****

《网络程序设计》实验报告

**实验1 Visual Studio 控制台编程getservbyname()**

【实验目的】

掌握Visual Studio控制台应用编程的基本方法

掌握Windows Sockets DLL的初始化和释放方法

掌握Windows Sockets API调用的一般步骤

使用Windows Sockets的API函数获得指定机器的信息

（包括主机名、服务名、协议、IP地址）

【实验过程】

1. **调用WSAStartup()进行初始化**

初始化Windows Sockets DLL需要使用函数WSAStartup(),其他套接字函数都要在WSAStartup()被成功调用后才能正常工作。

（1）创建类型为WSADATA的对象：

WSADATA wsaData;

（2）调用函数WSAStartup(),并根据返回值判断错误信息

iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);

if (iResult != 0) {

printf("WSAStartup failed: %d\n", iResult);

return -1;

}

1. **调用Windows Socket的GetServByName ()函数用于根据给定名字查找相应服务。**

Getservname()函数原型为：

struct servent \*getservbyname(const char \*servname, const char \*protoname)

调用GetServByName ()函数：

remoteServ = getservbyname(serv\_name, NULL);

1. **显示出函数能够获得的信息。**

printf("服务名: %s \n", remoteServ->s\_name);

printf("端口号: %d \n", ntohs((u\_short)remoteServ->s\_port));

printf("协议名: %s \n", remoteServ->s\_proto);

**4.调用WSACleanup ()，释放资源。**

当应用程序完成Windows Sockets使用后，应用程序或DLL必须调用WSACleanup()将其从Windows Sockets的实现中注销，并且该实现释放为应用程序或DLL分配的任何资源。

使用WSACleanup()对Windows Sockets DLL进行释放的具体步骤：

iResult = WSACleanup();

if (iResult != 0) {

printf("WSACleanup failed: %d\n", iResult);

return -1;

}

**5.完整代码展示。**

//Getservname()函数原型为：

//struct servent \*getservbyname(const char \*servname, const char \*protoname)

//用于根据给定名字查找相应服务

// getservbyname1.cpp : 定义控制台应用程序的入口点。

#include "stdafx.h"

#include <stdio.h>

#include <winsock2.h>

#include <ws2tcpip.h>

#include <windows.h> // 此头文件放在<winsock2.h>或<ws2tcpip.h>之后

#pragma comment(lib, "ws2\_32.lib") //连接静态库

int main(int argc, char\*\* argv)//argc:参数个数 argv:参数取值

{//声明 初始化变量

WSADATA wsaData; // 创建类型为aWSADATA的对象，存放socket版本号等信息

int iResult; // 接收WSAStarup()函数的返回值

char\* serv\_name;

struct servent\* remoteServ;//定义server entry结构体指针变量

// 注册套接字

//初始化启动winsock,声明使用WinSock2.2版本 调用WSAStartup(),并根据返回值判断错误信息

iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);//与操作系统确认支持的WinSock版本

if (iResult != 0) {

printf("WSAStartup failed: %d\n", iResult);

return -1;

}

char buff[128] = "";

printf("请输入服务名： \n");

while (~scanf("%s",buff))

{

serv\_name = buff;

printf("打印%s 服务内容： \n", serv\_name);

//获取服务器的信息存放在servent结构体中。getservbyname函数用于根据给定名字查找相应服务

remoteServ = getservbyname(serv\_name, NULL);

//显示出函数能够获得的信息

printf("服务名: %s \n", remoteServ->s\_name);//显示服务名

printf("端口号: %d \n", ntohs((u\_short)remoteServ->s\_port));//显示端口号

//ntohs()是一个函数名作用是将一个16位数由网络字节顺序转换为主机字节顺序

printf("协议名: %s \n", remoteServ->s\_proto);//显示协议名

}

//释放Windows Sockets DLL

iResult = WSACleanup();

if (iResult != 0) {

printf("WSACleanup failed: %d\n", iResult);

return -1;

}

return 0;

}

**6. 展示接口函数getservbyname0获得的信息截图。**

分别输入服务名https、http、ssh、ftp、smtp进行测试，并截图如图1所示。

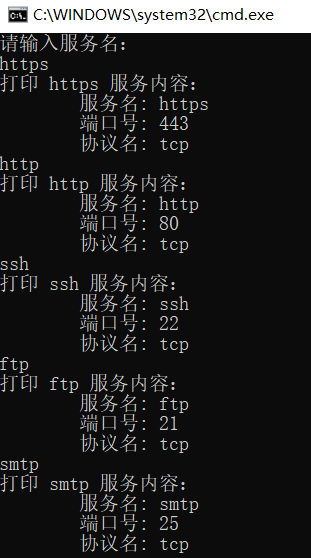


图1 测试截图

【实验总结】

本次编程实验让我对Windows Sockets API调用的一些步骤有了更清楚的认识。

其中，因为 Winsock的服务是以动态链接库形式来实现的，因此必须首先对 Winsock DLL进行初始化，通过调用 Wsastartup0便可以检则系统中有没有一个或者多个 Windows Socket的实现，该函数必须是应用程序或DLL调用的第一个 Windows Socket函数，应用程序或DLL只有在一次成功的 WSAStartup0调用之后，才能够进一步调用其他的 WinSock API函数。不再调用其他的 Winsock API函数时就必须调用 WSACleanup（）将应用程序从Windows Socket的实现中注销，释放资源，多少次调用WSAStartup),就要多少次调用WSACleanup。

**实验2 使用流式套接字实现网络通信**

【实验目的】

学会使用流式套接字编写基于C/S模型的两个通信程序

熟悉流式套接字的常用接口函数

掌握服务器端流式套接字编程的基本流程

掌握客户端流式套接字编程的基本流程

【实验过程】

1)使用基本的服务器编程模式，创建流式套接字，根据指定的端口号绑定服务，建立监听队列，并接受来自客户端的连接请求，收发数据；

使用基本的客户端编程模式，创建流式套接字，通过用户指定的服务器地址向指定服务器发起连接请求，与服务器之间实现收发数据，然后关闭连接。

本机测试：本机回送地址（127.0.0.1），测试本机进程间通信。

不同电脑测试：服务器（192.168.43.27）,客户端（192.168.43.40）。

2)通过改进服务器端，实现服务器循环为多个客户端提供服务；

要求服务器端能够显示对应客户端的ip地址；客户端实现用户输入信息方式与服务器对话。

本机测试：本机回送地址（127.0.0.1），测试本机进程间通信。

不同电脑测试：服务器（192.168.43.27）,客户端（192.168.43.40）。

3)采用 Windows环境下多线程开发方法改进服务器端，对每个客户连接请求独立创建通信线程,实现并发服务器。

本机测试：本机回送地址（127.0.0.1），测试本机进程间通信。

不同电脑测试：服务器（192.168.43.27）, 客户端1（192.168.43.19），客户端2（192.168.43.40）。

**实验结果截图**

1.服务器与客户端一对一通信

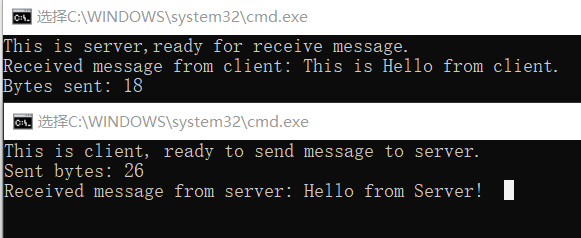
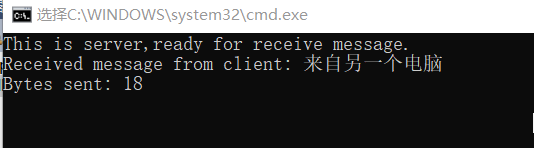
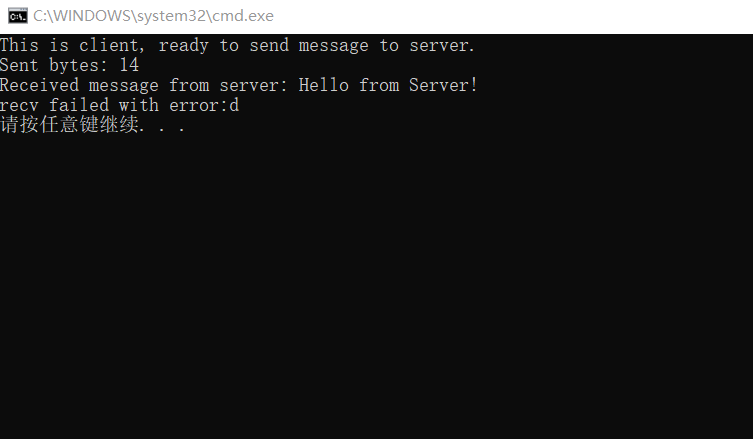


图1 本机服务器与客户端一对一



服务器（192.168.43.27）



客户端（192.168.43.40）

图 2 不同电脑服务器与客户端一对一

2.服务器循环提供服务

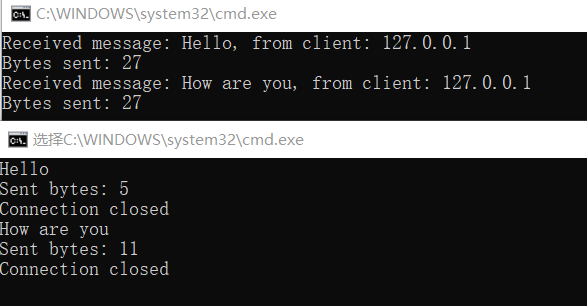
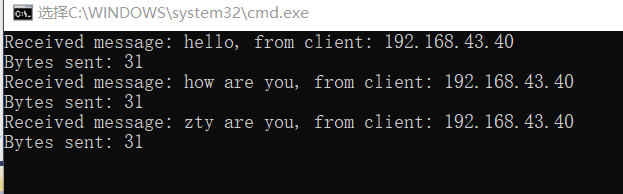
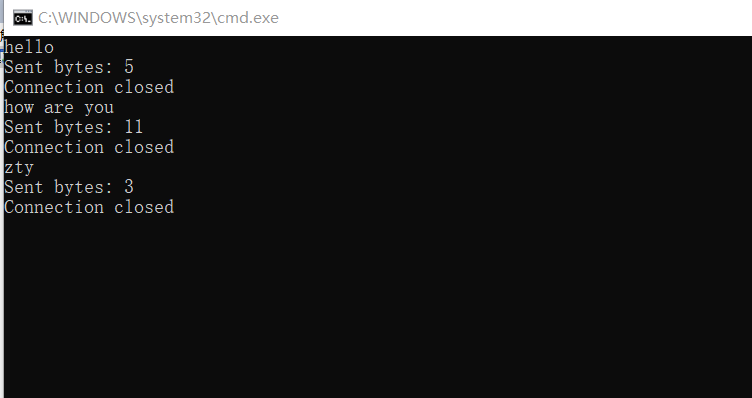


图3 本机



服务器（192.168.43.27）



客户端（192.168.43.40）

图4 不同电脑

3.多线程并发服务器

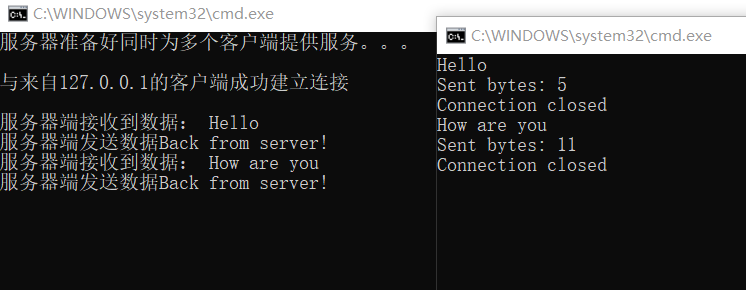
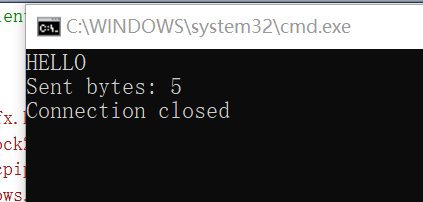


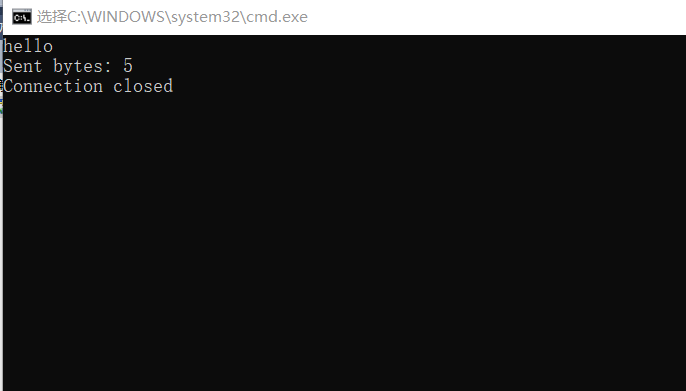
图5 本机



服务器（192.168.43.27）



客户端1（192.168.43.19）



客户端2（192.168.43.40）

图6 不同电脑

**调试中主要存在问题****总结：**

问题1：



图7 问题1截图

解决： 项目->SimpleHelloClient属性页->配置属性->常规->字符集，将“使用Unicode字符集”改为“使用多字节字符集”。

问题2：

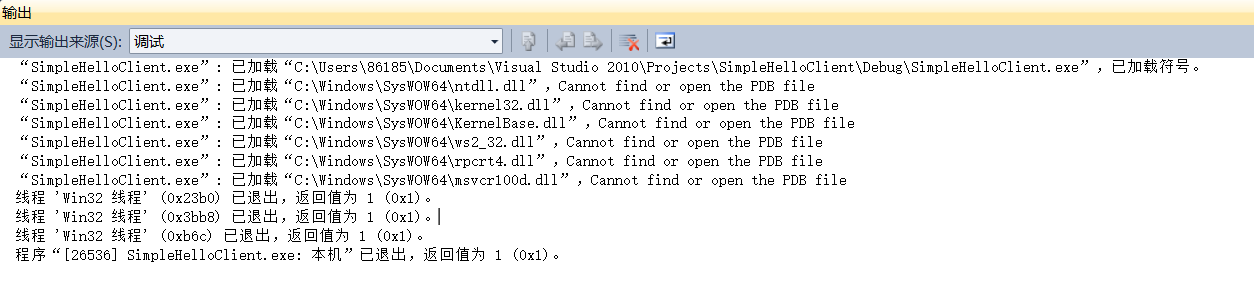


图8 问题2截图

解决：工具->选项->调试->符号,勾选“Microsoft符号服务器”。

问题3：

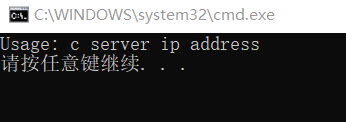


图9 问题3截图

解决：项目->SimpleHelloClient属性页->配置属性->调试,在命令参数中输入127.0.0.1（本机回送地址，用来测试本机进程间通信）。

问题4：

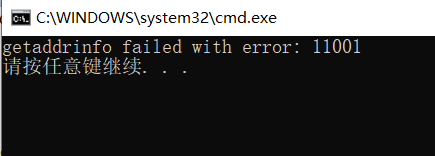


图10 问题4截图

解决：解决方法同问题1。

问题5：注意端口号保持一致！

**代码**

1. 服务器与客户端一对一

1.1服务器：

//SimpleHelloServer.cpp : 这是一个用流式套接字实现的一次性接收来自客户端信息并返回一个信息的服务器端程序。

#include "stdafx.h"

