1.解释cookies？

类型为“小型文本文件”，是某些网站为了辨别用户身份，进行Session跟踪而储存在用户本地终端上的数据（通常经过加密），由用户客户端计算机暂时或永久保存的信息。

2.解释P2P？

对等网络，即对等计算机网络，是一种在对等者（Peer）之间分配任务和工作负载的分布式应用架构，是对等计算模型在应用层形成的一种组网或网络形式。

点对点 端对端

参与者既是资源、服务和内容的提供者（Server），又是资源、服务和内容的获取者（Client）。

3.解释CDN？

Content Delivery Network，内容分发网络。

构建在现有网络基础之上的智能虚拟网络，依靠部署在各地的边缘服务器，通过中心平台的负载均衡、内容分发、调度等功能模块，使用户就近获取所需内容，降低网络拥塞，提高用户访问响应速度和命中率。

4.实验一

实验要求：

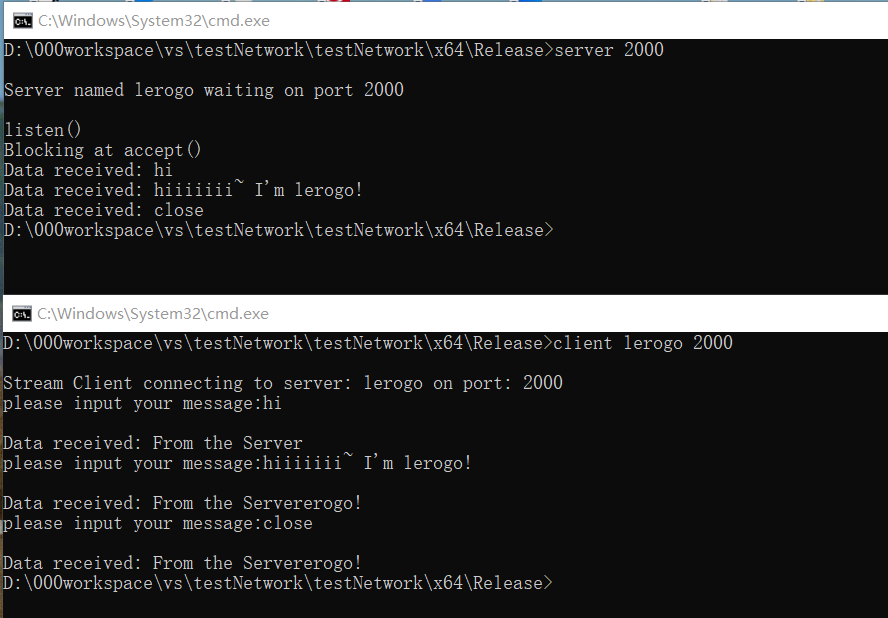
（√）1．>将这两个程序的代码及实现方法分析清楚，将每一行注释清楚。

（√）2．>将这两个程序改成基本的会话程序，能实现客户与服务器的会话。

（√）3．>可扩展会话程序的功能。记录调试这两个程序的过程。

具体见**末尾附件**。

截图：

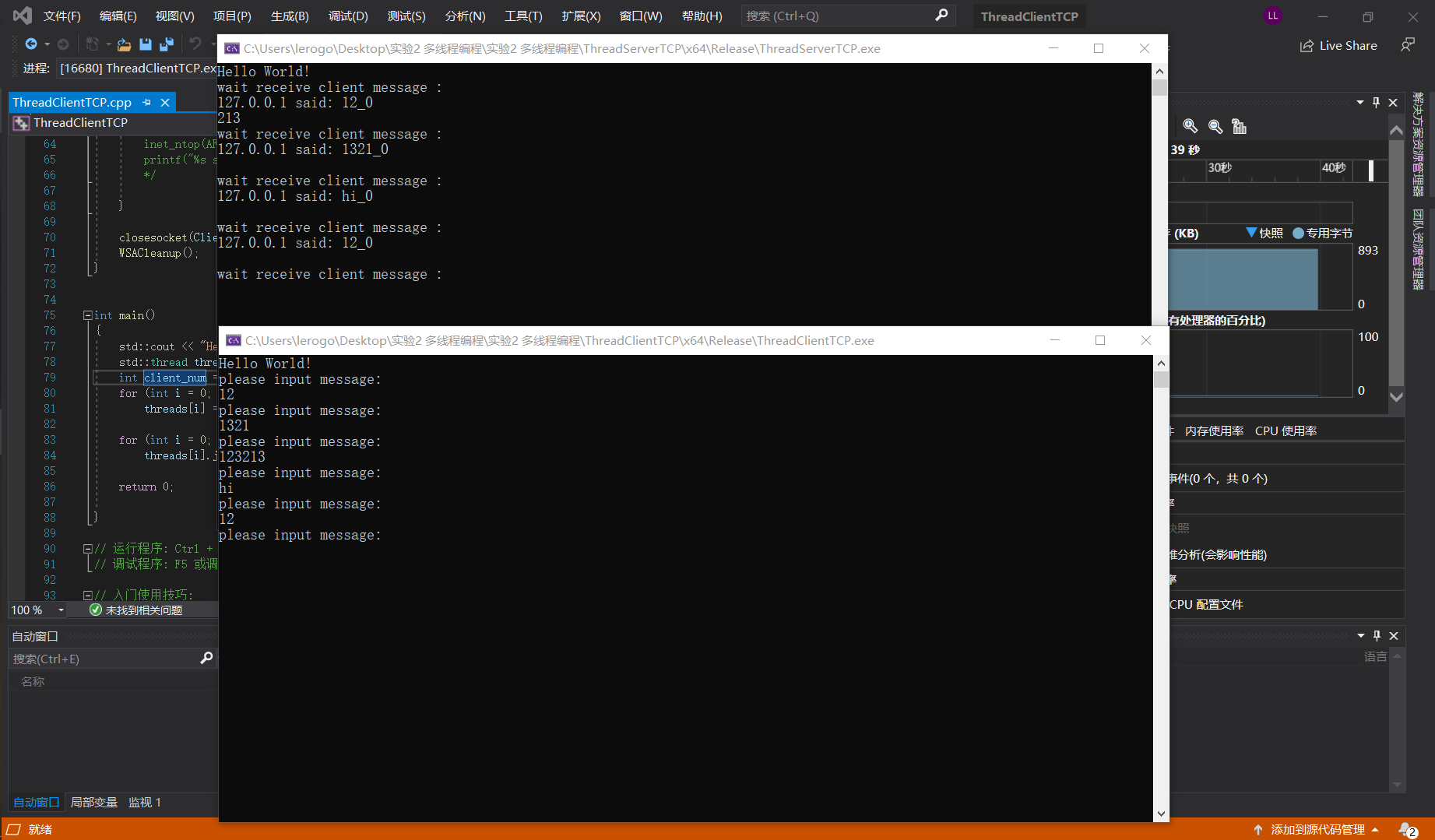


4.实验二

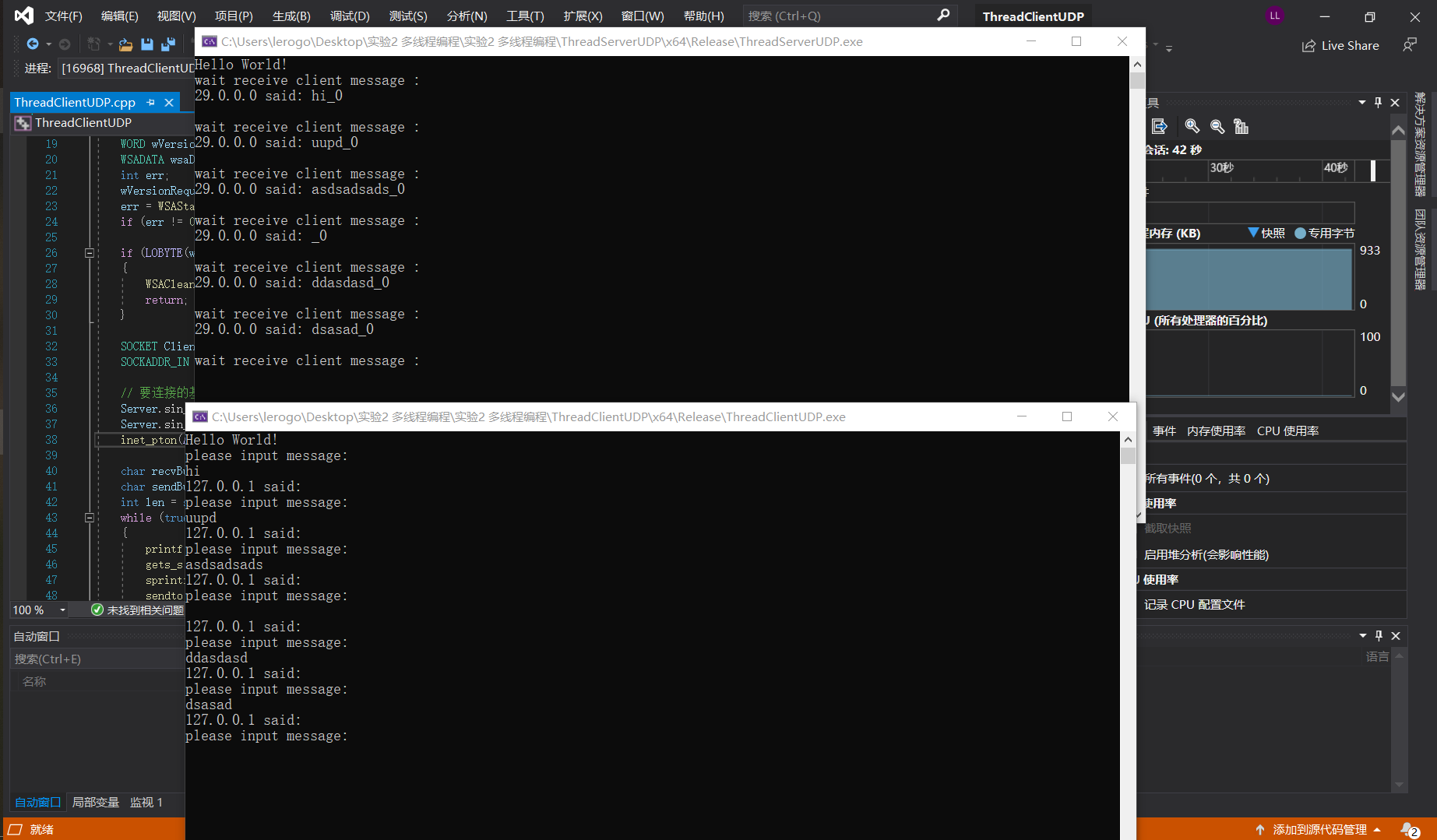
本实验用到的代码和实验一差距不大 只是用的127.0.0.1找地址。

运行截图：

TCP：



UDP：



# 实验一 附件：

## //server.cpp

// Server.cpp

// Extremely simple, stream server example. 简单的服务器示例

// Works in conjunction with Client.cpp. 与Client.cpp协同工作。

// The program sets itself up as a server using the TCP设置为使用TCP的服务器

//等待来自客户机的数据，显示传入的数据，向客户端发送消息，然后退出。

