string TimeNow() {
        time_t now = time(NULL);
        tm* tm_t = localtime(&now);
        std::stringstream ss; //将时间转化为字符串放进ss中
        ss << "[";
        if (tm_t->tm_hour < 10)
                ss << "0";
        ss << tm_t->tm_hour << ":";
        if (tm_t->tm_min < 10)
                ss << "0";
        ss << tm_t->tm_min << ":";
        if (tm_t->tm_sec < 10)
                ss << "0";
        ss << tm_t->tm_sec << "]";
        string str;
        ss >> str; //将stringstream放进str中
        //char* p = const_cast<char*>(str.c_str()); //将str转化为char*类型
        return str;
}
```

上述代码中，将获取到的时间信息按照“hh:mm:ss”的格式放进了`stringstream ss`中，再转化为字符串类型并返回，方便后续打印日志时显示当前时间。

3.1.2 线程函数

该部分函数将是主函数中创建线程时所要执行的函数，其中传来的参数`lpThreadParameter`是套接字`client`。当连接成功时，向客户端发生连接成功的字符信息`buf`，后续循环接受客户端发来的字符消息并打印在服务器端的屏幕上。同时，还将打印某客户端离开聊天时的信息。

```
DWORD WINAPI ThreadFun(LPVOID lpThreadParameter) {
        // 与客户端通讯，发送或者接受数据
        SOCKET c = (SOCKET)lpThreadParameter;//接受传来的线程参数，实际是主函数中的client这个socket
        // 发送与客户端连接成功的信息
        char buf[1000] = { 0 };
        sprintf_s(buf, "连接成功! 欢迎用户 %d 进入聊天室! （退出聊天请按q）", c);
        send(c, buf, 1000, 0);

        // 循环接收客户端数据
        int ret = 0;
        do {
                char server_buf[1000] = { 0 };
                ret = recv(c, server_buf, 1000, 0);//接受客户端发来的字符消息

                cout << "================================================" << endl;
                cout << TimeNow();
                cout << "用户" << c << ":    " << server_buf << endl;//打印消息内容

        } while (ret != SOCKET_ERROR && ret != 0);
        cout << "================================================" << endl;
        cout << endl;
        cout << endl;
        cout << endl;
        cout << TimeNow();
        cout << "用户" << c << "离开了聊天室! " << endl;;

        return 0;
}
```

3.1.3 进行一系列初始化操作

i. 初始化Socket DLL

```
WSADATA wsaData;//初始化Socket DLL，协商使用的Socket版本
        if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0){ //MAKEWORD(2, 2) 生成 WSAStartup 期望的版本控制字。
                cout << TimeNow();
                cout << "WSAStartup Error:" << WSAGetLastError() << endl;
                return 0;
        }
        else{
                cout << TimeNow();
                cout << "初始化SocketDLL成功!" << endl;
        }
```

ii. 创建流式套接字socket

```
// 创建流式套接字
        cout << TimeNow();
        cout << "正在创建流式套接字......" << endl;
        SOCKET s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);//指定创建的socket的地址类型、服务类型及TCP协议

        if (s == INVALID_SOCKET){
                cout << TimeNow();
                cout << "socket错误:" << WSAGetLastError() << endl;
                return 0;
        }
        else {
                cout << TimeNow();
                cout << "创建成功" << endl;
        }
```

iii. 绑定端口和IP地址

```
// 绑定端口和ip
        cout << TimeNow();
        cout << "正在绑定端口和IP" << endl;
        sockaddr_in addr;
        memset(&addr, 0, sizeof(sockaddr_in));
        addr.sin_family = AF_INET;//地址类型
```

```
        addr.sin_port = htons(8000);//端口号
        addr.sin_addr.s_addr = inet_addr("127.0.0.1");//IP地址

        int len = sizeof(sockaddr_in);
        if (bind(s, (SOCKADDR*)&addr, len) == SOCKET_ERROR){//将本地地址绑定到s这个socket
                cout << TimeNow();
                cout << "绑定错误:" << WSAGetLastError() << endl;
                return 0;
        }
        else{
                cout << TimeNow();
                cout << "端口号: 8000 IP: 127.0.0.1绑定成功! " << endl;

