Lab1——利用Socket设计和编写一个聊天程序

学号：2013921
姓名：周延霖
专业：信息安全

一、作业说明

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学院网站

二、协议设计

在本次实验中，传输层协议本人采用TCP协议，数据以流式传输。

> 应用层协议设计

(1) 语法

- 本次写的程序可以实现双人聊天，客户端和服务端都可以发送和接收消息
- 发送和接收消息的前提是双方要建立连接
- 客户端可以向服务器发送消息并接受服务器传来的消息，同样服务器也可以向客户端发送消息并接受客户端传输的消息
- 每条消息具体报文长度不能超过100个字符，并以空格或换行符表示结束，如过超过此长度，超过的部分将被自动忽略
- 如果一次输入中包含空格，将根据空格拆分成多个字符串分别发送

(2) 语义

关于消息发送和接收的显示

当成功接收到消息时，将在接收到的消息前打上时间戳后显示如下：

> “那个人在 X 号 X 点 X 分 X 秒发过来一条消息:\n 消息内容”

当成功发送消息时，将显示如下信息：

> “你在 X 号 X 点 X 分 X 秒发给了那个人一条消息\n”

如果发送消息“bye”，则证明用户即将结束聊天，则发送这条消息的端口则会显示如下信息：

> “你在 X 号 X 点 X 分 X 秒离开了与那个人聊天”

另外一方在收到“bye”后会显示如下信息：

> “那个人在 X 号 X 点 X 分 X 秒离开了与你的聊天”

至此聊天结束.

(3) 时序

本次实验采用多线程的方式设计，目的是为了将接收消息和发送消息分开执行，这样子不用在每一次发送完消息之后还要等待对方的消息到来后再发送，双方都可以随时发送多条消息并接收多条消息。

接收到的消息按照发送和接收的时间顺序来打印。

三、程序设计

(1)环境配置

本人是Mac电脑，无法调用Windows的接口，用的是室友的笔记本，环境如下：

- 操作系统：Windows 11
- 编译器：Visual Studio 2019
- 语言：C++
- 链接库：ws2_32.lib

(2)设计思路

> 服务端

在服务器server.exe程序开启之后，首先判断网络环境以及监听的接口是否有异常，如果一切正常则会显示正常信息并开始监听，如果出现异常则会释放掉设备后返回异常。

在监听的过程中如果对方上线，则会建立相应的连接，并开始进行消息的收发，直到输入或收到bye退出进程。

> 客户端

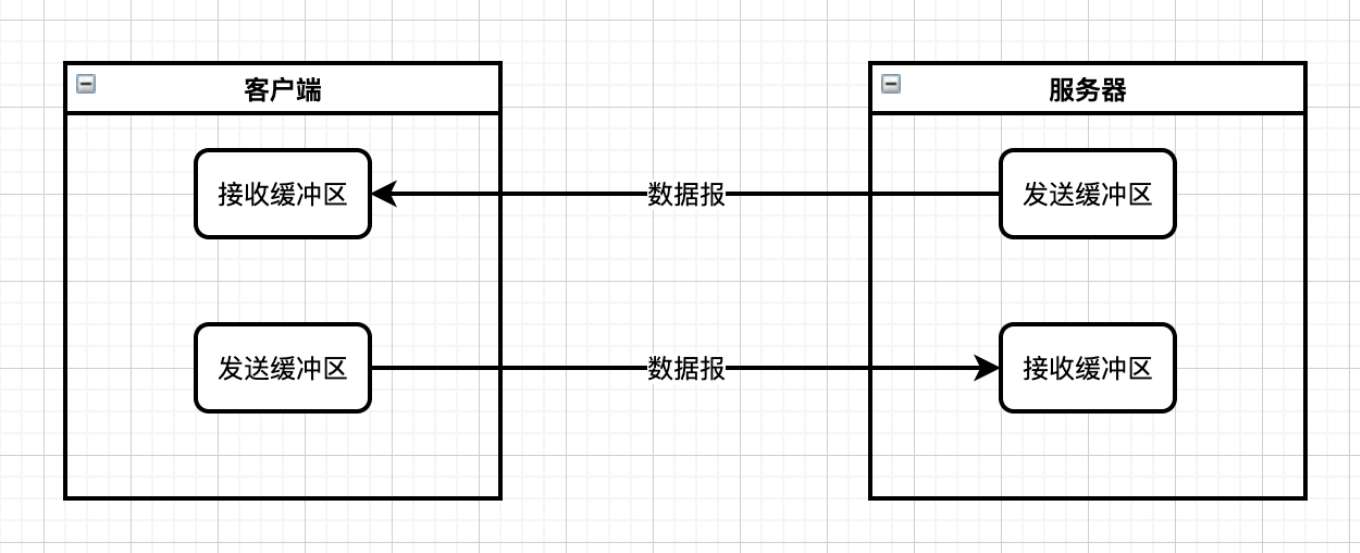
在进程client.exe开启之后，会对服务器进行尝试连接，如果此时还没有相应的服务器，则会显示信息并退出程序，如果此时连接上服务器，则会开始消息的收发，直到输入或收到bye退出进程。