// protoocl. It waits for data from a client, displayss

// the incoming data, sends a message back to the client

// and then exits.

// 编译并链接到wsock32.lib #pragma comment(lib,” wsock32.lib”)s

// Compile and link with wsock32.lib

//

// Pass the port number that the server should bind() to

// on the command line. Any port number not already in use

// can be specified.

//

// Example: Server 2000 以端口号2000为例

//

//lib 文件

#pragma comment(lib,"wsock32.lib")

//头文件

#include <stdio.h>//标准输入输出头文件

#include <winsock.h>//相对于Winsock2.h的老版本

#include <string.h>//字符串

// Function prototype

//函数声明

void StreamServer(short nPort);

// Helper macro for displaying errors

//输出error辅助宏显示错误

#define PRINTERROR(s)   \

        fprintf(stderr,"\n%: %d\n", s, WSAGetLastError())

//函数是socket编程时需要用到的一个函数，使用 WSAGetLastError() 函数来获得上一次的错误代码,而不是依靠全局错误变量, 是为了提供和将来的多线程环境相兼容。

/////////////////////////////////////////////////////////

void main(int argc, char\*\* argv)

{

    //变量声明

    //高位字节指明副版本、低位字节指明主版本

    WORD wVersionRequested = MAKEWORD(1, 1);        //SDK这个宏创建一个无符号16位整型，通过连接两个给定的无符号参数。

    WSADATA wsaData;                                //定义结构体变量wsaData

    int nRet;                                       //状态

    short nPort;                                    // 端口 port

    //

    // Check for port argument

    //

    //参数不够

    if (argc != 2)

    {

        //输出流

        fprintf(stderr, "\nSyntax: server PortNumber\n");

        return;

    }

    nPort = atoi(argv[1]);  //获取port atoi字符转int

    //

    // Initialize WinSock and check version

    //

    nRet = WSAStartup(wVersionRequested, &wsaData); //初始化连接

    //初始化错误

    if (wsaData.wVersion != wVersionRequested)

    {

        fprintf(stderr, "\n Wrong version\n");

        return;

    }

    // Do the stuff a stream server does

    StreamServer(nPort);

    // Release WinSock

    // 结束  释放

    WSACleanup();