#include <WinSock2.h>

#include <Windows.h>

#include <WS2tcpip.h>

//连接到WinSock 2对应的lib文件：Ws2\_32.lib

#pragma comment (lib,"ws2\_32.lib")

//定义默认的端口号和缓冲区长度

#define SERVER\_PORT "27015"

#define BUFFER\_LEN 512

int \_tmain(int argc, \_TCHAR\* argv[])

{

WSADATA wsaData;

int iResult;

SOCKET ListenSocket=INVALID\_SOCKET;

SOCKET ClientSocket=INVALID\_SOCKET;

struct addrinfo \*result=NULL;

struct addrinfo hints;

int isendResult;

char \*sendbuf="Hello from Server!";

char recvbuf[BUFFER\_LEN];

memset(recvbuf,0,BUFFER\_LEN\*sizeof(char));

//初始化WinSock

iResult=WSAStartup(MAKEWORD(2,2),&wsaData);

if(iResult!=0)

{

printf("WSAStartup failed with error: &d\n",iResult);

return 1;

}

ZeroMemory(&hints,sizeof(hints));

//声明IPv4地址簇，流式套接字，TCP协议

hints.ai\_family=AF\_INET;

hints.ai\_socktype=SOCK\_STREAM;

hints.ai\_protocol=IPPROTO\_TCP;

hints.ai\_flags=AI\_PASSIVE;

//解析服务器地址和端口号

iResult=getaddrinfo(NULL,SERVER\_PORT,&hints,&result);

if(iResult!=0){

printf("getaddrinfo failed with error %d\n",iResult);

WSACleanup();

return 1;

}

//为面向连接的服务器创建套接字

ListenSocket=socket(result->ai\_family,result->ai\_socktype,result->ai\_protocol);

if(ListenSocket==INVALID\_SOCKET){

printf("socket failed with error %d\n",WSAGetLastError());

freeaddrinfo(result);

WSACleanup();

return 1;

}

//为套接字绑定地址和端口号 iResult=bind(ListenSocket,result->ai\_addr,(int)result->ai\_addrlen);

if(iResult==SOCKET\_ERROR){

printf("bind failed with error %d\n",WSAGetLastError());

freeaddrinfo(result);

closesocket(ListenSocket);

WSACleanup();

return 1;

}

freeaddrinfo(result);

//监听连接请求

iResult=listen(ListenSocket,SOMAXCONN);

if(iResult==SOCKET\_ERROR){

printf("listen failed with error %d\n",WSAGetLastError());

freeaddrinfo(result);

closesocket(ListenSocket);

WSACleanup();

return 1;

}

printf("This is server,ready for receive message.\n");

//接受客户端的连接请求，返回连接套接字ClientSocket

ClientSocket=accept(ListenSocket,NULL,NULL);

if(iResult==INVALID\_SOCKET){

printf("accept failed with error %d\n",WSAGetLastError());

freeaddrinfo(result);

closesocket(ListenSocket);

WSACleanup();

return 1;

}

//在必须要监听套接字的情况下释放该套接字

closesocket(ListenSocket);

//持续接收数据直到对方关闭连接

do{

iResult=recv(ClientSocket,recvbuf,30,0);

if(iResult>0){

//情况1：成功接收到数据

printf("Received message from client: %s\n",recvbuf);

//将缓冲区的内容回送给客户端

isendResult=send(ClientSocket,sendbuf,strlen(sendbuf),0);

if(isendResult==SOCKET\_ERROR){

printf("send failed with error %d\n",WSAGetLastError());

closesocket(ClientSocket);

WSACleanup();

return 1;

}

printf("Bytes sent: %d\n",isendResult);

}

else if(iResult==0){

//情况2：连接关闭

printf("Connection closing...\n");

}

else{

//情况3：接收发生错误

printf("recv failed with error:d\n",WSAGetLastError());

closesocket(ClientSocket);

WSACleanup();

return 1;

}

}while(iResult>0);

//关闭套接字，释放资源

closesocket(ListenSocket);

WSACleanup();

return 0;

}

1.2客户端：

// SimpleHelloClient.cpp : 这是一个只向服务器发送一次信息的简单客户端程序。

#include "stdafx.h"

#define WIN32\_LEAN\_AND\_MEAN

#include <Windows.h>

#include <WinSock2.h>

#include <WS2tcpip.h>

#include <stdlib.h>

#include <stdio.h>

#include <WinBase.h>

//连接到WinSock 2对应的lib文件：Ws2\_32.lib,Mswsock.lib,AdvApi32.lib

#pragma comment (lib,"ws2\_32.lib")

#pragma comment (lib,"Mswsock.lib")

#pragma comment (lib,"AdvApi32.lib")

//定义默认的端口号和缓冲区长度

#define SERVER\_PORT "27015"

#define BUFFER\_LEN 512

int \_tmain(int argc, \_TCHAR\*argv[])

{

WSADATA wsaData;

SOCKET ConnectSocket;

struct addrinfo \*result=NULL,\*ptr=NULL,hints;

char \*sendbuf="This is Hello from client.";

char recvbuf[BUFFER\_LEN];

int iResult;

memset(recvbuf,0,BUFFER\_LEN\*sizeof(char));

//验证参数的合法性

if(argc!=2){

printf("Usage: %s server ip address\n",argv[0]);

return 1;

}

//初始化套接字

iResult=WSAStartup(MAKEWORD(2,2),&wsaData);

if(iResult!=0){

printf("WSAStartup failed with error: d%\n",iResult);

return 1;

}

ZeroMemory(&hints,sizeof(hints));

hints.ai\_family=AF\_INET;

hints.ai\_socktype=SOCK\_STREAM;

hints.ai\_protocol=IPPROTO\_TCP;

//解析服务器地址和端口号

iResult=getaddrinfo(argv[1],SERVER\_PORT,&hints,&result); //将输入参数argv[1]中指定的服务器信息写入result

if(iResult!=0){

printf("getaddrinfo failed with error: %d\n",iResult);

WSACleanup();

return 1;

}

//创建套接字

ConnectSocket=socket(result->ai\_family,result->ai\_socktype,result->ai\_protocol); //使用result指定的信息创建套接字

if(ConnectSocket==INVALID\_SOCKET){

printf("socket failed with error: %ld\n",WSAGetLastError());

WSACleanup();

return 1;

}

printf("This is client, ready to send message to server.\n");

//向服务器请求连接

iResult=connect(ConnectSocket,result->ai\_addr,result->ai\_addrlen); //使用套接字ConnectSocket向result中指定的服务器请求连接

if(iResult==SOCKET\_ERROR){

printf("connect failed with error: %ld\n",iResult);

closesocket(ConnectSocket);

WSACleanup();

return 1;

}

freeaddrinfo(result); //释放动态分配的地址信息结构体result

//发送缓冲区中的测试数据

iResult=send(ConnectSocket,sendbuf,strlen(sendbuf),0);

if(iResult==SOCKET\_ERROR){

printf("send failed with error: %ld\n",iResult);

closesocket(ConnectSocket);

WSACleanup();

return 1;

}

printf("Sent bytes: %d\n",iResult);

//持续接收数据，直到服务器关闭连接

do{

iResult=recv(ConnectSocket,recvbuf,sizeof(recvbuf),0);

if(iResult>0){

printf("Received message from server: %s\n",recvbuf);

}

else if(iResult==0){

printf("Connection closed\n");

}

else{

printf("recv failed with error:d\n",WSAGetLastError());

}

}while(iResult>0);

//关闭套接字

closesocket(ConnectSocket);

//释放资源

WSACleanup();

return 0;

}

1. 服务器循环提供服务

2.1服务器

// DuokehuServer.cpp：为多客户提供服务的服务器端程序。

#include "stdafx.h"

#include <WinSock2.h>

#include <Windows.h>

#include <WS2tcpip.h>

#pragma comment (lib,"ws2\_32.lib")

#define SERVER\_PORT "8888"

#define BUFFER\_LEN 512

int \_tmain(int argc, \_TCHAR\* argv[])

{

WSADATA wsaData;

SOCKET ListenSocket=INVALID\_SOCKET;

SOCKET ClientSocket=INVALID\_SOCKET;

struct addrinfo hints,\*result=NULL;

struct sockaddr\_in clientaddr;

char sendbuf[BUFFER\_LEN];

char recvbuf[BUFFER\_LEN];

int iResult,isendResult;

memset(recvbuf,0,BUFFER\_LEN\*sizeof(char));

iResult=WSAStartup(MAKEWORD(2,2),&wsaData);

if(iResult!=0)

{

printf("WSAStartup failed with error: &d\n",iResult);

return 1;

}

ZeroMemory(&hints,sizeof(hints));

hints.ai\_family=AF\_INET;

hints.ai\_socktype=SOCK\_STREAM;

hints.ai\_protocol=IPPROTO\_TCP;

hints.ai\_flags=AI\_PASSIVE;

iResult=getaddrinfo(NULL,SERVER\_PORT,&hints,&result);

if(iResult!=0){

printf("getaddrinfo failed with error %d\n",iResult);

WSACleanup();

return 1;

}

ListenSocket=socket(result->ai\_family,result->ai\_socktype,result->ai\_protocol);

if(ListenSocket==INVALID\_SOCKET){

printf("socket failed with error %d\n",WSAGetLastError());

freeaddrinfo(result);

WSACleanup();

return 1;

}

iResult=bind(ListenSocket,result->ai\_addr,(int)result->ai\_addrlen);

if(iResult==SOCKET\_ERROR){

printf("bind failed with error %d\n",WSAGetLastError());

freeaddrinfo(result);

closesocket(ListenSocket);

WSACleanup();

return 1;

}

freeaddrinfo(result);

iResult=listen(ListenSocket,SOMAXCONN);

if(iResult==SOCKET\_ERROR){

printf("listen failed with error %d\n",WSAGetLastError());

freeaddrinfo(result);

closesocket(ListenSocket);

WSACleanup();

return 1;

}

for(;;){

int addrlenth=sizeof(clientaddr);

ClientSocket=accept(ListenSocket,(sockaddr\*)&clientaddr,&addrlenth);

if(iResult==INVALID\_SOCKET){

printf("accept failed with error %d\n",WSAGetLastError());

closesocket(ListenSocket);

WSACleanup();

return 1;

}

char \*peeraddr= inet\_ntoa(clientaddr.sin\_addr);

do{

iResult=recv(ClientSocket,recvbuf,BUFFER\_LEN,0);

if(iResult>0){

printf("Received message: %s, from client: %s\n",recvbuf,peeraddr);

//显示收到的信息及对方的IP地址

strcpy(sendbuf,"Hello from server!"); //给发送缓冲区sendbuf赋初值

strcat(sendbuf,peeraddr);

//将服务器问候字符串后跟该客户端的IP地址，作为新的sendbuf内容。 isendResult=send(ClientSocket,sendbuf,strlen(sendbuf),0);

if(isendResult==SOCKET\_ERROR){

printf("send failed with error %d\n",WSAGetLastError());

closesocket(ClientSocket);

WSACleanup();

break;

}

printf("Bytes sent: %d\n",isendResult);

}

else if(iResult==0){

printf("Connection closing...\n");

iResult=shutdown(ClientSocket,SD\_SEND);

if(iResult==SOCKET\_ERROR){

printf("shutdown failed with error %d\n",WSAGetLastError());

closesocket(ClientSocket);

WSACleanup();

break;

}

closesocket(ClientSocket);

WSACleanup();

break;

}

else{

printf("recv failed with error:%d\n",WSAGetLastError());

iResult=shutdown(ClientSocket,SD\_SEND);

if(iResult==SOCKET\_ERROR){

printf("shutdown failed with error %d\n",WSAGetLastError());

closesocket(ClientSocket);

WSACleanup();

break;

}

}

}while(iResult>0);

}

closesocket(ListenSocket);

WSACleanup();

return 0;

}

2.2客户端

// ShuruxinxClient.cpp : 客户端程序，用户可以从键盘输入信息并发送给服务器。

//

#include "stdafx.h"

#include <WinSock2.h>

#include <WS2tcpip.h>

#include <Windows.h>

#include <WinBase.h>

#pragma comment (lib,"ws2\_32.lib")

#define SERVER\_PORT "27015"

#define BUFFER\_LEN 512

int \_tmain(int argc, \_TCHAR\* argv[])

{

struct addrinfo \*result=NULL,\*ptr=NULL,hints;

WSADATA wsaData;

SOCKET ConnectSocket;

char sendbuf[BUFFER\_LEN];

char recvbuf[BUFFER\_LEN];

int iResult;

if(argc!=2){

printf("Usage: %s server ip address\n",argv[0]);

return 1;

}

iResult=WSAStartup(MAKEWORD(2,2),&wsaData);

if(iResult!=0){

printf("WSAStartup failed with error: d%\n",iResult);

return 1;

}

ZeroMemory(&hints,sizeof(hints));

hints.ai\_family=AF\_INET;

hints.ai\_socktype=SOCK\_STREAM;

hints.ai\_protocol=IPPROTO\_TCP;

iResult=getaddrinfo(argv[1],SERVER\_PORT,&hints,&result); //将输入参数argv[1]中指定的服务器信息写入result

if(iResult!=0){

printf("getaddrinfo failed with error: %d\n",iResult);

WSACleanup();

return 1;

}

ConnectSocket=socket(result->ai\_family,result->ai\_socktype,result->ai\_protocol);

//使用result指定的信息创建套接字

if(ConnectSocket==INVALID\_SOCKET){

printf("socket failed with error: %ld\n",WSAGetLastError());

WSACleanup();

return 1;

}

iResult=connect(ConnectSocket,result->ai\_addr,result->ai\_addrlen); //使用套接字ConnectSocket向result中指定的服务器请求连接

if(iResult==SOCKET\_ERROR){

printf("connect failed with error: %ld\n",iResult);

closesocket(ConnectSocket);

WSACleanup();

return 1;

}

freeaddrinfo(result); //释放动态分配的地址信息结构体result

while(gets(sendbuf)!=NULL){ //从键盘获取输入字符串

if(\*sendbuf=='Q'){

closesocket(ConnectSocket);

return 0;

}

iResult=send(ConnectSocket,sendbuf,strlen(sendbuf),0);

if(iResult==SOCKET\_ERROR){

printf("send failed with error: %ld\n",iResult);

closesocket(ConnectSocket);

WSACleanup();

return 1;

}

printf("Sent bytes: %d\n",iResult);

do{

memset(recvbuf,0,BUFFER\_LEN\*sizeof(char));

iResult=recv(ConnectSocket,recvbuf,strlen(recvbuf),0);

if(iResult>0){

printf("Received message from server: %s\n",recvbuf);

}

else if(iResult==0){

printf("Connection closed\n");