> 多线程

如果光对网络进行连接，然后在主进程中用while循环来收发消息的话，则只能实现我发一句你发一句这样不够人性化的聊天方式。

在现实中，往往都是一个人可以发出多条信息，或者在收到多条信息后再进行相应的回复，为了可以实现这个更加人性化的功能，需要创建多个线程来分别实现消息的接收和发送，两个功能之间可以共享资源，但不会相互影响。

大致的流程图如下所示：



四、代码实现

(1)服务端

首先引入头文件，并定义一个宏BUF_SIZE,这样方便以后更改缓冲区的长度。

```
#include <stdio.h>
#include <WinSock2.h>
#include <windows.h>
#include <iostream>
#include <thread>
#include <string>
#pragma warning(disable : 4996)
#pragma comment(lib, "ws2_32.lib")  //加载 ws2_32.dll

#define BUF_SIZE 100
```

然后对网络库进行初始化，并在成功调用网络库后显示成功唤醒网络,在调用失败后显示出现了一点问题导致唤醒网络失败，请重启后再试(>﹏<),并退出程序。

```
    //初始化网络库DLL
    WSADATA wsaData;
    if (WSAStartup(MAKEWORD(2, 2), &wsaData) == 0) // 调用者希望的最高版本——
2.2版和可用socket详细信息
    {
        std::cout << "成功唤醒网络" << std::endl;
    }
    else {
        std::cout << "出现了一点问题导致唤醒网络失败，请重启后再试(>﹏<)" <<
std::endl;
        return 0;
    }
```

接下来对Socket进行创建并绑定端口，开始监听客户端的请求，并在客户端上线之后显示那个人终于出现了，现在可以开始你们的聊天,进入聊天。

```
//创建套接字
SOCKET servSock = socket(AF_INET, SOCK_STREAM, 0);

//绑定套接字
sockaddr_in sockAddr;
memset(&sockAddr, 0, sizeof(sockAddr));  //每个字节都用0填充
sockAddr.sin_family = PF_INET;  //使用IPv4地址
sockAddr.sin_addr.s_addr = inet_addr("127.0.0.1");  //具体的IP地址
sockAddr.sin_port = htons(1234);  //端口
bind(servSock, (SOCKADDR*)&sockAddr, sizeof(SOCKADDR));

// 进入监听状态，监听远程连接是否到来
if (listen(servSock, 20) == 0) {
    std::cout << "正在监听中，等待是会有结果的" << std::endl;
}
else {
    std::cout << "出现一些错误导致监听失败，请重启后再试(>﹏<)" << std::endl;
    return 0;
}

// 接收客户端请求
SOCKADDR clntAddr;
int nSize = sizeof(SOCKADDR);
SOCKET clntSock = accept(servSock, (SOCKADDR*)&clntAddr, &nSize);
if (clntSock > 0) {
    std::cout << "那个人终于出现了，现在可以开始你们的聊天" << std::endl;
}
```

最后对收发消息分别创建线程，聊天开始；并在聊天结束后关闭Socket及网络库的调用。

```
// 为收发数据分别创建线程
HANDLE hThread[2];
hThread[0] = CreateThread(NULL, 0, Recv, (LPVOID)&clntSock, 0, NULL);
hThread[1] = CreateThread(NULL, 0, Send, (LPVOID)&clntSock, 0, NULL);
WaitForMultipleObjects(2, hThread, TRUE, INFINITE);
CloseHandle(hThread[0]);
CloseHandle(hThread[1]);

// 关闭套接字
closesocket(clntSock);

// 关闭套接字
closesocket(servSock);

// 终止网络库DLL的使用
WSACleanup();
```

(2)客户端

在客户端与服务端类似，首先也是引入头文件，并定义一个宏BUF_SIZE。

```
#include <stdio.h>
#include <WinSock2.h>
#include <windows.h>
#include <iostream>
#include <thread>
#include <string>
#pragma warning(disable : 4996)
#pragma comment(lib, "ws2_32.lib")  //加载 ws2_32.dll

#define BUF_SIZE 100
```