}

////////////////////////////////////////////////////////////

void StreamServer(short nPort)

{

    //

    // Create a TCP/IP stream socket to "listen" with

    // 创建 TCP/IP 监听

    SOCKET  listenSocket;

    listenSocket = socket(AF\_INET,          // IPv4

        SOCK\_STREAM,        // TCP

        IPPROTO\_TCP);       // TCP

    //创建成功与否

    if (listenSocket == INVALID\_SOCKET)

    {

        PRINTERROR("socket()");

        return;

    }

    //

    // Fill in the address structure

    //

    SOCKADDR\_IN saServer;// 定义存放本地地址信息的结构体变量addrSrv，存放本地地址信息

    saServer.sin\_family = AF\_INET;// sin\_family指代协议族，其值一般是AF\_INET表示采用IPv4协议

    saServer.sin\_addr.s\_addr = INADDR\_ANY;  // 响应所有地址

    saServer.sin\_port = htons(nPort);       //将端口号赋给sin\_port ，htons()就是将一个数的高低位互换

    //

    // bind the name to the socket

    //

    int nRet;

    // 将listenSocket绑定到相应端口(nPort)上。绑定未必成功，如无错误发生，则bind()返回0。否则的话，将返回 - 1。

    nRet = bind(listenSocket,               // Socket

        (LPSOCKADDR)&saServer,      // Our address

        sizeof(struct sockaddr));   // Size of address structure

    //错误判断

    if (nRet == SOCKET\_ERROR)

    {

        PRINTERROR("bind()");

        closesocket(listenSocket);

        return;

    }

    //

    // This isn't normally done or required, but in this

    // example we're printing out where the server is waiting

    // so that you can connect the example client.

    //

    //打印server 让client找到

    int nLen;

    nLen = sizeof(SOCKADDR);

    char szBuf[256];

    //获取计算机名字

    nRet = gethostname(szBuf, sizeof(szBuf));

    if (nRet == SOCKET\_ERROR)

    {

        PRINTERROR("gethostname()");

        closesocket(listenSocket);

        return;

    }

    //

    // Show the server name and port number

    // 打印server名字（主机名）

    printf("\nServer named %s waiting on port %d\n",

        szBuf, nPort);

    //

    // Set the socket to listen

    //

    printf("\nlisten()");

    // 监听

    nRet = listen(listenSocket,                 // Bound socket绑定socket

        SOMAXCONN);                             // Number of connection request queue队列数（5）

    if (nRet == SOCKET\_ERROR)

    {

        PRINTERROR("listen()");

        closesocket(listenSocket);

        return;

    }

    //

        // Wait for an incoming request

        //

    SOCKET  remoteSocket;

    printf("\nBlocking at accept()");

    //在一个套接口接受的一个连接

    remoteSocket = accept(listenSocket,         // Listening socket

        NULL,                                   // Optional client address

        NULL);

    if (remoteSocket == INVALID\_SOCKET)

    {

        PRINTERROR("accept()");

        closesocket(listenSocket);

        return;

    }

    bool alwaysAccept = true;

    while (alwaysAccept) {

        //

        // We're connected to a client

        // New socket descriptor returned already

        // has clients address

        //

        // Receive data from the client

        //

        //从TCP连接的另一端接收数据。

        memset(szBuf, 0, sizeof(szBuf));

        nRet = recv(remoteSocket,           // Connected client 指定接收端套接字描述符

            szBuf,                          // Receive buffer 指明一个缓冲区，该缓冲区用来存放recv函数接收到的数据；

            sizeof(szBuf),                  // Lenght of buffer 指明buf的长度；

            0);                             // Flags

        if (nRet == INVALID\_SOCKET)

        {

            PRINTERROR("recv()");

            closesocket(listenSocket);

            closesocket(remoteSocket);

            return;

        }

        if (!strcmp(szBuf,"close")) {

            alwaysAccept = false;

        }

        //

        // Display received data

        //

        printf("\nData received: %s", szBuf);

        //提供服务

        //

        // Send data back to the client

        //

        strcpy\_s(szBuf, "From the Server");

        //向TCP连接的另一端发送数据。

        nRet = send(remoteSocket,               // Connected socket     指定发送端套接字描述符；

            szBuf,                      // Data buffer                  存放应用程序要发送数据的缓冲区；

            strlen(szBuf),              // Lenght of data               实际要发送的数据的字节数；

            0);                         // Flags

    }

    //

    // Close BOTH sockets before exiting

    // 关闭

    closesocket(remoteSocket);

    closesocket(listenSocket);

    return;