}

else{

printf("recv failed with error:d\n",WSAGetLastError());

}

}while(iResult>0);

}

closesocket(ConnectSocket);

WSACleanup();

return 0;

}

1. 多线程并发服务器

3.1服务器

// DuoxianchengServer.cpp : 采用多线程机制实现并发为多个客户端服务。

//

#include "stdafx.h"

#include "DuoxianchengServer.h"

#ifdef \_DEBUG

#define new DEBUG\_NEW

#endif

#define SERVER\_PORT "27015"

#define BUFFER\_LEN 512

#include <WinSock2.h>

#include <Windows.h>

#include <WS2tcpip.h>

#pragma comment (lib,"ws2\_32.lib")

// 唯一的应用程序对象

CWinApp theApp;

using namespace std;

UINT duoxiancheng\_server\_fun( LPVOID pParam );

int \_tmain(int argc, TCHAR\* argv[], TCHAR\* envp[])

{

int nRetCode = 0;

int iResult = 0;

SOCKET ListenSocket, ConnectSocket;

CWinThread \*pThread= NULL;

WSADATA wsaData;

struct addrinfo hints,\*result=NULL;

struct sockaddr\_in clientaddr;

HMODULE hModule = ::GetModuleHandle(NULL);

if (hModule != NULL)

{

// 初始化 MFC 并在失败时显示错误

if (!AfxWinInit(hModule, NULL, ::GetCommandLine(), 0))

{

// TODO: 更改错误代码以符合您的需要

\_tprintf(\_T("错误: MFC 初始化失败\n"));

nRetCode = 1;

}

else

{

// // TODO: 在此处为应用程序的行为编写代码。

// //Windows Sockets Dll初始化

iResult=WSAStartup(MAKEWORD(2,2),&wsaData);

if(iResult!=0){

printf("WSAStartup failed with error %d\n",iResult);

return 1;

}

ZeroMemory(&hints,sizeof(hints));

hints.ai\_family=AF\_INET;

hints.ai\_socktype=SOCK\_STREAM;

hints.ai\_protocol=IPPROTO\_TCP;

hints.ai\_flags=AI\_PASSIVE;

iResult=getaddrinfo(NULL,SERVER\_PORT,&hints,&result);

if(iResult!=0){

printf("getaddrinfo failed with error %d\n",iResult);

closesocket(ListenSocket);

WSACleanup();

return 1;

}

ListenSocket= socket(result->ai\_family,result->ai\_socktype,result->ai\_protocol);

if(ListenSocket==INVALID\_SOCKET){

printf("socket failed with error %ld\n",GetLastError());

WSACleanup();

return 1;

}

iResult=bind(ListenSocket,result->ai\_addr,(int)result->ai\_addrlen);

if(iResult==SOCKET\_ERROR){

printf("bind failed with error %d\n",WSAGetLastError());

closesocket(ListenSocket);

WSACleanup();

freeaddrinfo(result);

return 1;

}

freeaddrinfo(result);

iResult=listen(ListenSocket,SOMAXCONN);

if(iResult==SOCKET\_ERROR){

printf("listen failed with error %d\n",WSAGetLastError());

closesocket(ListenSocket);

WSACleanup();

return 1;

}

printf("服务器准备好同时为多个客户端提供服务。。。\n");

//开始

for(;;){

int addrlenth=sizeof(clientaddr);

ConnectSocket=accept(ListenSocket,(sockaddr\*)&clientaddr,&addrlenth);

if(ConnectSocket==INVALID\_SOCKET){

printf("accept failed with error %d\n",WSAGetLastError());

closesocket(ListenSocket);

WSACleanup();

return -1;

}

char \*peeraddr= inet\_ntoa(clientaddr.sin\_addr);

//建立连接成功

printf("\r\n与来自%s的客户端成功建立连接\n\n",peeraddr);

//启动回射线程

pThread = AfxBeginThread( duoxiancheng\_server\_fun, &ConnectSocket );

}

printf("accept failed with error %d\n",WSAGetLastError());

closesocket(ListenSocket);

WSACleanup();

return -1;

}

}

else

{

// TODO: 更改错误代码以符合您的需要

\_tprintf(\_T("错误: GetModuleHandle 失败\n"));

nRetCode = 1;

}

return nRetCode;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名：duoxiancheng\_server\_fun

输入参数：SOCKET s:服务器的连接套接字

输出参数：0：成功，-1：失败

功能：与客户端的具体通信功能实现函数\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

UINT duoxiancheng\_server\_fun( LPVOID pParam )

{

int iSendResult=0;

char sendbuf[BUFFER\_LEN];

char recvbuf[BUFFER\_LEN];

int iResult=0;

int err=0;

//将输入参数转换为连接套接字

SOCKET s =\*( (SOCKET \*)pParam);

do {

memset( recvbuf, 0, BUFFER\_LEN );

//接收数据

iResult = recv( s, recvbuf, BUFFER\_LEN, 0);

if (iResult > 0){

printf("服务器端接收到数据： %s\n", recvbuf);

//向客户端发送数据

memset( sendbuf, 0, BUFFER\_LEN );

strcat(sendbuf,"Back from server!");

iResult = send( s,sendbuf,BUFFER\_LEN, 0 );

if(iResult == SOCKET\_ERROR)

{

printf("send 函数调用错误，错误号:: %ld\n", WSAGetLastError());

err = closesocket(s);

if (err == SOCKET\_ERROR){

printf("closesocket函数调用错误，错误号：%d\n", WSAGetLastError());

}

iResult = -1;

}

else

printf("服务器端发送数据Y%s\n", sendbuf);

}

else{

if (iResult == 0)

printf("对方连接关闭，退出\n");

else{

printf("recv 函数调用错误，错误号: %d\n", WSAGetLastError());

iResult = -1;

}

err = closesocket(s);

if (err == SOCKET\_ERROR){

printf("closesocket 函数调用错误，错误号：%d\n", WSAGetLastError());

}

break;

}

} while (iResult > 0);

return iResult;

}

3.2客户端

代码同2.2客户端

【实验总结】

1.“使用Unicode字符集”改为“使用多字节字符集”。

2.勾选“Microsoft符号服务器”。

3. 端口号保持一致

4.注意本机进程通信与不同主机通信传参不同

5.仅实现服务器循环为一个客户端提供服务，并没有实现循环为多个客户端提供服务。

6.编写多线程并发服务器代码，由于没能解决调试时的错误，所以没有使用含csocketframe类的代码，而是用了老师提供的另一个代码。

**实验3 流式套接字应用程序运行分析**

【实验要求】

1、服务器运行异常现象分析

模拟服务器端各种运行异常情況，用 Wireshark抓包，分析对应的现象并提出解决建议。

场景一：服务器进程崩溃

场景二：服务器宿主机崩溃或网络紊乱

场景三：服务器宿主机崩溃后重启

2、流式套接字传输效率分析

测试一：对于固定长度的数据，分析发送次数对发送效率的影响

测试二、测试接收绶冲区大小与数据传送时间的关系

【实验过程】

**1.正常客户端和服务器通信**

通过“netstat -p tcp”命令观察客户端和服务器主机中已经建立的TCP连接状态，

启动服务器，服务器程序可以是之前流式套接字实现的服务器。使用“netstat -a ”观察服务器的连接状态，系统中的27015端口已经处于监听状态。



图 1 通过netstat命令观察到服务器启动后tcp的状态

启动wireshark，设置过滤条件“TCP”，开始捕获网络中的TCP数据。此处省略Wireshark抓包截图。

启动客户端，并在连接建立成功后，使用“netstat -a”命令观察当前主机中的TCP连接状态，系统中的27015端口已经增加了一个ESTABLISHED状态，同时还保留LISTENING状态的记录。如图2所示。

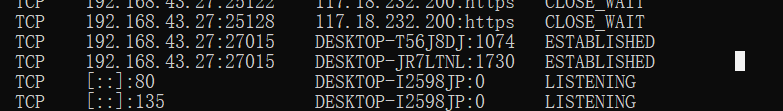


图2 客户端与服务器建立连接后的TCP在7210端口上的连接状态

客户端向服务器端发送信息，运行效果如图3、4所示。

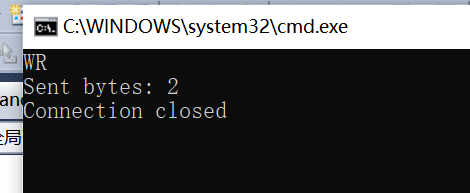


图3 客户端

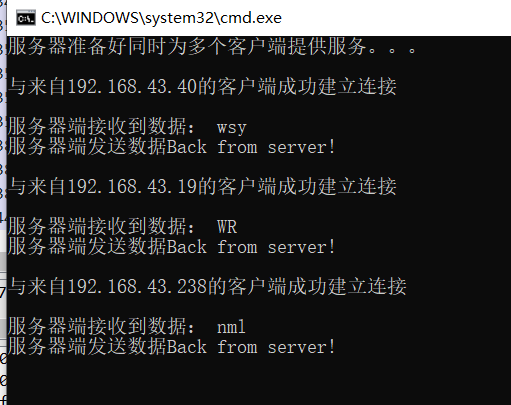


图4 服务器端

停止Wireshark抓包，观察回射过程的通信过程。如图5所示。

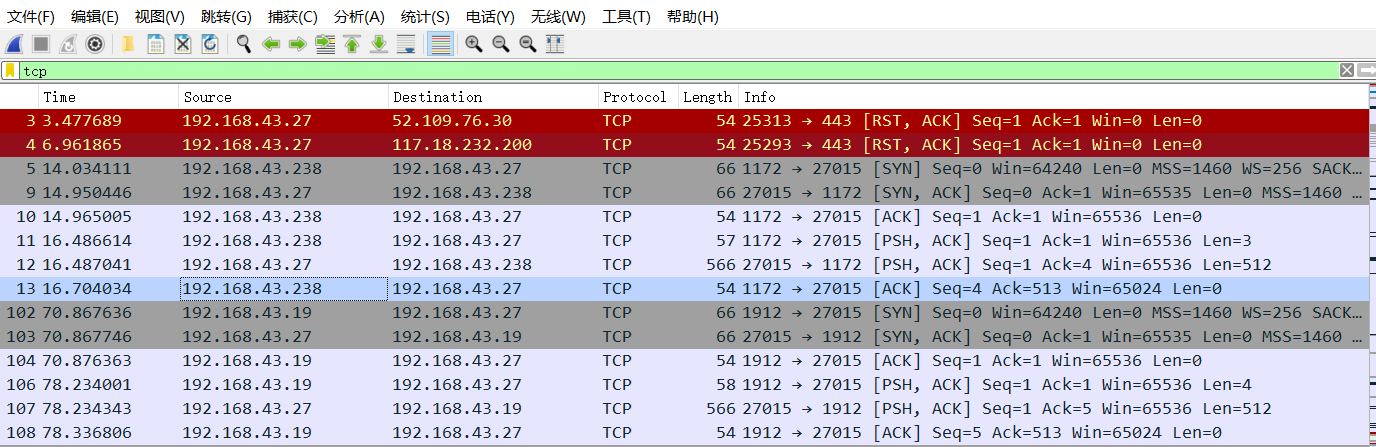
****

图 5 应用程序进程间通信数据包捕获情况

**2.服务器进程崩溃**

2.1正常通信，同实验过程1。

2.2开始wireshark抓包。

服务器进程崩溃，客户端出现错误，如图6所示。

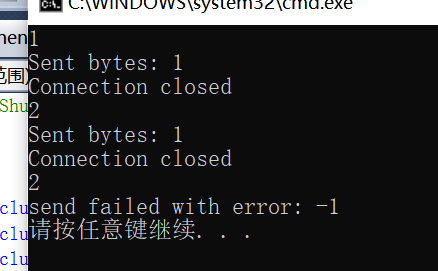


图6 客户端

2.3关闭服务器进程。

在服务器电脑上通过Alt+Ctrl+Del复合键启动任务管理器，在任务管理器中结束服务器进程。如图7所示。



图7 关闭服务器进程

2.4客户端停止抓包，观察到继续回射时，服务器返回RST包。

由于服务器端异常终止，服务器端的TCP协议在收到来自客户端的数据时，返回RST数据包，以通知客户端，服务器出现了异常关闭的情况。客户端可以通过获取出错代码来退出连接。如图8所示。