        }

        // 监听
        listen(s, 5);
        cout << TimeNow();
        cout << "等待用户进入聊天室......" << endl;
```

3.1.4 循环接受客户端的连接

在该部分代码中，主线程循环接受客户端的建连请求。并在成功连接后创建线程启动线程函数，则客户端可以开始在聊天室内发送消息。

```
// 主线程循环接收客户端的连接
        while (true){
                sockaddr_in addrClient; //远程端的地址
                len = sizeof(sockaddr_in);
                // 接受成功返回与client通讯的Socket
                SOCKET client = accept(s, (SOCKADDR*)&addrClient, &len);//接受server请求等待队列中的连接请求即client
                if (client != INVALID_SOCKET){
                        cout << TimeNow();
                        cout << "用户" << client << "连接成功! " << endl;
                        // 创建线程，并且传入与client通讯的套接字
                        HANDLE hThread = CreateThread(NULL, 0, ThreadFun, (LPVOID)client, 0, NULL);
                        if(!hThread)
                                CloseHandle(hThread); // 关闭对线程的引用
                }

```

```
16          }
```

iv. 最后通过 `closesocket(s)` 和 `WSACleanup()` 关闭监听套接字并清理winsock2的环境。

3.2 client端

3.2.1 获取当前时间函数。代码同server端

3.2.2 初始化操作，与server端类似

i. 加载winsock2的环境

```
1 WSADATA  wd;
2 if (WSAStartup(MAKEWORD(2, 2), &wd) != 0){
3         cout << TimeNow();
4         cout << "WSAStartup错误: " << GetLastError() << endl;
5         return 0;
6 }
```

ii. 创建流式套接字

```
1 SOCKET  s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
2 if (s == INVALID_SOCKET){
3         cout << TimeNow();
4         cout << "socket错误: " << GetLastError() << endl;
5         return 0;
6 }
```

iii. 连接服务器

```
1 sockaddr_in   addr;
2 addr.sin_family = AF_INET;
3 addr.sin_port = htons(8000);
4 addr.sin_addr.s_addr = inet_addr("127.0.0.1");
5 int len = sizeof(sockaddr_in);
6 if (connect(s, (SOCKADDR*)&addr, len) == SOCKET_ERROR){//向服务端发送建连请
  求
7         cout << TimeNow();
```

```
        cout << "连接错误: " << GetLastError() << endl;
        return 0;
}
```

3.2.3 接受服务器的消息

在server端代码中提到，连接成功会给对应的客户端发送连接成功的信息，在这里客户端进行接受并打印在客户端屏幕上。