然后也需要对网络库进行初始化，并在成功调用网络库后显示成功唤醒网络,在调用失败后显示出现了一点问题导致唤醒网络失败，请重启后再试(>﹏<),并退出程序。

```
    //初始化网络库DLL
    WSADATA wsaData;
    if (WSAStartup(MAKEWORD(2, 2), &wsaData) == 0) // 调用者希望的最高版本——
2.2版和可用socket详细信息
    {
        std::cout << "成功唤醒网络" << std::endl;
    }
    else {
        std::cout << "出现了一点问题导致唤醒网络失败，请重启后再试(>﹏<)" <<
std::endl;
        return 0;
    }
```

接下来创建Socket并尝试和服务端进行连接，如果目前有服务端在等待连接并且连接成功，则会显示那个人终于等到你了，现在可以开始你们的聊天了，否则则会显示那个人还没有出现，请重启后再试(>﹏<)，并退出程序。

```
    sockaddr_in sockAddr;
    memset(&sockAddr, 0, sizeof(sockAddr));  //每个字节都用0填充
    sockAddr.sin_family = PF_INET;
    sockAddr.sin_addr.s_addr = inet_addr("127.0.0.1");
    sockAddr.sin_port = htons(1234);
    SOCKET sock = socket(AF_INET, SOCK_STREAM, 0);

    if (connect(sock, (SOCKADDR*)&sockAddr, sizeof(SOCKADDR)) == 0)
    {
        std::cout << "那个人终于等到你了，现在可以开始你们的聊天了" << std::endl;
    }
    else {
        std::cout << "那个人还没有出现，请重启后再试(>﹏<)" << std::endl;
```

```
        return 0;
    }
```

最后也是对收发消息分别创建线程，聊天开始；并在聊天结束后关闭Socket及网络库的调用。

```
    HANDLE hThread[2]; // 为收发数据分别创建线程
    hThread[0] = CreateThread(NULL, 0, Recv, (LPVOID)&sock, 0, NULL); // 创
建接收线程
    hThread[1] = CreateThread(NULL, 0, Send, (LPVOID)&sock, 0, NULL); // 创
建发送线程
    WaitForMultipleObjects(2, hThread, TRUE, INFINITE);
    CloseHandle(hThread[0]);
    CloseHandle(hThread[1]);
    closesocket(sock);
    WSACleanup();
```

(3)多线程

由于考虑到现实情况中往往都是多条消息一起发送，和多条消息一起接收而不是双方一次发一条后就只能等待对方的消息到来，所以建立了多线程。

在多线程中其实只需要将想运行（即想提升权限）的代码段封装成一个函数并进行调用即可，在客户端和服务端的收发函数基本上是一样的，所以放在一起介绍。

首先是发送函数，为消息的发送创建一个缓冲区并使用while循环来实现线程的不断运行，在发送消息成功后打印带有时间的成功的信息，并在检测到有bye的字样时退出线程。

```
DWORD WINAPI Send(LPVOID sockpara) { // 发送消息函数
    SOCKET * sock = (SOCKET*)sockpara;
    char bufSend[BUF_SIZE] = { 0 };
    while (1) {
        //printf("Input a string: ");
        std::cin >> bufSend;
        // 发送数据
        int t = send(*sock, bufSend, strlen(bufSend), 0);
        // 发送bye代表结束聊天
        if (strcmp(bufSend, "bye") == 0)
        {
            SYSTEMTIME st = { 0 };
            GetLocalTime(&st); // 获取当前时间
            closesocket(*sock); // 关闭套接字
            std::cout <<
"============================================================" <<
std::endl;
            std::cout << "你在" << st.wDay << "号" << st.wHour << "点" <<
st.wMinute << "分" << st.wSecond << "秒离开了与那个人聊天" << std::endl;
            std::cout <<
"============================================================" <<
std::endl;
```

```
            return 0L;
        }
        if (t > 0) {
            SYSTEMTIME st = { 0 };
            GetLocalTime(&st);
            std::cout << "-----------------------------------------------
-------------" << std::endl;
            std::cout << "你在" << st.wDay << "号" << st.wHour << "点" <<
st.wMinute << "分" << st.wSecond << "秒发给了那个人一条消息\n";
            std::cout << "-----------------------------------------------
-------------" << std::endl;
        }
        memset(bufSend, 0, BUF_SIZE);
    }
}
```