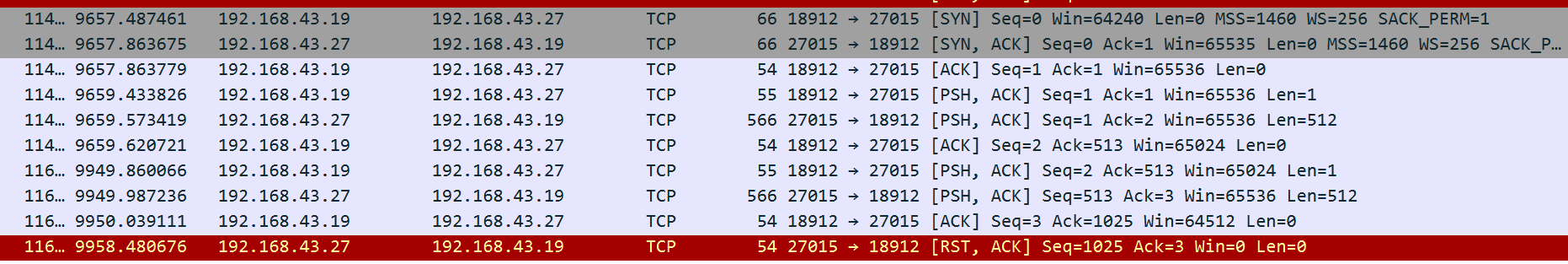
****

图8 服务器崩溃后客户端收到RST包

**3.主机崩溃或网络紊乱**

3.1正常通信，同实验过程1。

3.2断掉服务器网线，或禁用网卡。

通过禁用网卡的方法模拟主机崩溃或网络紊乱的情况。如图9所示。

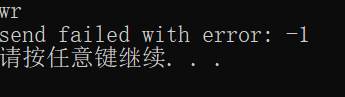


图9a 客户端

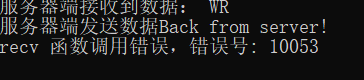


图9b 服务器端

3.3停止抓包，客户端尝试多次重传无果后退出。

如图10所示，在服务器端的协议无法工作时，只能捕获到客户端发往服务器的协议数据包，没有服务器端的返回数据包。客户端在多次尝试发送后，报错并退出连接。

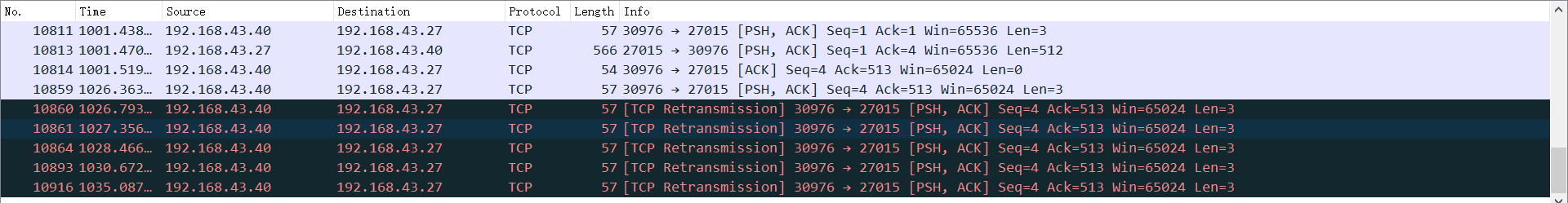


图10 客户端多次重传尝试

**4．主机崩溃后重启**

4.1正常通信，同实验过程1。

4.2服务器断开后重启。

服务器断开后， 如图11所示。

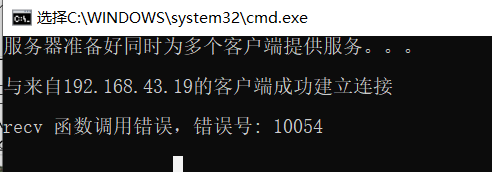


图11 服务器断开

4.3停止抓包，服务器重启，给客户端返回RST包

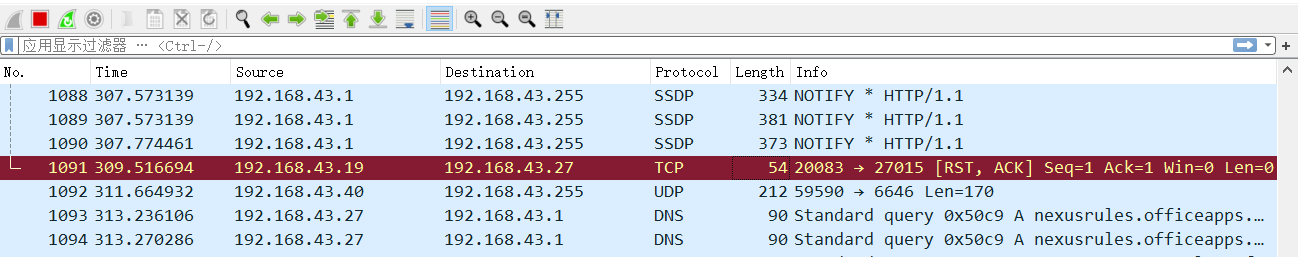


图12a

如图12a所示，服务器宿主机崩溃后，客户端多次尝试重发数据，没有响应，后超时退出。故也可能收不到RST数据包。

服务器主机重启后，客户端需要重新建立与服务器的连接。如图12b所示。

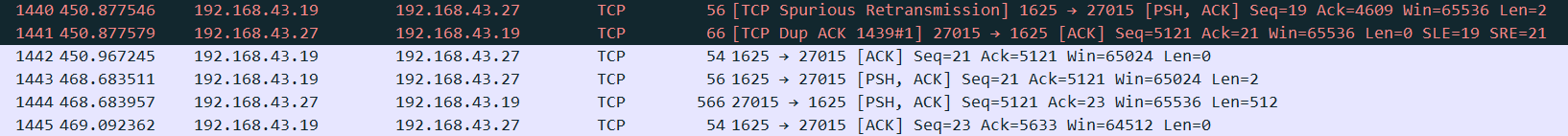


图12b 服务器重启后客户端重新与其建立连接

**5.对于固定长度n,分析发送次数对发送效率的影响**

实验设置总发送长度为10000字节，选择单次发送长度分别为10000字节、1000字节、100字节、10字节和1字节。发送次数越多，系统调用协议的次数越多，产生的系统消耗也就越大。如图13所示。

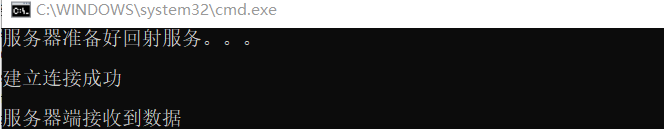
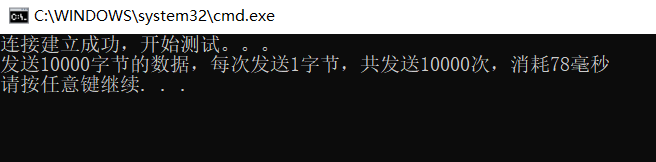
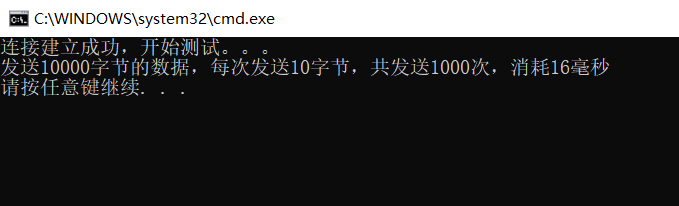
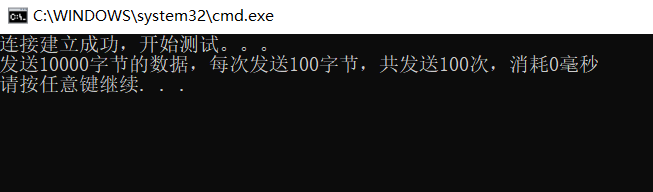
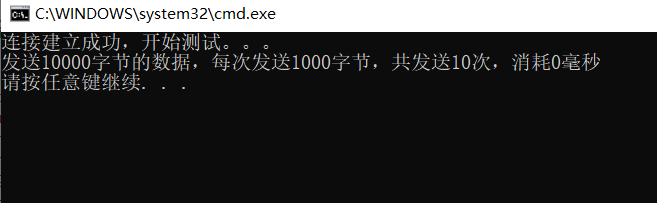


图13a 服务器









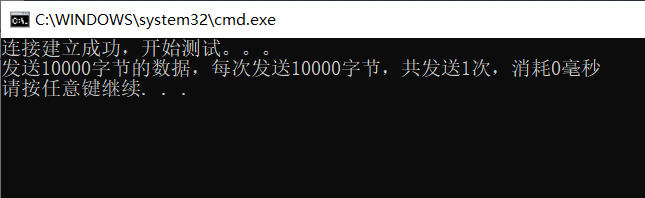


图13b 客户端

**6.测试接收缓冲区与数据传送时间的关系。**

6.1服务器端模拟一个慢速设备的处理。

在应用程序接收缓冲仅设置100字节的空间，对客户端发来的数据按照100字节为单位进行接收，且每次接收间隔100毫秒。

6.2传参数测试

两次测试时，客户端均按照每次发送10000字节，共发送10次。第一次设置服务器接收缓冲区长度为1000000字节，客户端发送时间为62毫秒。第二次设置服务器缓冲区为100字节，客户端发送时间为52625毫秒。如图14所示。

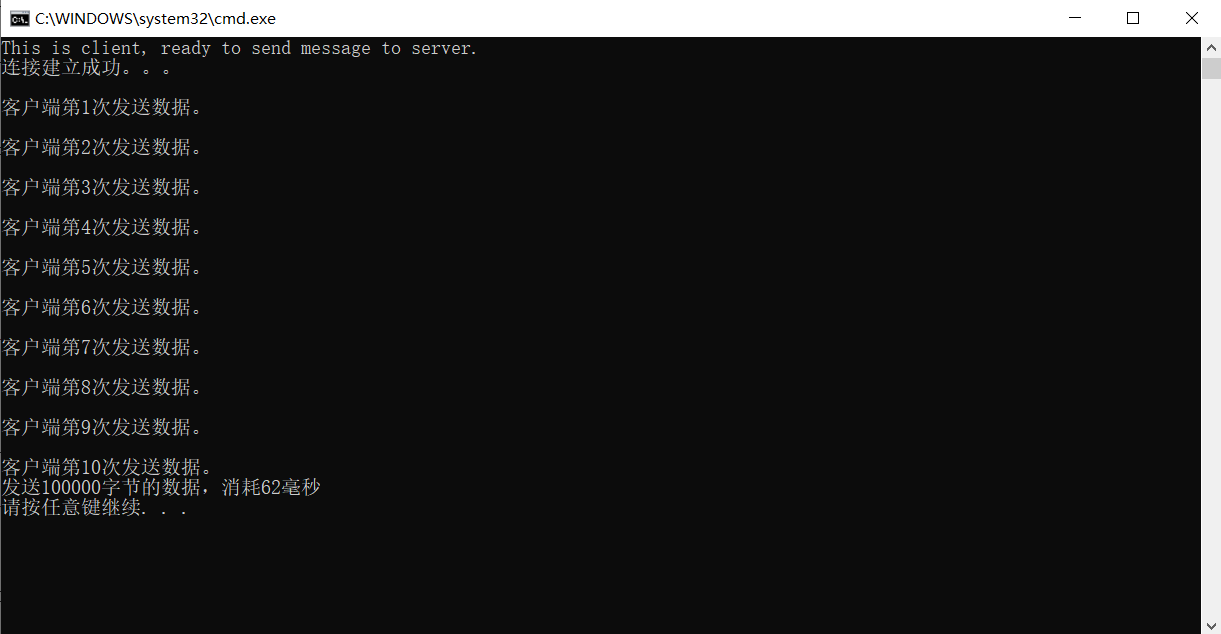
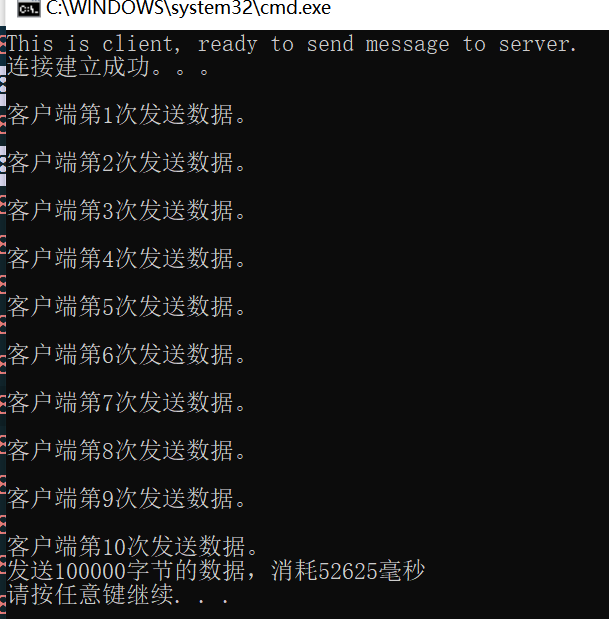
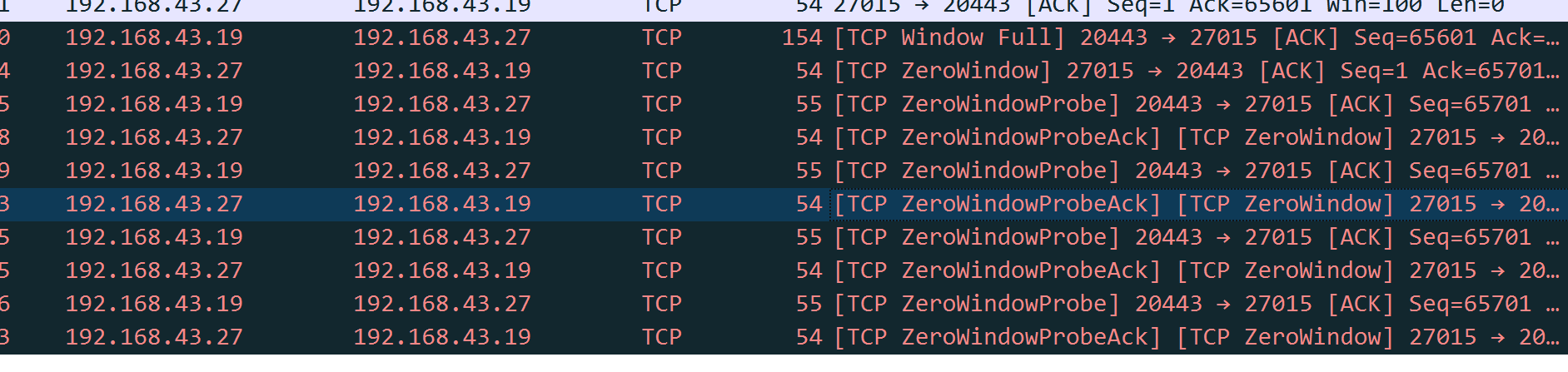
 

图14a服务器缓冲区为1000000 图14b 服务器缓冲区为100

对于客户端发送的总字节数<服务器的系统接收缓存，消耗时间很少；对于客户端发送数据的总字节数>服务器系统接收缓存，消耗的时间很长。

6.3 Wireshark抓包

由于服务器处理速度慢，造成太小的接收缓存不足以存储客户端发来的大量数据，从而限制了发送速度。通过Wireshark抓包观察到，第二次测试网络中出现了大量的接收窗口询问数据包[TCP zerowindowProbe]和[TCP zerowindowProbeACK]。如图15所示。

图15

TCP报文中的window size表示发出这个报文的一端准备多少bytes的数据，当TCP的一端一直接收数据，但是应用层没有及时读取的话，数据一直在TCP模块中缓存，最终受限于接收缓存的大小，window size会变为0，此时我们称呼这个接收窗口为零窗(zero window)，对端也不能再发送更多的数据。

**代码**

**测试一**

**服务器端**

// EchoTCPServer.cpp : 定义控制台应用程序的入口点。

#include "stdafx.h"

#include "CSocketFrame.h"

#include "winsock2.h"

#define ECHOPORT "27015"

int tcp\_server\_fun\_echo( SOCKET s );

int main(int argc, char\* argv[])

{

CSocketFrame frame;

int iResult = 0;

SOCKET ListenSocket, ConnectSocket;

//输入参数合法性检查

if (argc != 1)

{

printf("usage: EchoTCPServer-sendn");

return -1;

}

//Windows Sockets Dll初始化

frame.start\_up();

//创建服务器端的流式套接字并在指定端口号上监听

ListenSocket = frame.tcp\_server( NULL, ECHOPORT );

if ( ListenSocket == -1 )

return -1;

printf("服务器准备好回射服务。。。\n");

for ( ; ; ) {

if((ConnectSocket = accept( ListenSocket, NULL, NULL )) != INVALID\_SOCKET ){

//建立连接成功

printf("\r\n建立连接成功\n\n");

//回射

iResult = tcp\_server\_fun\_echo( ConnectSocket );

//如果出错，关闭当前连接套接字，继续接收其它客户端的请求

if(iResult == -1)

printf("当前连接已关闭或出错!\n");

}

else{

printf("accept 函数调用错误，错误号： %d\n", WSAGetLastError());

frame.quit( ListenSocket );

return -1;

}

//关闭连接套接字

if ( closesocket( ConnectSocket ) == SOCKET\_ERROR)

printf("closesocket 函数调用错误，错误号：%d\n", WSAGetLastError());;

}

frame.quit( ListenSocket );

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名：tcp\_server\_fun

输入参数：SOCKET s:服务器的连接套接字

输出参数：0：成功，-1：失败

功能：tcp\_server的具体功能函数

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int tcp\_server\_fun\_echo( SOCKET s )