```
//接收服务端的消息
        char buf[1000] = { 0 };
        recv(s, buf, 1000, 0);
        cout << buf << endl;

```

3.2.4 循环输入内容

在该部分进行聊天消息的输入；并规定输入 `q` 时退出聊天程序。

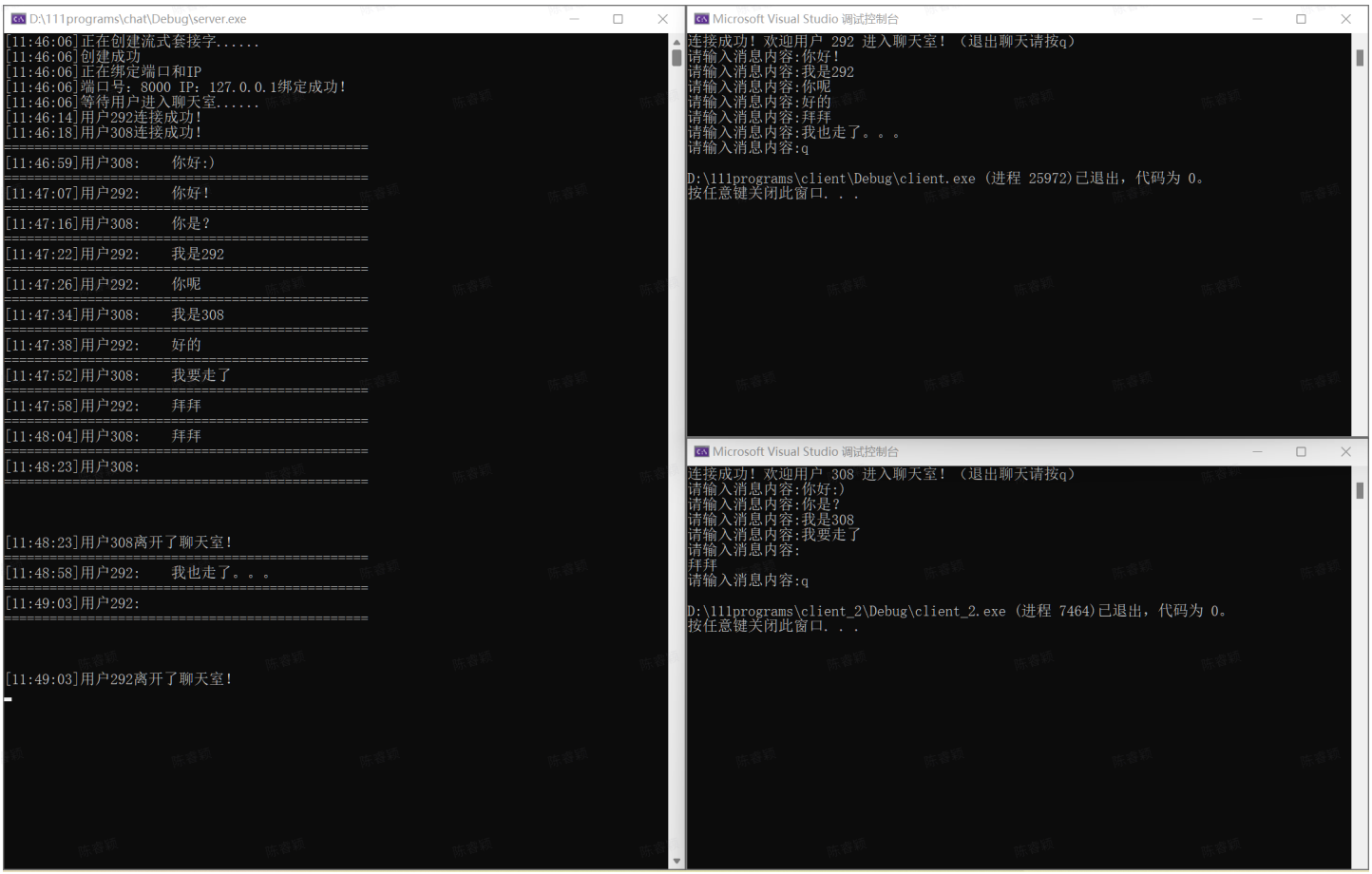
```
        //随时给服务端发消息
        int  ret = 0;
        int flag = 1; //是否退出的标记
        do{
                char client_buf[1000] = { 0 };
                cout << "请输入消息内容:";
                cin.getline(client_buf,1000);
                if (client_buf[0] == 'q') {
                        flag = 0; //如果输入q则跳出循环
                }
                if(flag==1)
                ret = send(s, client_buf, 1000, 0);
        } while (ret != SOCKET_ERROR && ret != 0 && flag==1);
```

3.2.5 关闭socket

最后通过 `closesocket(s)` 和 `WSACleanup()` 关闭监听套接字并清理winsock2的环境。

4. 程序界面展示及运行说明

如下图所示，此为多人聊天室中有两名用户进入聊天室聊天的情形。可以看到两名用户在命令行中输入聊天内容，其显示在服务器端的命令行中，并标明了消息的发送者和时间。用户输入q（quit）即可退出聊天程序。每一位用户离开时都会在服务器端的聊天界面上显示。



5. 实验过程中遇到的问题分析

a. 在实现显示时间的这个功能上，一开始我使用的方法是在客户端输入消息时将时间内容拼接进入消息并发送给服务器端。但测试后发现当输入的文字过长时会出现溢出的情况，具体原因可能是发送为char[]型字符，而将时间内容string进行转化后会使得数组空间存在问题，导致程序中断等。经修改，将显示时间的部分改为在server端即可正常显示。