其次是接收函数，与发送函数很相近，也是为消息的接收创建一个缓冲区并使用while循环来实现线程的不断运行，在接收消息成功后打印带有时间的成功的信息和消息的内容，并在检测到有bye的字样时退出线程。

```
DWORD WINAPI Recv(LPVOID sock_) { // 接收消息函数
    char bufRecv[BUF_SIZE] = { 0 };
    SOCKET *sock = (SOCKET*)sock_; // 创建套接字
    while (1) {
        int t = recv(*sock, bufRecv, BUF_SIZE, 0); // 接收数据
        if (strcmp(bufRecv, "bye") == 0)
        {
            SYSTEMTIME st = { 0 };
            GetLocalTime(&st); // 获取当前时间
            closesocket(*sock);
            std::cout <<
"============================================================" <<
std::endl;
            std::cout << "那个人在" << st.wDay << "号" << st.wHour << "点" <<
st.wMinute << "分" << st.wSecond << "秒离开了与你的聊天" << std::endl;
            std::cout <<
"============================================================" <<
std::endl;
            return 0L;
        }
        if (t > 0) {
            SYSTEMTIME st = { 0 };
            GetLocalTime(&st);
            std::cout << "-----------------------------------------------
-------------" << std::endl;
            std::cout << "那个人在" << st.wDay << "号" << st.wHour << "点" <<
st.wMinute << "分" << st.wSecond << "秒发过来一条消息:\n";
            printf("%s\n", bufRecv);
            std::cout << "-----------------------------------------------
-------------" << std::endl;
        }
```

```
            memset(bufRecv, 0, BUF_SIZE); // 将存放数据的数组初始化为0
        }
    }
```

五、问题与思考

(1)问题

在多线程设计的时候，会将sockAddr传入给函数，在每个线程中创建套接字，并在线程结束后对其进行关闭。

但是这种方式的设计会出现两方面的问题，程序会被发送消息的线程或者是接收消息的线程所卡死。如果被接收线程卡住，则界面不会收到任何消息，但程序还在运行；如果被发送线程卡住，程序依旧在运行，但是对方却接收不到你输入的消息。

(2)思考

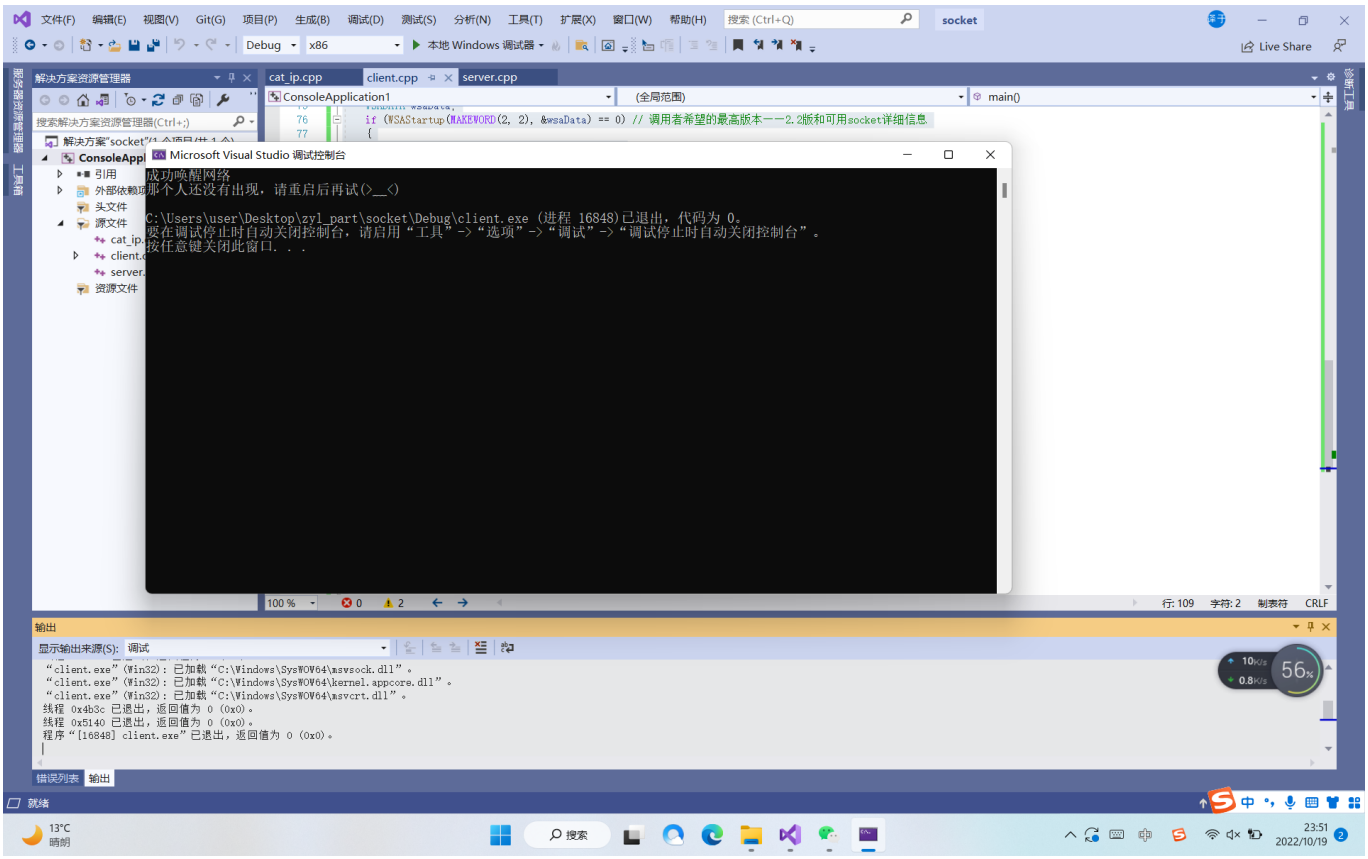
对于这个问题，我一开始想的是可能是Socket产生了冲突，所以我在main函数中直接初始化一个Socket，然后将其传入到线程中，本来以为大功告成了，但还是出现被卡住的问题。