{

int iResult = 0;

char recvline[MAXLINE];

do {

memset( recvline, 0, MAXLINE );

//接收数据

iResult = recv( s, recvline, MAXLINE, 0);

if (iResult > 0){

printf("服务器端接收到数据%s\n", recvline);

//回射发送已收到的数据

iResult = send( s,recvline,iResult, 0 );

if(iResult == SOCKET\_ERROR)

{

printf("send 函数调用错误，错误号: %ld\n", WSAGetLastError());

iResult = -1;

}

else

printf("服务器端发送数据%s\n", recvline);

}

else{

if (iResult == 0)

printf("对方连接关闭，退出\n");

else{

printf("recv 函数调用错误，错误号: %d\n", WSAGetLastError());

iResult = -1;

}

break;

}

} while (iResult > 0);

return iResult;

}

**客户端**

// EchoTCPClient.cpp.cpp : 定义控制台应用程序的入口点。

#include "stdafx.h"

#include <time.h>

#include "CSocketFrame.h"

int tcp\_client\_fun\_sendn(SOCKET s ,int totallen ,int singlelen);

#define ECHOPORT "27015"

int main(int argc, char\* argv[])

{

CSocketFrame frame;

int iResult;

SOCKET ClientSocket;

int totallen,singlelen;

//输入参数合法性检查

if (argc != 4)

{

printf("usage: EchoTCPClient-sendn <IPaddress> <TotalLen> <SingleLen>");

return -1;

}

//Windows Sockets Dll初始化

frame.start\_up();

//创建客户端的流式套接字，并与服务器建立连接

printf("连接建立成功，开始测试。。。\n");

ClientSocket = frame.tcp\_client( ( char \*)argv[1], ECHOPORT );

if ( ClientSocket == -1 )

return -1;

totallen = atoi(argv[2]);

singlelen =atoi(argv[3]);

//开始测试

iResult = tcp\_client\_fun\_sendn(ClientSocket ,totallen,singlelen);

frame.quit( ClientSocket );

return iResult;

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名：tcp\_client\_fun\_sendn

输入参数：

SOCKET s:服务器的连接套接字

int totallen:总共发送的字节数

int singlelen:一次send调用发送的字节数

输出参数：0：成功，-1：失败

功能：tcp\_client测试客户端的具体功能函数

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int tcp\_client\_fun\_sendn(SOCKET s ,int totallen ,int singlelen)

{

int iResult, leftlen, times=0;

char sendline[MAXLINE],recvline[MAXLINE];

memset(sendline,0,MAXLINE);

memset(recvline,0,MAXLINE);

char \*sendbuf;

DWORD beginticks,endticks;

if(totallen<1 || singlelen<1 || singlelen > totallen)

return -1;

//根据totallen为发送缓存申请空间并为发送缓存赋值

sendbuf = (char \*)malloc(totallen);

memset(sendbuf,0,totallen);

for(int i=0; i<totallen; i++)

sendbuf[i]=i;

char \*ptr = sendbuf;

leftlen=totallen;

//获取当前时间

beginticks = GetTickCount();

//循环发送数据

while(leftlen > 0)

{

iResult = send(s,ptr,singlelen,0);

if(iResult == SOCKET\_ERROR)

{

printf("send 函数调用错误，错误号： %ld\n", WSAGetLastError());

delete(sendbuf);

return -1;

}

ptr +=iResult;

leftlen -=iResult;

times++;

}

endticks =GetTickCount();

printf("发送%d字节的数据，每次发送%d字节，共发送%d次，消耗%d毫秒\r\n",totallen,singlelen,times,endticks-beginticks );

free(sendbuf);

return iResult;

}

**测试二**

**服务器端**

// EchoTCPServer2.cpp : 定义控制台应用程序的入口点。

#include "stdafx.h"

#include "CSocketFrame.h"

#include "winsock2.h"

#define ECHOPORT "27015"

#define BUFLEN 100

int tcp\_server\_fun\_testrecvbuf( SOCKET s );

int main(int argc, char\* argv[])

{

CSocketFrame frame;

int iResult = 0;

int buflen;

SOCKET ListenSocket, ConnectSocket;

//输入参数合法性检查

if (argc != 2)

{

printf("usage: EchoTCPServer-testrecvbuf <RecvBufLen>");

return -1;

}

buflen =atoi( argv[1]);

//Windows Sockets Dll初始化

frame.start\_up();

//创建服务器端的流式套接字并在指定端口号上监听

ListenSocket = frame.tcp\_server( NULL, ECHOPORT );

if ( ListenSocket == -1 )

return -1;

//设置服务器接收缓存

iResult = setsockopt( ListenSocket, SOL\_SOCKET, SO\_RCVBUF, ( char \* )&buflen, sizeof( buflen ));

if ( iResult == SOCKET\_ERROR){

printf("setsockopt function failed with error %d\n", WSAGetLastError());

frame.clean\_up();

return -1;

}

printf("服务器准备好测试。。。\n");

for ( ; ; ) {

ConnectSocket = accept( ListenSocket, NULL, NULL );

if(ConnectSocket != INVALID\_SOCKET ){

//建立连接成功

printf("\r\n建立连接成功\n\n");

//回射

iResult = tcp\_server\_fun\_testrecvbuf( ConnectSocket );

//如果出错，关闭当前连接套接字，继续接收其它客户端的请求

if(iResult == -1)

printf("当前连接已关闭或出错!\n");

}

else{

printf("accept 函数调用错误，错误号： %d\n", WSAGetLastError());

frame.quit( ListenSocket );

return -1;

}

//关闭连接套接字

if ( closesocket( ConnectSocket ) == SOCKET\_ERROR)

printf("closesocket 函数调用错误，错误号：%d\n", WSAGetLastError());;

}

frame.quit( ListenSocket );

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名：tcp\_server\_fun\_testrecvbuf

输入参数：SOCKET s:服务器的连接套接字

输出参数：0：成功，-1：失败

功能：tcp\_server服务器的具体功能函数

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int tcp\_server\_fun\_testrecvbuf( SOCKET s )

{

int iResult = 0;

char recvline[BUFLEN];

do {

memset( recvline, 0, BUFLEN );

//接收数据

iResult = recv( s, recvline, BUFLEN, 0);

if (iResult > 0)

printf("服务器端接收到数据%d字节\n", iResult);

else{

if (iResult == 0)

printf("对方连接关闭，退出\n");

else{

printf("recv 函数调用错误，错误号: %d\n", WSAGetLastError());

iResult = -1;

}

break;

}

Sleep(100);

} while (iResult > 0);

return iResult;

}

**客户端**

// EchoTCPClient2.cpp : 定义控制台应用程序的入口点。

#include "stdafx.h"

#include<string>

using namespace std;

#include "CSocketFrame.h"

int tcp\_client\_fun\_echo(FILE \*fp,SOCKET s);

#define ECHOPORT "27015"

#define BUFLEN 10000

int tcp\_client\_fun\_testrecvbuf(SOCKET s, int times);

int main(int argc, char\* argv[])

{

CSocketFrame frame;

int iResult;

int times;

SOCKET ClientSocket;

//输入参数合法性检查

if (argc != 3)

{

printf("usage: EchoTCPClient-testrecvbuf <IPaddress> <Times>");

return -1;

}

times = atoi(argv[2]);

//Windows Sockets Dll初始化

frame.start\_up();

//创建客户端的流式套接字，并与服务器建立连接

printf("连接建立成功。。。\n");

ClientSocket = frame.tcp\_client( ( char \*)argv[1], ECHOPORT );

if ( ClientSocket == -1 )

return -1;

//开始回射请求的发送与接收

iResult = tcp\_client\_fun\_testrecvbuf(ClientSocket,times);

frame.quit( ClientSocket );

return iResult;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名：tcp\_client\_fun\_testrecvbuf

输入参数：

SOCKET s:服务器的连接套接字

int times:发送次数，每次10000字节

输出参数：0：成功，-1：失败

功能：tcp\_client：根据用户指定的发送次数循环发送数据给服务器

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int tcp\_client\_fun\_testrecvbuf(SOCKET s, int times)

{

int iResult;

char sendline[BUFLEN],recvline[BUFLEN];

DWORD beginticks,endticks;

memset(sendline,0,BUFLEN);

memset(recvline,0,BUFLEN);

for(int i=0; i< BUFLEN; i++)

sendline[i]=i;

//获取当前时间

beginticks = GetTickCount();

//循环发送

for(int i=0;i<times;i++)

{

iResult = send(s,sendline,BUFLEN,0);

if(iResult == SOCKET\_ERROR)

{

printf("send 函数调用错误，错误号： %ld\n", WSAGetLastError());

return -1;

}

printf("\r\n客户端第%d次发送数据。\r\n", i+1);

}

endticks =GetTickCount();

printf("发送%d字节的数据，消耗%d毫秒 \r\n",times\*BUFLEN,endticks-beginticks );

return iResult;

}

【实验总结】

添加CSocketFrame框架时，出现了很多错误。一些错误通过修改、添加头文件可以解决。需要注意的是，在CSocketFrame.h文件中，需要注释掉构造函数和析构函数。此外，还可以尝试不使用类框架完成。

**实验4 数据报套接字通信编程**

【实验目的】

1、用数据报套接字实现客户端与服务器之间的通信

要求服务器可以循环响应多个客户端的请求；

客户端需要从键盘输入要发送的信息

2、实现用数据报套接字发送广播信息

参考《网络程序设计》实验指导书中实验4.2的内容说明

实现在服务器端收到来自一个客户端的信息后，将信息广播给所有的在线客户端

客户端要能够有效接收并显示出收到的广播信息。

【实验过程】

代码：

服务器：

// UDPCommunicateServer.cpp.cpp : 定义控制台应用程序的入口点。

//

#include "stdafx.h"

#include "winsock2.h"

#include "windows.h"

#include "WS2tcpip.h"

#include "stdio.h"

#include "stdlib.h"

#define DEFAULT\_BUFFLEN 512

#define SERVER\_PORT "27015"

#pragma comment (lib,"Ws2\_32.lib")

int \_tmain(int argc, \_TCHAR\* argv[])

{

WSADATA wsaData;

int iSendResult,iResult = 0;

SOCKET ServerSocket=INVALID\_SOCKET;

struct addrinfo hints,\*result=NULL;

char recvbuf[DEFAULT\_BUFFLEN]="0";

int recvbuflen=DEFAULT\_BUFFLEN;

sockaddr\_in clientaddr;

int clientlen=sizeof(sockaddr\_in);

//Windows Sockets Dll初始化

iResult=WSAStartup(MAKEWORD(2,2),&wsaData);

if(iResult!=0)

{

printf("WSAStartup failed with error:%d\n",iResult);

return 1;

}

ZeroMemory(&hints,sizeof(hints));

hints.ai\_family=AF\_INET;

hints.ai\_socktype=SOCK\_DGRAM;

hints.ai\_protocol=IPPROTO\_UDP;

hints.ai\_flags=AI\_PASSIVE;

iResult=getaddrinfo(NULL,SERVER\_PORT,&hints,&result);

if(iResult!=0)

{

printf("getaddrinfo failed with error:%d\n",iResult);

WSACleanup();

return 1;

}

ServerSocket = socket(result->ai\_family,result->ai\_socktype,result->ai\_protocol );

if ( ServerSocket == INVALID\_SOCKET)

{

printf("socket failed with error:%d\n",WSAGetLastError());

freeaddrinfo(result);

WSACleanup();

return 1;

}

iResult=bind(ServerSocket,result->ai\_addr,(int)result->ai\_addrlen);

if(iResult==SOCKET\_ERROR)

{

printf("Bind failed with error:%d\n",WSAGetLastError());

freeaddrinfo(result);

closesocket(ServerSocket);

WSACleanup();

return 1;

}

freeaddrinfo(result);

printf("服务器准备好回射服务。。。\n");

for ( ; ; ) {

ZeroMemory(&clientaddr,sizeof(clientaddr));

iResult=recvfrom( ServerSocket,recvbuf,recvbuflen,0,(SOCKADDR\*)&clientaddr,&clientlen);

if(iResult >0)

{

printf("服务器端接收到数据:%s\n",recvbuf);

iSendResult=sendto(ServerSocket,recvbuf,iResult,0,(SOCKADDR\*)&clientaddr,clientlen);

if(iSendResult==SOCKET\_ERROR)

{

printf("Bind failed with error:%d\n",WSAGetLastError());

closesocket(ServerSocket);

WSACleanup();

return 1;

}

printf("服务器端发送数据:%s\n",recvbuf);

}

else if (iResult==0)

{

printf("Connection closing...\n");

}

else

{

printf("receive failed with error:%d\n",WSAGetLastError());

closesocket(ServerSocket);

WSACleanup();

return 1;

}

}

closesocket( ServerSocket );

WSACleanup();

return 0;

}

客户端：

// UDPCommunicateClient.cpp

//

#include "stdafx.h"

#include "winsock2.h"

#include "windows.h"

#include "WS2tcpip.h"

#include "stdio.h"

#include "stdlib.h"

#define DEFAULT\_BUFFLEN 512

#define SERVER\_PORT "27015"

#pragma comment (lib,"Ws2\_32.lib")

int \_tmain(int argc, CHAR\* argv[])

{

WSADATA wsaData;

int iResult = 0;

SOCKET ConnectLessSocket=INVALID\_SOCKET;

struct addrinfo hints,\*result=NULL;

char sendbuf[DEFAULT\_BUFFLEN];

int sendbuflen = DEFAULT\_BUFFLEN;

char recvbuf[DEFAULT\_BUFFLEN]="0";

int recvbuflen=DEFAULT\_BUFFLEN;

if(argc!=2){

printf("Usage:%s server-name\n",argv[0]);

return 1;

}

iResult=WSAStartup(MAKEWORD(2,2),&wsaData);

if(iResult!=0)

{

printf("WSAStartup failed with error:%d\n",iResult);

return 1;

}

ZeroMemory(&hints,sizeof(hints));

hints.ai\_family=AF\_INET;

hints.ai\_socktype=SOCK\_DGRAM;

hints.ai\_protocol=IPPROTO\_UDP;

hints.ai\_flags=AI\_PASSIVE;

iResult=getaddrinfo(argv[1],SERVER\_PORT,&hints,&result);

if(iResult!=0)

{

printf("getaddrinfo failed with error:%d\n",iResult);

WSACleanup();

return 1;

}

ConnectLessSocket = socket(result->ai\_family,result->ai\_socktype,result->ai\_protocol );

if ( ConnectLessSocket == INVALID\_SOCKET)

{

printf("socket failed with error:%d\n",WSAGetLastError());

freeaddrinfo(result);

WSACleanup();

return 1;

}

printf("客户端启动成功，请输入回射字符串。。。\n");

while(gets(sendbuf)!= NULL)

{

if(\*sendbuf=='Q'){

closesocket(ConnectLessSocket);

WSACleanup();

return 0;

}

iResult=sendto(ConnectLessSocket,sendbuf,(int)strlen(sendbuf),0,result->ai\_addr,(int)result->ai\_addrlen);

if(iResult==SOCKET\_ERROR){

printf("sendto failed with error:%d\n",WSAGetLastError());

closesocket(ConnectLessSocket);

freeaddrinfo(result);