}

## //client.cpp

// Extremely simple, stream client example.简单的客户端示例

// Works in conjunction with Server.cpp.与Server.cpp协同工作

// The program attempts to connect to the server and port specified on the command line. 程序尝试连接到命令行中指定的服务器和端口。

// The Server program prints the needed information when it is started. Once connected,the program sends data to the server, waits for a response and then exits. 服务器程序在启动时打印所需的信息。连接后，程序将向服务器发送数据，等待响应，然后退出。

// Compile and link with wsock32.lib. 编译并链接到wsock32.lib #pragma comment(lib,” wsock32.lib”)

// Pass the server name and port number on the command line. 在命令行上传递服务器名称和端口号。

// Example: Client MyMachineName 2000连接端口2000，我的主机名字是sc-PC

#include <stdio.h>   //标准输入输出头文件

#include <winsock.h> //相对于Winsock2.h的老版本

#include <string.h>  //string类

// Function prototype声明协议函数

void StreamClient(char \*szServer, short nPort);

// Helper macro for displaying errors辅助宏显示错误

#define PRINTERROR(s)   \                            //函数printerror()功能:系统返回提示信息，例如添加信息跳转提示。

fprintf(stderr, "\n%: %d\n", s, WSAGetLastError()) //使用 WSAGetLastError() 函数 来获得上一次的错误代码,为了提供和将来的多线程环境相兼容。

    void main(int argc, char \*\*argv)

{

    WORD wVersionRequested = MAKEWORD(1, 1); //wVersionRequested是Windows Sockets API提供的调用方可使用的最高版本号，MAKEWORD(1, 1)第一个参数为低位字节；第二个参数为高位字节

    WSADATA wsaData;                         //定义结构体变量wsaData

    int nRet;

    short nPort;

    //检查主机和端口参数

    if (argc != 3)

    {

        fprintf(stderr, "\nSyntax: client ServerName PortNumber\n"); //fprintf()函数根据指定的format(格式)发送信息(参数)到由stream(流)指定的文件

        return;

    }

    nPort = atoi(argv[2]);

    //对WinSock进行初始化并检查版本

    nRet = WSAStartup(wVersionRequested, &wsaData); //WSAStartup()函数向操作系统说明要用哪个库文件,因此就可以将库文件与当前的应用程序绑定，从而调用该版本的socket的各种函数,WSAStartup()进行相应的socket库绑定。

    if (wsaData.wVersion != wVersionRequested)      //wVersionRequested 要使用的 Windows Sockets API 版本

    {

        fprintf(stderr, "\n Wrong version\n"); // fprintf()函数根据指定的format(格式)发送信息(参数)到由stream(流)指定的文件

        return;

    }

    StreamClient(argv[1], nPort);

    //释放WinSock

    WSACleanup();

}

void StreamClient(char \*szServer, short nPort) //创建客户流

{

    printf("\nStream Client connecting to server: %s on port: %d",

           szServer, nPort);

    //找服务器

    LPHOSTENT lpHostEntry;

    lpHostEntry = gethostbyname(szServer); // gethostbyname()函数用域名或者主机名获取地址，操作系统提供的库函数

    if (lpHostEntry == NULL)

    {

        PRINTERROR("gethostbyname()"); //printerror()功能：系统返回提示信息，这里是返回提示：gethostbyname()

        return;

    }

    //创建TCP/IP流套接字

    SOCKET theSocket;

    theSocket = socket(AF\_INET,      // 地址族

                       SOCK\_STREAM,  //sock\_stream对应于TCP协议，SOCK\_DGRAM对应UDP协议，SOCK\_RAW对应原始Socket

                       IPPROTO\_TCP); //参数protocol用来指定socket所使用的传输协议编号

    if (theSocket == INVALID\_SOCKET)

    {

        PRINTERROR("socket()"); //返回提示：socket()

        return;

    }

    SOCKADDR\_IN saServer;

    saServer.sin\_family = AF\_INET; // 选择 AF\_INET 的目的就是使用 IPv4 进行通信。IPv4 使用 32 位地址，相比 IPv6 的 128 位来说，计算更快，便于用于局域网通信。

    saServer.sin\_addr = \*((LPIN\_ADDR)\*lpHostEntry->h\_addr\_list);

    // ^服务器的地址

    saServer.sin\_port = htons(nPort); //命令行中的端口号

    //连接服务器

    int nRet;

    nRet = connect(theSocket,                //第一个参数：创建的套接字

                   (LPSOCKADDR)&saServer,    // Server address第二个参数：要链接的服务器地址

                   sizeof(struct sockaddr)); //第三个参数：服务器地址的长度

    if (nRet == SOCKET\_ERROR)