经过许久的思考之后，我感觉可能第一种解决办法并不能改变的原因是冲突未解决，那我传入地址是否可以呢，不出所料，成功了。

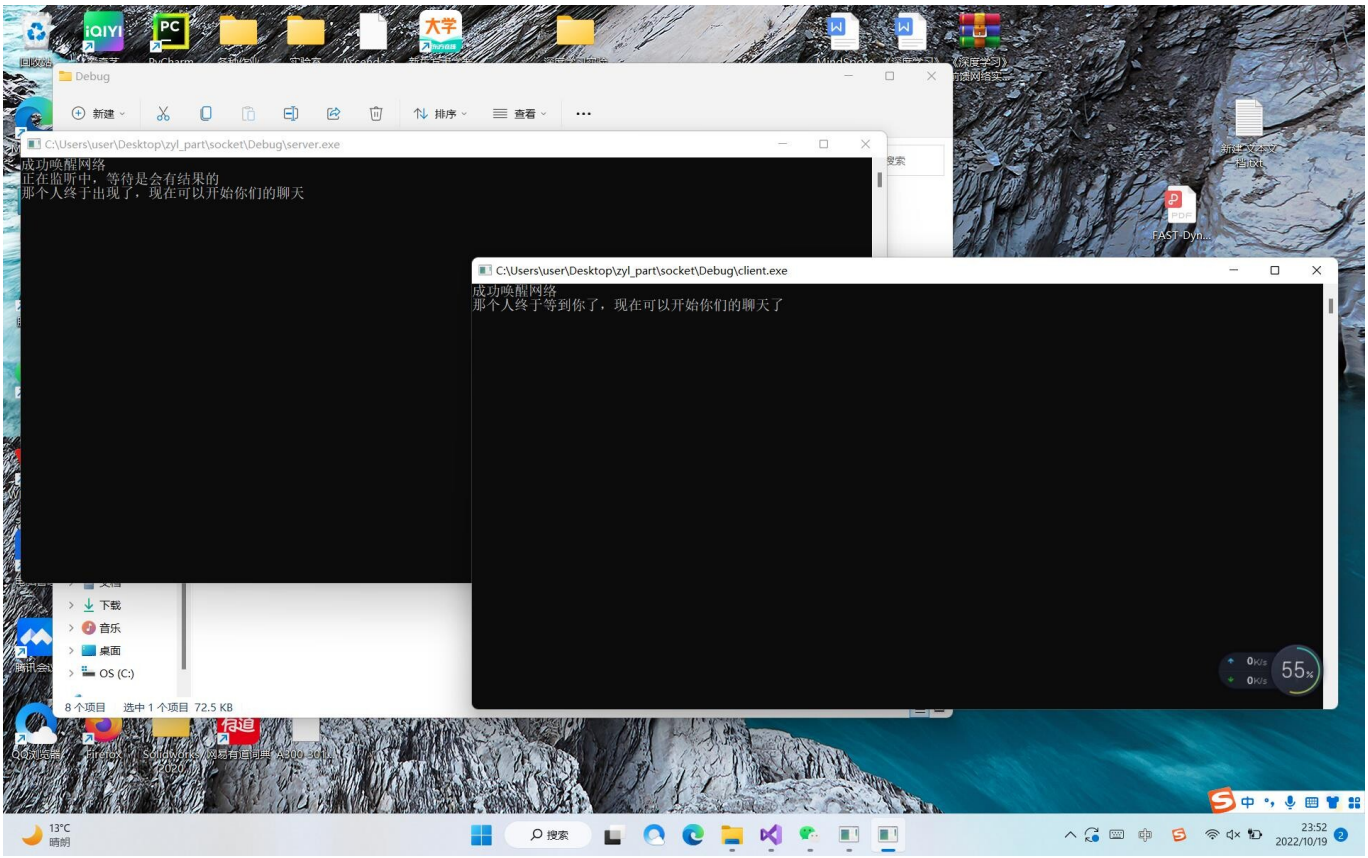
本次实验结束后，在CSDN上搜索后发现，多线程中如果Socket在线程中创建，或者只是简单的传入,则会在执行时发生两个Socket请求一个端口的问题。但是如果传入的是地址，则两个线程可以在收发消息时进行复用，解决问题。