WSACleanup();

return 1;

}

printf("客户端发送数据：%s\n",sendbuf);

iResult=recvfrom(ConnectLessSocket,recvbuf,recvbuflen,0,NULL,NULL);

if(iResult>0)

{

printf("客户端接收到数据：%s\n",recvbuf);

}else if(iResult==0)

{

printf("connection closed.\n");

}

else

{

printf("Recvfrom failed with error: %d\n",WSAGetLastError());

}

}

closesocket( ConnectLessSocket );

WSACleanup();

return 0;

}

结果截图：

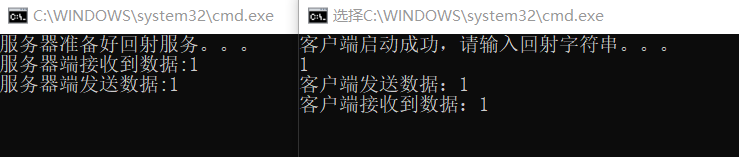


图1 运行结果

出现的问题：

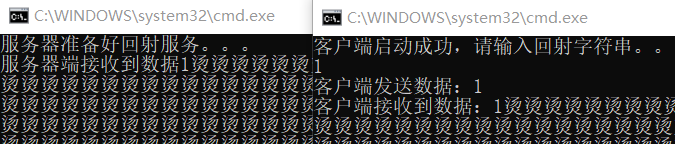


图2 缓冲区未清空的运行结果

问题分析：

在打印数组的时候“越界”导致的，因为程序在默认初始化char类型的数组时，初始化的值是“烫”字，一般情况下是字符串未初始化再加上字符串中的结尾标识符‘\0’在调用函数操作的过程搞丢了，导致程序无法判断数组是否读完，然后直接输出数组的全部成员（包括系统默认的“烫”字），所以才会出现大量的“烫”字。

解决方法：

1.给要输出的数组后面追加 '\0'

2.给数组初始化，如令：char recvbuf[DEFAULT\_BUFFLEN]="0";

代码：

服务器：

// BroadcastServer.cpp : 定义控制台应用程序的入口点。

//

#define \_CRT\_SECURE\_NO\_WARNINGS

#define \_WINSOCK\_DEPRECATED\_NO\_WARNINGS

#define WIN32\_LEAN\_AND\_MEAN

#include <windows.h>

#include <winsock2.h>

#include <ws2tcpip.h>

#include <stdlib.h>

#include <stdio.h>

// 连接到WinSock 2对应的lib文件：Ws2\_32.lib

#pragma comment (lib, "Ws2\_32.lib")

// 定义默认的缓冲区长度和端口号

constexpr auto DEFAULT\_BUFLEN = 512;

#define SERVERPORT "7210"

#define BROADCAST\_PORT "6000"

void Print\_SerIP();

int \_\_cdecl main(void)

{

WSADATA wsaData;

int iResult;

SOCKET ServerSocket = INVALID\_SOCKET; // 单播套接字，用于接收数据

SOCKET BroadcastSocket = INVALID\_SOCKET; // 广播套接字，用于发送数据

struct addrinfo\* result1 = NULL, \* result2 = NULL;

struct addrinfo hints;

int iSendResult;

char recvbuf[DEFAULT\_BUFLEN];

char sendbuf[DEFAULT\_BUFLEN];

int recvbuflen = DEFAULT\_BUFLEN;

int sendbuflen = DEFAULT\_BUFLEN;

sockaddr\_in clientaddr, broadaddr;

int clientlen = sizeof(sockaddr\_in);

// 初始化WinSock

iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);

if (iResult != 0)

{

printf("WSAStartup failed with error: %d\n", iResult);

return 1;

}

ZeroMemory(&hints, sizeof(hints));

// 声明IPv4地址族、数据报套接字、UDP协议

hints.ai\_family = AF\_INET;

hints.ai\_socktype = SOCK\_DGRAM;

hints.ai\_protocol = IPPROTO\_UDP;

/\* ----------------- 服务器端广播地址 ---------------------------\*/

// 解析服务器地址和端口号

iResult = getaddrinfo("255.255.255.255", BROADCAST\_PORT, &hints, &result2);

if (iResult != 0)

{

printf("getaddrinfo failed with error: %d\n", iResult);

WSACleanup();

return 1;

}

// 创建服务器端的广播数据报套接字

BroadcastSocket = socket(result2->ai\_family, result2->ai\_socktype,

result2->ai\_protocol);

if (BroadcastSocket == INVALID\_SOCKET)

{

printf("socket failed with error: %ld\n", WSAGetLastError());

freeaddrinfo(result2);

WSACleanup();

return 1;

}

//freeaddrinfo(result2);

//设置服务器开启广播选项

BOOL bBroadcast = TRUE;

iResult = setsockopt(BroadcastSocket, SOL\_SOCKET, SO\_BROADCAST, (const char\*)&bBroadcast, sizeof(BOOL));

if (iResult == SOCKET\_ERROR) {

printf("setsockopt函数调用错误，错误号： %d\n", WSAGetLastError());

WSACleanup();

return -1;

}

/\* ----------------- 服务器端单播地址 ---------------------------\*/

hints.ai\_flags = AI\_PASSIVE;

// 解析服务器地址和端口号

iResult = getaddrinfo(INADDR\_ANY, SERVERPORT, &hints, &result1);

if (iResult != 0)

{

printf("getaddrinfo failed with error: %d\n", iResult);

WSACleanup();

return 1;

}

// 创建服务器端的数据报套接字

ServerSocket = socket(result1->ai\_family, result1->ai\_socktype,

result1->ai\_protocol);

if (ServerSocket == INVALID\_SOCKET)

{

printf("socket failed with error: %ld\n", WSAGetLastError());

freeaddrinfo(result1);

WSACleanup();

return 1;

}

// 为套接字绑定地址和端口号

iResult = bind(ServerSocket, result1->ai\_addr, (int)result1->ai\_addrlen);

if (iResult == SOCKET\_ERROR)

{

printf("bind failed with error: %d\n", WSAGetLastError());

freeaddrinfo(result1);

closesocket(ServerSocket);

WSACleanup();

return 1;

}

//freeaddrinfo(result1);

// 服务器端，单播套接字用于接收，则可以关闭发送功能

//shutdown(ServerSocket, 1);

// 。。。 关闭接收功能

//shutdown(BroadcastSocket, 0);

Print\_SerIP();

printf("UDP server starting\n");

// 循环： 单播接收数据，用广播发送

while (true)

{

ZeroMemory(&clientaddr, sizeof(clientaddr));

iResult = recvfrom(ServerSocket, recvbuf, recvbuflen, 0,

(SOCKADDR\*)&clientaddr, &clientlen);

if (iResult > 0)

{

// 重新拼装数据，准备发送

memset(sendbuf, 0, sendbuflen);

strcpy(sendbuf, inet\_ntoa(clientaddr.sin\_addr));

strcat(sendbuf, ":");

strcat(sendbuf, recvbuf); // 可能会使sendbuf溢出

iResult = sendto(BroadcastSocket, sendbuf, strlen(sendbuf) + 1, 0, result2->ai\_addr, (int)result2->ai\_addrlen);

if (iResult > 0)

printf("服务器端发送广播数据%s\n", sendbuf);

else if (iResult == SOCKET\_ERROR)

{

printf("sendto 函数调用错误，错误号: %ld\n", WSAGetLastError());

continue;

}

}

else {

printf("recvfrom 函数调用错误，错误号: %d\n", WSAGetLastError());

continue;

}

}

// 关闭套接字，释放资源

closesocket(ServerSocket);

WSACleanup();

return 0;

}

// 输出本机IP

void Print\_SerIP()

{

char host[255];

if (gethostname(host, sizeof(host)) == SOCKET\_ERROR)

{

printf("无法获取主机名\n");

}

else

{

printf("本机计算机名为:\t\b%s\n", host);

}

struct hostent\* p = gethostbyname(host);

if (p == 0)

{

printf("无法获取计算机主机名及IP");

}

else

{

printf("本地环回测试IP为:\t127.0.0.1\n");

//本机IP:利用循环,输出本机所有IP

for (int i = 0; p->h\_addr\_list[i] != 0; i++)

{

struct in\_addr in;

memcpy(&in, p->h\_addr\_list[i], sizeof(struct in\_addr));

printf("第%d块网卡的IP为:\t\b%s\n", i + 1, inet\_ntoa(in));

}

}

}

客户端：

// BroadcastServer.cpp : 定义控制台应用程序的入口点。

//

#define \_CRT\_SECURE\_NO\_WARNINGS

#define \_WINSOCK\_DEPRECATED\_NO\_WARNINGS

#define WIN32\_LEAN\_AND\_MEAN

#define BROADCAST\_PORT "6000"

#undef UNICODE

#include <windows.h>

#include <winsock2.h>

#include <ws2tcpip.h>

#include <stdlib.h>

#include <stdio.h>

#include <process.h>

// 连接到WinSock 2对应的lib文件：Ws2\_32.lib

#pragma comment (lib, "Ws2\_32.lib")

// 定义默认的缓冲区长度和端口号

#define DEFAULT\_BUFLEN 512

#define SERVERPORT "7210"

#define BROADCAST\_PORT "6000"

void udp\_client\_recv\_thread(SOCKET\* c); // 线程函数，用于处理广播接收数据

int \_\_cdecl main(void)

{

system("mode con cols=80 lines=25"); // 设置窗口大小

system("title client");

WSADATA wsaData;

int iResult;

SOCKET ConnectLessSocket = INVALID\_SOCKET; // 单播套接字，用于接收数据

SOCKET BroadRcvSocket = INVALID\_SOCKET; // 广播套接字，用于发送数据

struct addrinfo\* result1 = NULL; // 发

struct addrinfo\* result2 = NULL; // 收

struct addrinfo hints;

int iSendResult;

char recvbuf[DEFAULT\_BUFLEN];

char sendbuf[DEFAULT\_BUFLEN];

int recvbuflen = DEFAULT\_BUFLEN;

int sendbuflen = DEFAULT\_BUFLEN;

sockaddr\_in clientaddr;

int clientlen = sizeof(sockaddr\_in);

char server\_name[50] = "";

// 初始化WinSock

iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);

if (iResult != 0)

{

printf("WSAStartup failed with error: %d\n", iResult);

return 1;

}

ZeroMemory(&hints, sizeof(hints));

// 声明IPv4地址族、数据报套接字、UDP协议

hints.ai\_family = AF\_INET;

hints.ai\_socktype = SOCK\_DGRAM;

hints.ai\_protocol = IPPROTO\_UDP;

/\*--------- 发 数据套接字 ----------------------\*/

while (true)

{

printf("请输入服务器域名/IP地址：\n>>>");

fflush(stdout);

rewind(stdin); // 清空缓冲区

scanf\_s("%s", server\_name, 49); // 输入服务器IP/域名

//strcpy(server\_name, "127.0.0.1");

// 解析服务器地址和端口号

iResult = getaddrinfo(server\_name, SERVERPORT, &hints, &result1);

if (iResult != 0) {

printf("getaddrinfo failed with error: %d\n", iResult);

WSACleanup();

return 1;

}

// 创建数据报套接字

ConnectLessSocket = socket(result1->ai\_family, result1->ai\_socktype,

result1->ai\_protocol);

if (ConnectLessSocket == INVALID\_SOCKET) {

printf("socket failed with error: %ld\n", WSAGetLastError());

WSACleanup();

return 1;

}

break;

}

/\*--------- 收 数据套接字 ----------------------\*/

// 解析服务器地址和端口号 // INADDR\_ANY

hints.ai\_flags = AI\_PASSIVE;

iResult = getaddrinfo(NULL, BROADCAST\_PORT, &hints, &result2);

if (iResult != 0)

{

printf("getaddrinfo failed with error: %d\n", iResult);

WSACleanup();

return 1;

}

// 为无连接的服务器创建套接字

BroadRcvSocket = socket(result2->ai\_family, result2->ai\_socktype,

result2->ai\_protocol);

if (BroadRcvSocket == INVALID\_SOCKET)

{

printf("socket failed with error: %ld\n", WSAGetLastError());

freeaddrinfo(result2);

WSACleanup();

return 1;

}

// 为套接字绑定地址和端口号

iResult = bind(BroadRcvSocket, result2->ai\_addr, (int)result2->ai\_addrlen);

if (iResult == SOCKET\_ERROR)

{

printf("bind failed with error: %d\n", WSAGetLastError());

freeaddrinfo(result2);

closesocket(BroadRcvSocket);

WSACleanup();

return 1;

}

freeaddrinfo(result2);

// 接收数据

\_beginthread((\_beginthread\_proc\_type)udp\_client\_recv\_thread, 0, &BroadRcvSocket);

while (1) {

rewind(stdin);

fgets(sendbuf, DEFAULT\_BUFLEN, stdin);

printf("-------------------------\n");

// 发送缓冲区中的数据

iResult = sendto(ConnectLessSocket, sendbuf, (int)strlen(sendbuf) + 1, 0,

result1->ai\_addr, (int)result1->ai\_addrlen);

if (iResult == SOCKET\_ERROR) {

printf("sendto failed with error: %d\n", WSAGetLastError());

closesocket(ConnectLessSocket);

WSACleanup();

return 1;

}

//ZeroMemory(&clientaddr, sizeof(clientaddr));

//iResult = recvfrom(BroadRcvSocket, recvbuf, recvbuflen, 0,

// (SOCKADDR\*)&clientaddr, &clientlen);

//if (iResult > 0)

//{

// // 情况1：成功接收到数据

// printf("=========================\n");

// printf("Bytes received: [%d]\n%s\n", iResult, recvbuf);

//}

//else

//{

// // 情况3：接收发生错误

// printf("recv failed with error: %d\n", WSAGetLastError());

// closesocket(BroadRcvSocket);

// WSACleanup();

// return 1;

//}

}

// 关闭套接字，释放资源

closesocket(BroadRcvSocket);

WSACleanup();

return 0;

}

void udp\_client\_recv\_thread(SOCKET\* c)//

{

int iResult;

char recvbuf[DEFAULT\_BUFLEN];

sockaddr\_in clientaddr;

int clientlen = sizeof(sockaddr\_in);

while (true)

{

memset(recvbuf, 0, DEFAULT\_BUFLEN);

ZeroMemory(&clientaddr, sizeof(clientaddr));

iResult = recvfrom(\*c, recvbuf, DEFAULT\_BUFLEN, 0, NULL, NULL);

if (iResult > 0)

{

printf("%s ", recvbuf); // recvbuf中有换行符号

}

else

{

printf("recvfrom 函数调用错误，错误号: %d\n", WSAGetLastError());

return;

}

}

\_endthread();