    {

        PRINTERROR("socket()"); //返回提示：socket()

        closesocket(theSocket); //释放套接字

        return;

    }

    bool alwaysConnect = true;

    while (alwaysConnect)

    {

        char szBuf[256];

        printf("\nplease input your message:");

        scanf\_s("%[^\n]%\*c", szBuf);

        if (!strcmp(szBuf, "close"))

        {

            alwaysConnect = false;

        }

        nRet = send(theSocket,     //连接的套接字

                    szBuf,         //数据缓冲区

                    strlen(szBuf), //数据长度

                    0);            // Flags

        if (nRet == SOCKET\_ERROR)

        {

            PRINTERROR("send()");   //返回提示：send()

            closesocket(theSocket); //释放套接字

            return;

        }

        //等待回应

        nRet = recv(theSocket,     //第一个参数：本机创建的套接字

                    szBuf,         // Receive buffer第二个参数：接受消息的字符串

                    sizeof(szBuf), // Size of receive buffer第三个参数：允许接收字符串的最大长度

                    0);            // Flags第四个参数：会对函数行为产生影响，一般设置为0

        if (nRet == SOCKET\_ERROR)

        {

            PRINTERROR("recv()");   //返回提示：recv()

            closesocket(theSocket); //释放套接字

            return;

        }

        //

        // 显示接收到的数据

        //

        printf("\nData received: %s", szBuf);

    }

    closesocket(theSocket); //释放套接字

    return;

}

# 附录2 实验二附件

## //ThreadServerTCP.cpp

// socket通信  服务器

//TCP：在客户端和服务器之间提供可靠的字节流传输（“管道”）

#include "stdafx.h" //stdafx.h中没有函数库，只是定义一些环境参数，使编译出来的程序能在32位的操作系统环境下工作

#pragma comment(lib, "ws2\_32.lib")//连接Winsock2.h的静态库文件

#include <Winsock2.h>//winsock第2版的头文件，里面有多种函数可以帮助使用Winsock

#include <WS2tcpip.h>//设置或获取套接字选项

#include <stdio.h>//标准输入输出头文件

#include <string.h>//包含一些常用的C字符串处理函数

#include <stdlib.h>//包含了C语言的一些常用库函数

#include <thread>//声明std::thread线程类

#include <iostream>//表示输入输出流

void func(SOCKET arg, SOCKADDR\_IN client\_addr, int pos)// socket编程, SOCKADDR\_IN 是声明套接字类型

{

    char recvBuf[128];//接收的字符串 接收Buf,，设置长度为128

    char sendBuf[128];//发送的字符串 发送Buf，设置长度为128

    char tempBuf[256];//临时字符串，设置长度为256

    int sockConn = arg;

    while (true)

    {

        //从客户端接收消息

        printf("wait receive client message :\n");//输出“等待接收客户端的消息”

        recv(sockConn, recvBuf, 128, 0);//

        // 第一个参数：本机创建的套接字

        // 第二个参数：接收的字符串

        // 第三个参数：接收的字符串长度

        // 第四个参数：会对函数行为产生影响，一般设置为0

        //解析客户端地址信息

        char ipClient[16];//客户IP字符串，设置长度为16

        inet\_ntop(AF\_INET, &client\_addr.sin\_addr, ipClient, sizeof(ipClient));//第一个参数指定地址族，对于TCP/IP协议的套接字，地址族为AF\_INET；第二个参数指向套接字的IP地址（sin\_addr）；第三个参数是用户IP；第四个参数是用户IP地址的大小。

        printf("%s said: %s\n", ipClient, recvBuf);//输出“用户IP said: 接收字符串”

        //向客户端发送消息

        gets\_s(sendBuf);

        send(sockConn, sendBuf, strlen(sendBuf) + 1, 0);//

        // 第一个参数：本机创建的套接字

        // 第二个参数：要发送的字符串

        // 第三个参数：发送字符串长度

        // 第四个参数：会对函数行为产生影响，一般设置为0

    }

}

int main()

{

    std::cout << "Hello World!\n";//使用std::是在调用标准命名空间，输出“Hello World!”