六、执行程序

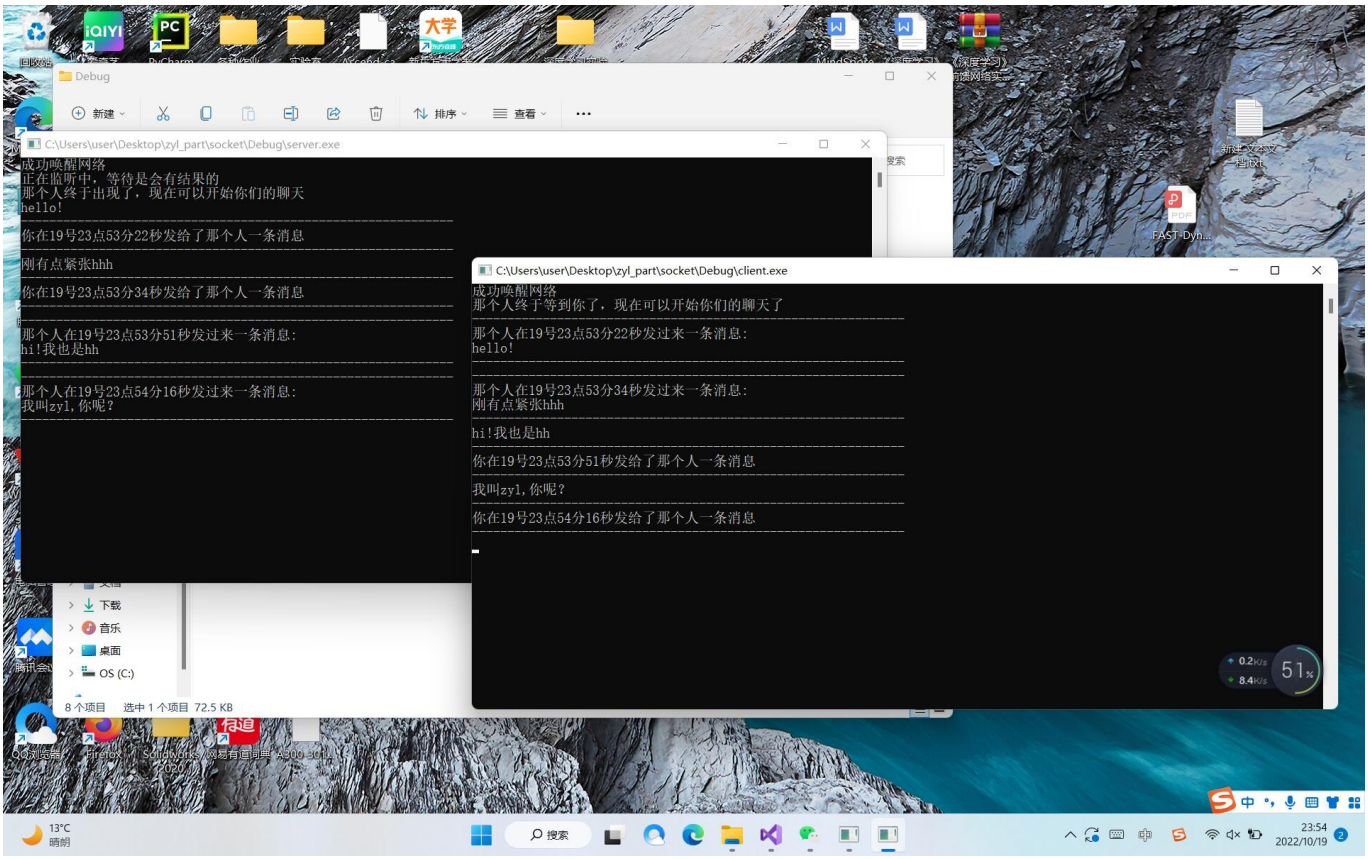
> 客户端没有连接上服务端时的界面

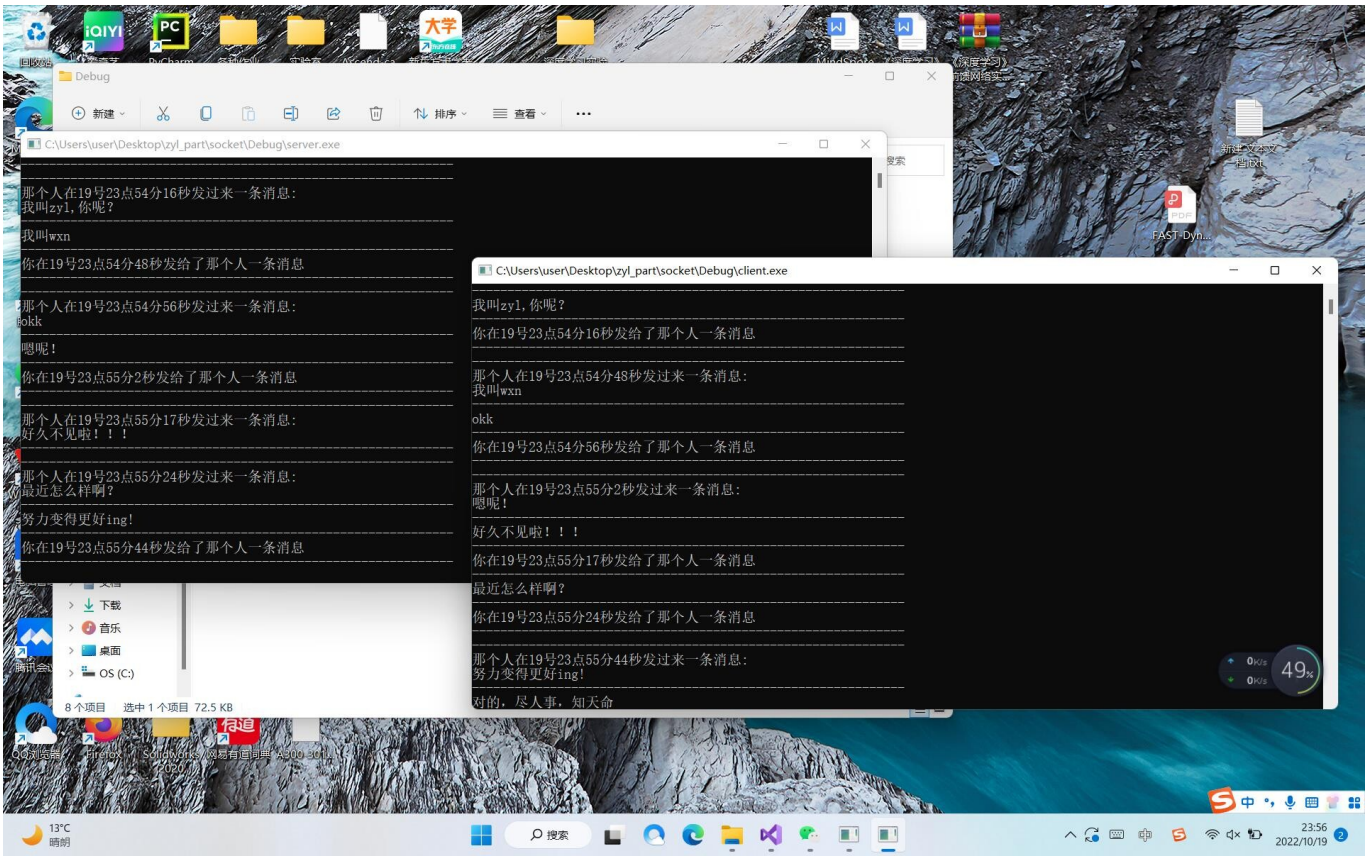


客户端和服务端成功连接的界面

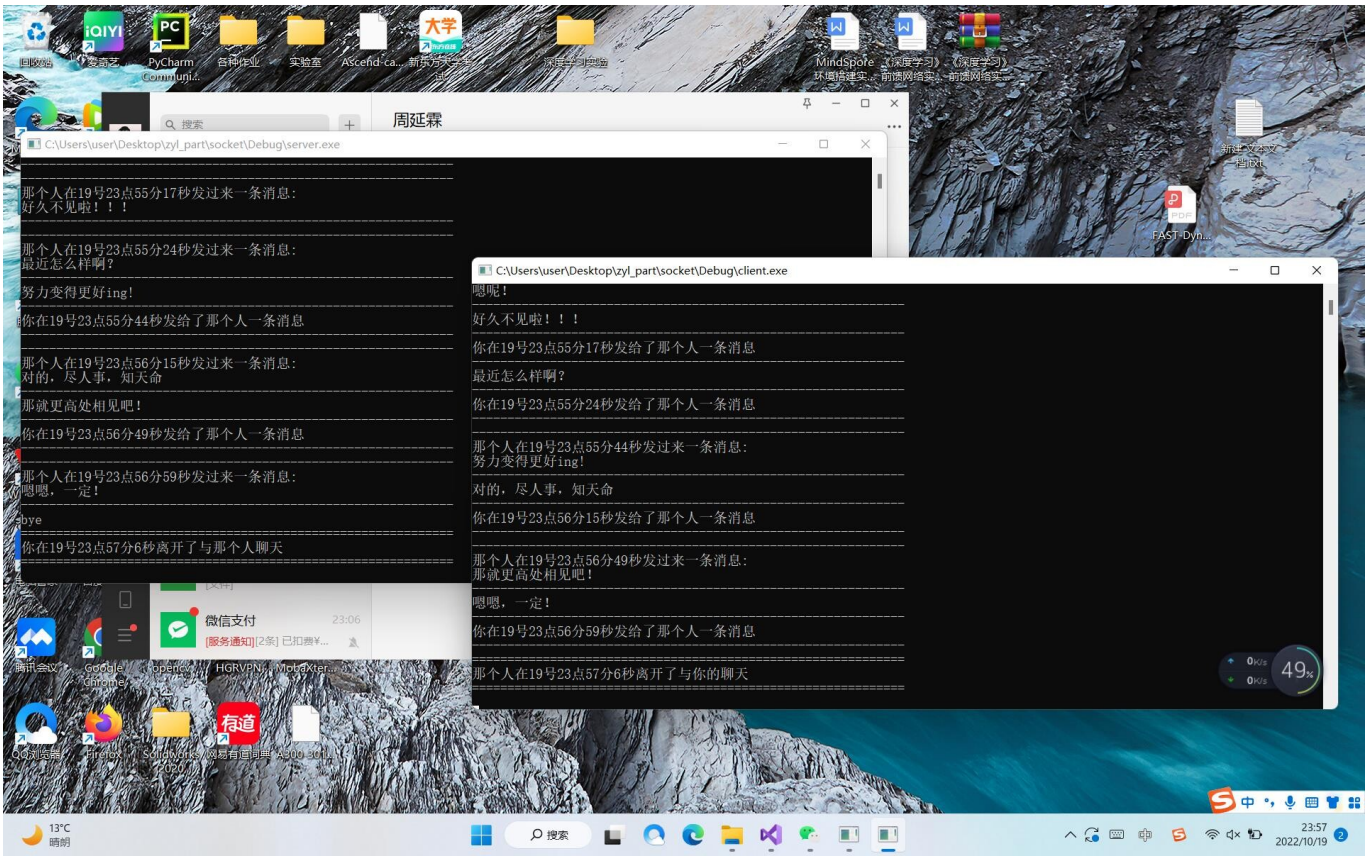


双方互相聊天的界面（可以实现一方同时发多条信息）





当收到bye后结束聊天的界面



七、总结与展望

(1)总结

本次实验是计算机网络的第一次实验，对我还是蛮有难度的，虽然说在去年的Java课程中也有关于网络的相关内容，但是在Java中大多数的东西都是封装好的，并不需要自己去写，所以对于网络偏下层一些的东西并不是很了解。

在这次实验中用的是C++语言，也算是回顾了大一所学的内容，通过本次实验，也让我对网络的地址、端口、Socket、各种函数等更加的了解，也对C++中的多线程编程有了一定的掌握。

(2)展望

本次实验让我对网络方面的东西产生了很大的兴趣，由于本学期也选上了网络技术与应用这门课，感觉这两门课所用的东西是相辅相成的，希望自己可以结合运用，在本学期得到更好的发展。