}

结果截图：

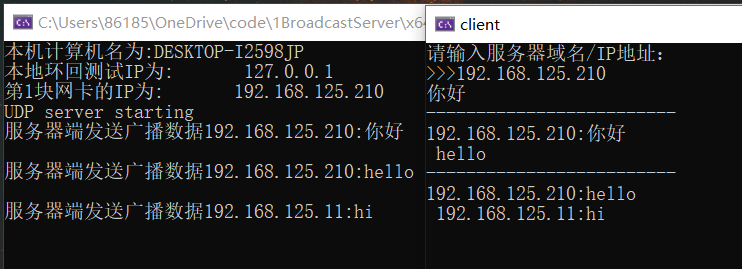


图3 服务端：虚拟机1，客户端：虚拟机1运行结果

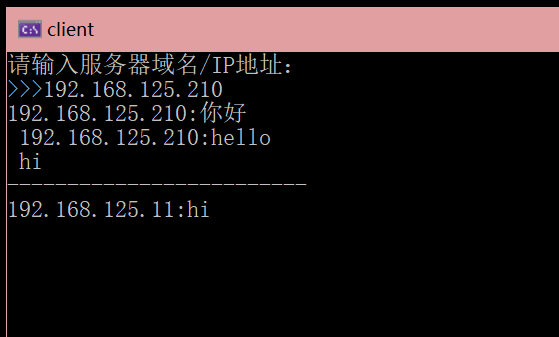


图4 客户端：虚拟机2运行结果

wireshark抓包：



图5 wireshark 抓包

问题解决：

在vs2022 current中，运行后的部分错误可通过将符合模式“是”改为“否”进行解决，如图6所示。

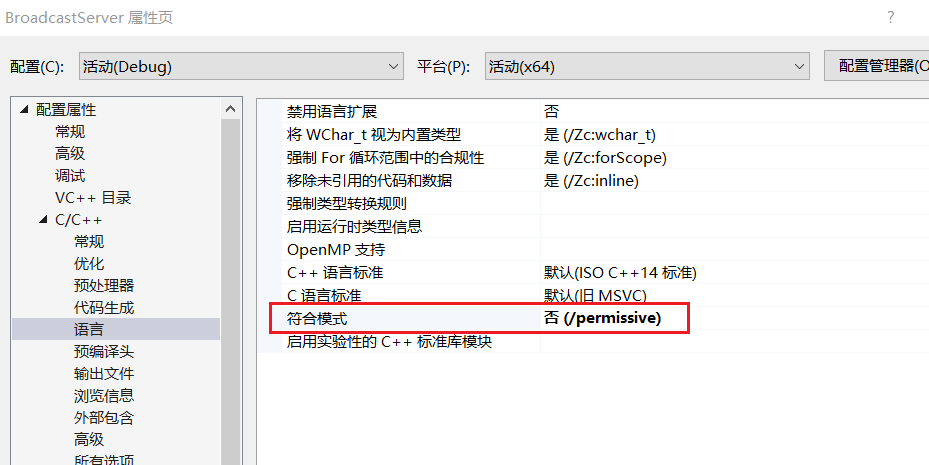


图6 修改符合模式选项

【实验总结】

在实验1中接收缓冲区未清空，会出现大量的“烫”字，通过给数组初始化，char recvbuf[DEFAULT\_BUFFLEN]="0"解决。由于vs版本的原因，运行后可能会出现很多的问题，在vs2022 current中，将符合模式改为“否”可以解决部分问题。

**实验5 原始套接字编程实例--ping程序实现**

【实验要求】

1.构造CMP协议首部结构

2.构造ICMP回射请求结构。

3.构造ICMP回射应答结构。

4.构造IP首部结构。

5.创建原始套接字。

6.根据用户指定的地址获取目标IP.

7.对目标地址循环发送CMP请求。

8.等待对方响应，并计算时间间隔

9.仿照ping指令输出形式给出执行输出。

【实验过程】

Ping.h

#pragma once

#pragma pack(1)

#define ICMP\_ECHOREPLY 0

#define ICMP\_ECHOREQ 8

// IP首部由 RFC 791定义

typedef struct tagIPHDR

{

u\_char VIHL; // 版本Version和报头长度IHL

u\_char TOS; // 服务类型Type Of Service

short TotLen; // 总长度字段Total Length

short ID; // 标识Identification

short FlagOff; // 标志位和片偏移Flags and Fragment Offset

u\_char TTL; // 生存期Time To Live

u\_char Protocol; // 协议字段

u\_short Checksum; // 校验和

struct in\_addr iaSrc; // 源ip地址Internet Address - Source

struct in\_addr iaDst; // 目标ip地址Internet Address - Destination

}IPHDR, \* PIPHDR;

// ICMP首部由RFC 792定义

typedef struct tagICMPHDR

{

u\_char Type; // 类型

u\_char Code; // 代码

u\_short Checksum; // 校验和

u\_short ID; // 标识Identification

u\_short Seq; // 序号Sequence

char Data; // Data

}ICMPHDR, \* PICMPHDR;

#define REQ\_DATASIZE 32 // Echo Request Data size

// ICMP回射请求结构

typedef struct tagECHOREQUEST

{

ICMPHDR icmpHdr;//icmp协议首部

DWORD dwTime;//时间戳

char cData[REQ\_DATASIZE];//icmp数据

}ECHOREQUEST, \* PECHOREQUEST;

// ICMP回射应答结构

typedef struct tagECHOREPLY

{

IPHDR ipHdr; //ip协议首部

ECHOREQUEST echoRequest; //icmp请求结构

char cFiller[256]; //填充

}ECHOREPLY, \* PECHOREPLY;

#pragma pack()

#pragma once

Ping.cpp

// PING.C -- Ping program using ICMP and RAW Sockets

#define \_WINSOCK\_DEPRECATED\_NO\_WARNINGS

//#include "StdAfx.h"

#include <stdio.h>

#include <stdlib.h>

//#include <winsock.h>

#include <WinSock2.h>

#pragma comment (lib,"Ws2\_32.lib")

#include "ping.h"

// 内部函数

void Ping(LPCSTR pstrHost);

void ReportError(LPCSTR pstrFrom);

int WaitForEchoReply(SOCKET s);

u\_short in\_cksum(u\_short\* addr, int len);

// ICMP回送请求/回复功能

int SendEchoRequest(SOCKET, LPSOCKADDR\_IN);

DWORD RecvEchoReply(SOCKET, LPSOCKADDR\_IN, u\_char\*);

// main()函数

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

{

WSADATA wsaData;

WORD wVersionRequested = MAKEWORD(1, 1);

int nRet;

// 检查输入合法性

if (argc != 2)

{

fprintf(stderr, "\nUsage: ping hostname\n");

return;

}

//初始化套接字

nRet = WSAStartup(wVersionRequested, &wsaData);

if (nRet)

{

fprintf(stderr, "\nError initializing WinSock\n");

return;

}

//核对套接字版本

if (wsaData.wVersion != wVersionRequested)

{

fprintf(stderr, "\nWinSock version not supported\n");

return;

}

//调用Ping()函数完成具体功能

Ping(argv[1]);

// 释放 WinSock

WSACleanup();

}

// Ping()函数

void Ping(LPCSTR pstrHost)

{

//定义变量

SOCKET rawSocket;

LPHOSTENT lpHost;

struct sockaddr\_in saDest;

struct sockaddr\_in saSrc;

DWORD dwTimeSent;

DWORD dwElapsed;

u\_char cTTL;

int nLoop;

int nRet;

// 创建原始套接字，指定协议类型为IPPROTO\_ICMP协议

rawSocket = socket(AF\_INET, SOCK\_RAW, IPPROTO\_ICMP);

if (rawSocket == SOCKET\_ERROR)

{

ReportError("socket()");

return;

}

//根据用户指定的地址获取目标IP

lpHost = gethostbyname(pstrHost);

if (lpHost == NULL)

{

fprintf(stderr, "\nHost not found: %s\n", pstrHost);

return;

}

//填充套接字的目的端点地址

saDest.sin\_addr.s\_addr = \*((u\_long FAR\*) (lpHost->h\_addr));

saDest.sin\_family = AF\_INET;

saDest.sin\_port = 0;

// 在控制台输出当前的工作

printf("\nPinging %s [%s] with %d bytes of data:\n",pstrHost,

inet\_ntoa(saDest.sin\_addr),REQ\_DATASIZE);

//对目标地址连续ping 4次

for (nLoop = 0; nLoop < 4; nLoop++)

{

// 发送ICMP请求

SendEchoRequest(rawSocket, &saDest);

// 等待响应到达

nRet = WaitForEchoReply(rawSocket);

if (nRet == SOCKET\_ERROR)

{

ReportError("select()");

break;

}

if (!nRet)

{

printf("\nTimeOut");

break;

}

//接收响应

dwTimeSent = RecvEchoReply(rawSocket, &saSrc, &cTTL);

//计算响应间隔时间

dwElapsed = GetTickCount64() - dwTimeSent;

//输出结果

printf("\nReply from: %s: bytes=%d time=%ldms TTL=%d",

inet\_ntoa(saSrc.sin\_addr),REQ\_DATASIZE,dwElapsed,cTTL);

}

printf("\n");

//关闭套接字

nRet = closesocket(rawSocket);

if (nRet == SOCKET\_ERROR)

ReportError("closesocket()");

}

// SendEchoRequest()函数填充ICMP ECHO请求报文结构，并调用sendto()函数完成单词ICMP ECHO请求的发送。

int SendEchoRequest(SOCKET s, LPSOCKADDR\_IN lpstToAddr)

{

static ECHOREQUEST echoReq;

static int nId = 1;

static int nSeq = 1;

int nRet;

// 填充ICMP ECHO请求

echoReq.icmpHdr.Type = ICMP\_ECHOREQ;

echoReq.icmpHdr.Code = 0;

echoReq.icmpHdr.Checksum = 0;

echoReq.icmpHdr.ID = nId++;

echoReq.icmpHdr.Seq = nSeq++;

// 填写一些要发送的数据

for (nRet = 0; nRet < REQ\_DATASIZE; nRet++)

echoReq.cData[nRet] = ' ' + nRet;

// 获得当前的时钟并填充

echoReq.dwTime = GetTickCount64();

//计算校验和

echoReq.icmpHdr.Checksum = in\_cksum((u\_short\*)&echoReq, sizeof(ECHOREQUEST));

//发送回射请求

nRet = sendto(s, //套接字

(LPSTR)&echoReq, //缓冲区

sizeof(ECHOREQUEST), //缓冲区长度

0, //发送标志

(LPSOCKADDR)lpstToAddr, //目的地址

sizeof(SOCKADDR\_IN)); //地址长度

if (nRet == SOCKET\_ERROR)

ReportError("sendto()");

return (nRet);

}

//RecvEchoReply()函数接收ICMP ECHO应答，并提取其携带的时间戳返回

DWORD RecvEchoReply(SOCKET s, LPSOCKADDR\_IN lpsaFrom, u\_char\* pTTL)

{

ECHOREPLY echoReply;

int nRet;

int nAddrLen = sizeof(struct sockaddr\_in);

// 接收回射应答

nRet = recvfrom(s, //套接字

(LPSTR)&echoReply, //接收缓冲区

sizeof(ECHOREPLY), //缓冲区长度

0, //接收标志

(LPSOCKADDR)lpsaFrom, //数据来源地址

&nAddrLen); //来源地址长度

//判断接收返回值

if (nRet == SOCKET\_ERROR)

ReportError("recvfrom()");

// 返回发送时间和IP头中的TTL

\*pTTL = echoReply.ipHdr.TTL;

return(echoReply.echoRequest.dwTime);

}

void ReportError(LPCSTR pWhere)

{

fprintf(stderr, "\n%s error: %d\n",

WSAGetLastError());

}

// WaitForEchoReply()函数采用select模型来等待ICMP ECHO应答

int WaitForEchoReply(SOCKET s)

{

struct timeval Timeout;

fd\_set readfds;

//设置读等待套接字组

readfds.fd\_count = 1;

readfds.fd\_array[0] = s;

Timeout.tv\_sec = 5;

Timeout.tv\_usec = 0;

//等待套接字上的网络事件

return(select(1, &readfds, NULL, NULL, &Timeout));

}

//计算校验和

u\_short in\_cksum(u\_short\* addr, int len)

{

register int nleft = len;

register u\_short\* w = addr;

register u\_short answer;

register int sum = 0;

//使用 32 位累加器 ,进行 16 位的反馈计算

while (nleft > 1) {

sum += \*w++;

nleft -= 2;

}

//补全奇数位

if (nleft == 1) {

u\_short u = 0;

\*(u\_char\*)(&u) = \*(u\_char\*)w;

sum += u;

}

//将反馈的 16 位从高位移到低位

sum = (sum >> 16) + (sum & 0xffff); //将高16位添加到低16位

sum += (sum >> 16); //加进位

answer = ~sum; //截断为16位

return (answer);

}

根据用户输入的主机名称或ip地址，向对方发送ICMP请求报文，收到回复的ICMP应答报文后，计算时间差来显示与对方的连通情况。程序的输入参数是[www.taobao.com](http://www.taobao.com)，程序中调用gethostbyname()函数将主机名转换为ip地址139.170.154.162。并循环四次向该地址发送ICMP请求包，每次收到ICMP应答包后计算往返的时间间隔。数据包包括8个字节的ICMP协议头，32个字节包括ICMP协议的数据，其中协议头中包含时间字段。运行结果如图1所示。

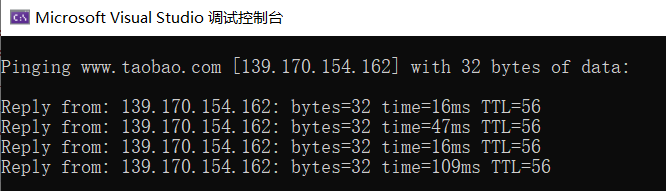


图1

使用wireshark软件，可以将程序运行过程中发送和接收到的数据包捕获到。图2给出了所捕获的数据包情况。

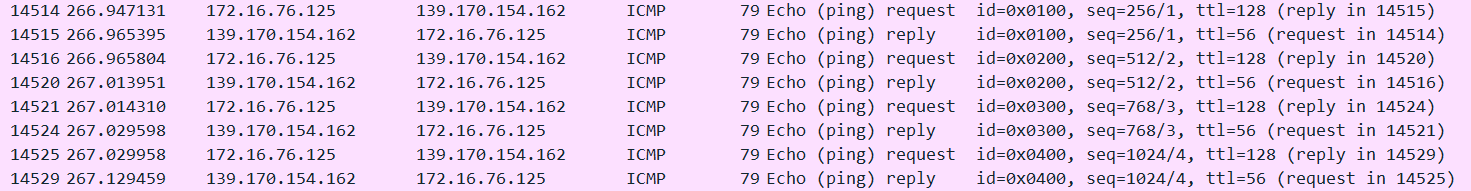


图2

【实验总结】

通过这次实验，我加深了对Socket原始套接字编程的理解，经过实现Ping程序，熟悉了IP、ICMP等，掌握TCP/IP网络协议的基本实现方法。同时学会如果通过讨论、交流、找资料来独立解决所遇到的问题。在编写过程中，一些基本的常见的函数不会应用，这使我发现自己知识的匮乏，在以后的学习过程中得要继续努力。