/\*

WSAStartup必须是应用程序或DLL调用的第一个Windows Sockets函数。

它允许应用程序或DLL指明Windows Sockets API的版本号及获得特定Windows Sockets实现的细节。

应用程序或DLL只能在一次成功的WSAStartup()调用之后才能调用进一步的Windows Sockets API函数。

\*/

    WORD wVersionRequested;// wVersionRequested是Windows Sockets API提供的调用方可使用的最高版本号

    WSADATA wsaData;// 定义结构体变量wsaData, 用来存储被WSAStartup函数调用后返回的Windows Sockets数据

    int err;//定义整型变量err

    wVersionRequested = MAKEWORD(1, 1);//第一个参数为低位字节；第二个参数为高位字节

    err = WSAStartup(wVersionRequested, &wsaData);//对winsock进行初始化，加载一个Winsock库版本

    if (err != 0) { return 0; }

    if (LOBYTE(wsaData.wVersion)!= 1||HIBYTE(wsaData.wVersion)!=1)//LOBYTE（）取得16进制数最低位；HIBYTE（）取得16进制数最高那个字节的内容

    {

        WSACleanup();//在一个多线程的环境下，WSACleanup（）中止了Windows Sockets在所有线程上的操作。

        return 0;

    }//进行版本检查

    // 申请存储线程的数组

    std::thread threads[10];//使用std的thread创建一个线程对象

    int thread\_num = 0;//先记线程个数为0

    /\*socket通讯\*/

    //申请套接字

    SOCKET Svr = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);//创建SOCKET对象Svr,AF\_INET指IPV4版本, 第一个参数表示ip协议族，第二个参数表示数据流方式，第三个参数是TCP协议。

    SOCKADDR\_IN addr;//定义存放本地地址信息的结构体变量addr

    //要绑定的基础信息

    addr.sin\_family = AF\_INET;//sin\_family指协议族，一般是AF\_INET,表示采用ipv4协议

    addr.sin\_port = htons(6002);//将端口号（6002）赋给sin\_port, htons 将整型类型转换成网络字节序

    addr.sin\_addr.S\_un.S\_addr = htonl(INADDR\_ANY);//htonl将长整型转换成网络字节序

    //进行绑定

    int len = sizeof(sockaddr);

    bind(Svr, (struct sockaddr\*) & addr, len);//指定名字、类型、长度

    //监听套接字

    int ret = listen(Svr, 10); //第一个参数：要监听的socket（套接字）,第二个参数：等待连接队列的最大长度

    if (ret == SOCKET\_ERROR)//若发生错误

    {

        printf("侦听失败\n");

        closesocket(Svr);//释放套接字Svr

    }

    //存储请求连接的套接字信息

    SOCKADDR\_IN addrClient;

    while (true)

    {

        //接受连接，返回一个socket

        SOCKET sockConn = accept(Svr, (struct sockaddr\*) & addrClient, &len);

        if (sockConn == INVALID\_SOCKET)

        {

            //printf("无效socket\n");

            continue;

        }

        //将通讯细节放在线程里处理

        threads[thread\_num] = std::thread(func, sockConn, addrClient, thread\_num);

        thread\_num++;

        if (thread\_num == 5)

        {

            printf("线程池达到数量上限");

        }

    }

    //等待线程结束

    for (int i = 0; i < thread\_num; i++)

        threads[i].join();

    //释放套接字

    closesocket(Svr);

    WSACleanup();//终止对套接字库的使用

    return 0;

}

// 运行程序: Ctrl + F5 或调试 >“开始执行(不调试)”菜单

// 调试程序: F5 或调试 >“开始调试”菜单

// 入门使用技巧:

//   1. 使用解决方案资源管理器窗口添加/管理文件

//   2. 使用团队资源管理器窗口连接到源代码管理

//   3. 使用输出窗口查看生成输出和其他消息

//   4. 使用错误列表窗口查看错误

//   5. 转到“项目”>“添加新项”以创建新的代码文件，或转到“项目”>“添加现有项”以将现有代码文件添加到项目

//   6. 将来，若要再次打开此项目，请转到“文件”>“打开”>“项目”并选择 .sln 文件

## //ThreadClientTCP.cpp

//socket通信  客户端

//TCP：在客户端和服务器之间提供可靠的字节流传输（“管道”）

#include "stdafx.h"//stdafx.h中没有函数库，只是定义一些环境参数，使编译出来的程序能在32位的操作系统环境下工作

#pragma comment(lib, "ws2\_32.lib")//连接Winsock2.h的静态库文件

#include <Winsock2.h>//winsock第2版的头文件，里面有多种函数可以帮助使用Winsock

#include <WS2tcpip.h>//设置或获取套接字选项

#include <stdio.h>//标准输入输出头文件

#include <string.h>//包含一些常用的C字符串处理函数

#include <stdlib.h>//包含了C语言的一些常用库函数

#include <thread>//声明std::thread线程类

#include <iostream>//表示输入输出流

void new\_client(int pos)

{

    WORD wVersionRequested;//wVersionRequested是Windows Sockets API提供的调用方可使用的最高版本号

    WSADATA wsaData;//定义结构体变量wsaData

    int err;//定义整型变量err

    wVersionRequested = MAKEWORD(1, 1);//第一个参数为低位字节；第二个参数为高位字节

    err = WSAStartup(wVersionRequested, &wsaData);//对winsock进行初始化，加载一个Winsock库版本

    if (err != 0) { return; }

    if (LOBYTE(wsaData.wVersion)!=1||HIBYTE(wsaData.wVersion)!=1)//LOBYTE（）取得16进制数最低位；HIBYTE（）取得16进制数最高那个字节的内容

    {

        WSACleanup();//在一个多线程的环境下，WSACleanup（）中止了Windows Sockets在所有线程上的操作。

        return;

    }

    SOCKET Client = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);//第一个参数表示ip协议族，第二个参数表示数据流方式，第三个参数是TCP协议

    SOCKADDR\_IN Server;//定义存放本地地址信息的结构体变量Server

    // 要连接的基础信息

    Server.sin\_family = AF\_INET;//sin\_family指协议族，一般是AF\_INET,表示采用ipv4协议

    Server.sin\_port = htons(6002);//将端口号（6002）赋给sin\_port, htons 将整型类型转换成网络字节序

    inet\_pton(AF\_INET, "127.0.0.1", &Server.sin\_addr); //点分十进制地址转换成网络字节序

    // 向服务端发起连接

    int ret = connect(Client, (struct sockaddr\*) & Server, sizeof(Server));//

        //第一个参数：创建的套接字

        //第二个参数：要链接的套接字地址

        //第三个参数：单位长度

    if (ret == SOCKET\_ERROR)

    {

        printf("连接失败\n");

        closesocket(Client);//释放套接字Client

        WSACleanup();//终止对套接字库的使用

        return;

    }

    char recvBuf[128];//接收字符串，设置长度为128

    char sendBuf[128];//发送字符串，设置长度为128

    while (true)

    {

        printf("please input message:\n");//输出“请输入信息：”

        gets\_s(sendBuf);//得到输出字符串

        sprintf\_s(sendBuf, "%s\_%d", sendBuf, pos);

        send(Client, sendBuf, strlen(sendBuf) + 1, 0);//

        // 第一个参数：本机创建的套接字

        // 第二个参数：要发送的字符串

        // 第三个参数：发送字符串长度

        // 第四个参数：会对函数行为产生影响，一般设置为0

        recv(Client, recvBuf, 128, 0);//

        // 第一个参数：本机创建的套接字

        // 第二个参数：接受消息的字符串

        // 第三个参数：允许接收字符串的最大长度

        // 第四个参数：会对函数行为产生影响，一般设置为0

        char ipServer[16];//服务器方的IP，设置大小为16字节

        inet\_ntop(AF\_INET, &Server.sin\_addr, ipServer, sizeof(ipServer));//第一个参数指定地址族为AF\_INET；第二个参数指向套接字的IP地址（sin\_addr）；第三个参数是服务器IP；第四个参数是服务器IP地址的大小

        printf("%s said: %s\n", ipServer, recvBuf);//输出“服务器IP said: 接收字符串”

    }

    closesocket(Client);//释放套接字

    WSACleanup();//终止对套接字库的使用

}

int main()

{

    std::cout << "Hello World!\n";//输出“Hello World!”

    std::thread threads[10];//建立线程对象

    int client\_num = 1;

    for (int i = 0; i < client\_num; i++)

        threads[i] = std::thread(new\_client, i);

    for (int i = 0; i < client\_num; i++)//等待线程结束

        threads[i].join();

    return 0;

}

// 运行程序: Ctrl + F5 或调试 >“开始执行(不调试)”菜单

// 调试程序: F5 或调试 >“开始调试”菜单

// 入门使用技巧:

//   1. 使用解决方案资源管理器窗口添加/管理文件

//   2. 使用团队资源管理器窗口连接到源代码管理

//   3. 使用输出窗口查看生成输出和其他消息

//   4. 使用错误列表窗口查看错误

//   5. 转到“项目”>“添加新项”以创建新的代码文件，或转到“项目”>“添加现有项”以将现有代码文件添加到项目

//   6. 将来，若要再次打开此项目，请转到“文件”>“打开”>“项目”并选择 .sln 文件