**实验6 基于消息选择模型的聊天软件**

【实验要求】

1、定义套接字网络事件对应的用户消息

2、如果不存在窗口，则创建窗口和窗口例程支持函数

3、调用 WSAAsyncSelect0函数为套接字设置网络事件、用户消息和消息接收窗口之间的关系。

4、增加消息循环的具体功能，或者添加消息与消息处理函数的映射关系。

5、添加消息处理框架的具体功能，判断是哪个套接字上发生了网络事件，使用 WSAGETSELECTEVENT宏了解所发生的网络事件，从而进行相应的处理。

界面功能：

设计用户界面，能够判断用户输入的合法性，接收用户输入的服务器IP地址和端口信息，获取用户的的聊天内容，并将聊天内容显示在用户界面中。

网络功能：

提供服务器和客户端两种角色的网络功能。对服务器，设计若干函数来完成服务器的主体功能；对客户端，设计若干函数完成客户端的主体功能。

异步I/O处理功能：

基于 WSAAsyncSelectt模型实现异步/O,构建程序中合理的消息处理框架，其中包括自定义消息声明，消息接收窗口的创建与注册，消息的判断与处理等。

【实验过程】

// MFCApplication1Dlg.h: 头文件

#pragma once

#define MAXLINE 200

#define SERVER 1

#define CLIENT 0

#define WM\_READ WM\_USER+101

// CMFCApplication1Dlg 对话框

class CMFCApplication1Dlg : public CDialogEx

{

// 构造

public:

CMFCApplication1Dlg(CWnd\* pParent = nullptr); // 标准构造函数

// 对话框数据

#ifdef AFX\_DESIGN\_TIME

enum { IDD = IDD\_MFCAPPLICATION1\_DIALOG };

#endif

protected:

virtual void DoDataExchange(CDataExchange\* pDX); // DDX/DDV 支持

// 实现

protected:

HICON m\_hIcon;

// 生成的消息映射函数

virtual BOOL OnInitDialog();

afx\_msg void OnSysCommand(UINT nID, LPARAM lParam);

afx\_msg void OnPaint();

afx\_msg HCURSOR OnQueryDragIcon();

DECLARE\_MESSAGE\_MAP()

public:

CButton m\_nServerType;

CIPAddressCtrl m\_ulIP;

CEdit m\_usPort;

CEdit m\_SendText;

CListBox m\_List;

SOCKET m\_Socket = INVALID\_SOCKET;

struct sockaddr\_in m\_peer, m\_local;

afx\_msg void OnBnClickedBegin();

afx\_msg void OnBnClickedSend();

afx\_msg void OnBnClickedServer();

afx\_msg void OnBnClickedClient();

afx\_msg LRESULT OnRecvFrom(WPARAM wParam, LPARAM lParam);

};

// MFCApplication1Dlg.cpp: 实现文件

//

#include "pch.h"

#include "framework.h"

#include "MFCApplication1.h"

#include "MFCApplication1Dlg.h"

#include "afxdialogex.h"

#ifdef \_DEBUG

#define new DEBUG\_NEW

#endif

// 用于应用程序“关于”菜单项的 CAboutDlg 对话框

class CAboutDlg : public CDialogEx

{

public:

CAboutDlg();

// 对话框数据

#ifdef AFX\_DESIGN\_TIME

enum { IDD = IDD\_ABOUTBOX };

#endif

protected:

virtual void DoDataExchange(CDataExchange\* pDX); // DDX/DDV 支持

// 实现

protected:

DECLARE\_MESSAGE\_MAP()

};

CAboutDlg::CAboutDlg() : CDialogEx(IDD\_ABOUTBOX)

{

}

void CAboutDlg::DoDataExchange(CDataExchange\* pDX)

{

CDialogEx::DoDataExchange(pDX);

}

BEGIN\_MESSAGE\_MAP(CAboutDlg, CDialogEx)

END\_MESSAGE\_MAP()

// CMFCApplication1Dlg 对话框

CMFCApplication1Dlg::CMFCApplication1Dlg(CWnd\* pParent /\*=nullptr\*/)

: CDialogEx(IDD\_MFCAPPLICATION1\_DIALOG, pParent)

{

m\_hIcon = AfxGetApp()->LoadIcon(IDR\_MAINFRAME);

}

void CMFCApplication1Dlg::DoDataExchange(CDataExchange\* pDX)

{

CDialogEx::DoDataExchange(pDX);

DDX\_Control(pDX, IDC\_SERVER, m\_nServerType);

DDX\_Control(pDX, IDC\_IPADDRESS, m\_ulIP);

DDX\_Control(pDX, IDC\_PORT, m\_usPort);

DDX\_Control(pDX, IDC\_SENDTEXT, m\_SendText);

DDX\_Control(pDX, IDC\_LIST1, m\_List);

}

BEGIN\_MESSAGE\_MAP(CMFCApplication1Dlg, CDialogEx)

ON\_WM\_SYSCOMMAND()

ON\_WM\_PAINT()

ON\_WM\_QUERYDRAGICON()

ON\_BN\_CLICKED(IDC\_BEGIN, &CMFCApplication1Dlg::OnBnClickedBegin)

ON\_BN\_CLICKED(IDC\_SEND, &CMFCApplication1Dlg::OnBnClickedSend)

ON\_BN\_CLICKED(IDC\_SERVER, &CMFCApplication1Dlg::OnBnClickedServer)

ON\_BN\_CLICKED(IDC\_CLIENT, &CMFCApplication1Dlg::OnBnClickedClient)

ON\_MESSAGE(WM\_READ, &CMFCApplication1Dlg::OnRecvFrom)

END\_MESSAGE\_MAP()

// CMFCApplication1Dlg 消息处理程序

BOOL CMFCApplication1Dlg::OnInitDialog()

{

CDialogEx::OnInitDialog();

// 将“关于...”菜单项添加到系统菜单中。

// IDM\_ABOUTBOX 必须在系统命令范围内。

ASSERT((IDM\_ABOUTBOX & 0xFFF0) == IDM\_ABOUTBOX);

ASSERT(IDM\_ABOUTBOX < 0xF000);

CMenu\* pSysMenu = GetSystemMenu(FALSE);

if (pSysMenu != nullptr)

{

BOOL bNameValid;

CString strAboutMenu;

bNameValid = strAboutMenu.LoadString(IDS\_ABOUTBOX);

ASSERT(bNameValid);

if (!strAboutMenu.IsEmpty())

{

pSysMenu->AppendMenu(MF\_SEPARATOR);

pSysMenu->AppendMenu(MF\_STRING, IDM\_ABOUTBOX, strAboutMenu);

}

}

// 设置此对话框的图标。 当应用程序主窗口不是对话框时，框架将自动

// 执行此操作

SetIcon(m\_hIcon, TRUE); // 设置大图标

SetIcon(m\_hIcon, FALSE); // 设置小图标

// TODO: 在此添加额外的初始化代码

return TRUE; // 除非将焦点设置到控件，否则返回 TRUE

}

void CMFCApplication1Dlg::OnSysCommand(UINT nID, LPARAM lParam)

{

if ((nID & 0xFFF0) == IDM\_ABOUTBOX)

{

CAboutDlg dlgAbout;

dlgAbout.DoModal();

}

else

{

CDialogEx::OnSysCommand(nID, lParam);

}

}

// 如果向对话框添加最小化按钮，则需要下面的代码

// 来绘制该图标。 对于使用文档/视图模型的 MFC 应用程序，

// 这将由框架自动完成。

void CMFCApplication1Dlg::OnPaint()

{

if (IsIconic())

{

CPaintDC dc(this); // 用于绘制的设备上下文

SendMessage(WM\_ICONERASEBKGND, reinterpret\_cast<WPARAM>(dc.GetSafeHdc()), 0);

// 使图标在工作区矩形中居中

int cxIcon = GetSystemMetrics(SM\_CXICON);

int cyIcon = GetSystemMetrics(SM\_CYICON);

CRect rect;

GetClientRect(&rect);

int x = (rect.Width() - cxIcon + 1) / 2;

int y = (rect.Height() - cyIcon + 1) / 2;

// 绘制图标

dc.DrawIcon(x, y, m\_hIcon);

}

else

{

CDialogEx::OnPaint();

}

}

//当用户拖动最小化窗口时系统调用此函数取得光标

//显示。

HCURSOR CMFCApplication1Dlg::OnQueryDragIcon()

{

return static\_cast<HCURSOR>(m\_hIcon);

}

void CMFCApplication1Dlg::OnBnClickedBegin()

{

int iResult = 0;

WSADATA wsaData;

UpdateData(TRUE);

//Windows Sockets Dll初始化

iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);

if (iResult != 0)

{

m\_List.InsertString(0, "WSAStartup函数出错.");

return;

}

//当程序作为服务器

if (m\_nServerType.GetState() == SERVER)

{

//为服务器的本地地址m\_local设置用户输入的IP和端口号

memset(&m\_local, 0, sizeof(m\_local));

m\_local.sin\_family = AF\_INET;

m\_local.sin\_addr.S\_un.S\_addr = htonl(INADDR\_ANY);

m\_local.sin\_port = htons(GetDlgItemInt(IDC\_PORT));

//创建服务器端的数据报套接字并在指定端口号上监听

m\_Socket = socket(AF\_INET,SOCK\_DGRAM,0);

if (m\_Socket == INVALID\_SOCKET)

{

m\_List.InsertString(0, "socket函数出错!");

WSACleanup();

return;

}

iResult = bind(m\_Socket, (const struct sockaddr\*)&m\_local, sizeof(SOCKADDR\_IN));

if (iResult == SOCKET\_ERROR) {

m\_List.InsertString(0, "bind函数出错!");

closesocket(m\_Socket);

WSACleanup();

return;

}

m\_List.InsertString(0, \_T("服务器已启动"));

}

else

{

//创建客户端的数据报套接字并向服务器请求建立连接

UpdateData(TRUE);

if (m\_ulIP.IsBlank() || GetDlgItemInt(IDC\_PORT) == NULL)

{

m\_List.InsertString(0, "IP 地址和端口号不能为空");

return;

}

// 指明服务器的地址m\_peer为用户输入的IP和端口号

DWORD ipaddr;

m\_ulIP.GetAddress(ipaddr);

memset(&m\_peer, 0, sizeof(m\_peer));

m\_peer.sin\_family = AF\_INET;

m\_peer.sin\_addr.S\_un.S\_addr = htonl(ipaddr);

m\_peer.sin\_port = htons(GetDlgItemInt(IDC\_PORT));

// 创建客户端的数据报套接字

m\_Socket = socket(AF\_INET, SOCK\_DGRAM,0 );

if (m\_Socket == INVALID\_SOCKET)

{

m\_List.InsertString(0, "socket函数出错!");

WSACleanup();

return;

}

m\_List.InsertString(0, \_T("客户端已启动"));

}

iResult = WSAAsyncSelect(m\_Socket, m\_hWnd, WM\_READ, FD\_READ);

if (iResult == SOCKET\_ERROR)

{

m\_List.InsertString(0, "WSAAsyncSelect设定失败!");

return;

}

//更新对话框控件的可操作性

GetDlgItem(IDC\_SENDTEXT)->EnableWindow(TRUE);

GetDlgItem(IDC\_SEND)->EnableWindow(TRUE);

GetDlgItem(IDC\_IPADDRESS)->EnableWindow(FALSE);

GetDlgItem(IDC\_SERVER)->EnableWindow(FALSE);

GetDlgItem(IDC\_CLIENT)->EnableWindow(FALSE);

GetDlgItem(IDC\_PORT)->EnableWindow(FALSE);

GetDlgItem(IDC\_BEGIN)->EnableWindow(FALSE);

}

void CMFCApplication1Dlg::OnBnClickedSend()

{

time\_t ticks;

char buff[MAXLINE], info[MAXLINE];

int iResult = -1;

// 更新对话框里的数据

UpdateData(TRUE);

if (m\_SendText.GetWindowTextLengthA() > MAXLINE)

{

m\_List.InsertString(0, "发送内容超长！");

return;

}

if (m\_peer.sin\_addr.S\_un.S\_addr == 0)

{

m\_List.InsertString(0, "目标地址为空，请确定正确的目标地址！");

return;

}

// 获取当前时间

ticks = time(NULL);

memset(buff, 0, sizeof(buff));

sprintf\_s(buff, MAXLINE, "%.24s\r\n", ctime(&ticks));

if (m\_nServerType.GetState() == SERVER)

{

sprintf\_s(info, MAXLINE, "%s服务器向%s发送数据:", buff, inet\_ntoa(m\_peer.sin\_addr));

}

else

{

sprintf\_s(info, MAXLINE, "%s客户端向%s发送数据:", buff, inet\_ntoa(m\_peer.sin\_addr));

}

// 发送数据

char Content[MAXLINE] = { 0 };

m\_SendText.GetWindowText((LPTSTR)&Content, MAXLINE);

iResult = sendto(m\_Socket,Content, m\_SendText.GetWindowTextLength(),0, (SOCKADDR\*)&m\_peer, sizeof(m\_peer));

if (iResult == SOCKET\_ERROR)

{

sprintf\_s(info, MAXLINE, "sendto函数调用错误，错误号：%d", WSAGetLastError());

m\_List.InsertString(0, info);

closesocket(m\_Socket);

}

else

{

m\_List.InsertString(0, strcat(info,Content));

}

}

void CMFCApplication1Dlg::OnBnClickedServer()

{

// TODO: 在此添加控件通知处理程序代码

m\_nServerType.SetCheck(SERVER);

UpdateData(TRUE);

}

void CMFCApplication1Dlg::OnBnClickedClient()

{

// TODO: 在此添加控件通知处理程序代码

m\_nServerType.SetCheck(CLIENT);

UpdateData(TRUE);

}

LRESULT CMFCApplication1Dlg::OnRecvFrom(WPARAM wParam, LPARAM lParam)

{

int iResult;

time\_t ticks;

char buff[MAXLINE], info[MAXLINE];

char recvbuf[MAXLINE];

struct sockaddr\_in cliaddr = {};

int addrlen = sizeof(sockaddr\_in);

CString str;

memset(recvbuf, 0, MAXLINE);

// 获取当前时间

ticks = time(NULL);

memset(buff, 0, sizeof(buff));

sprintf\_s(buff, MAXLINE, "%.24s\r\n", ctime(&ticks));

// 判断网络事件

if (WSAGETSELECTERROR(lParam))

m\_List.InsertString(0, "套接字错误。");

else

{

switch (WSAGETSELECTEVENT(lParam))

{

case FD\_READ:

// 接收数据

iResult = recvfrom(m\_Socket, recvbuf, MAXLINE, 0, (SOCKADDR\*)&cliaddr, &addrlen);

if (iResult == SOCKET\_ERROR)

{

sprintf\_s(info, MAXLINE, "%s接收数据发生错误，错误号：%d", buff, WSAGetLastError());

str = info;

m\_List.InsertString(0, str);

return 0;

}

sprintf\_s(info, MAXLINE, "%s接收到来自%s的数据：", buff, inet\_ntoa(cliaddr.sin\_addr));

str = info;

m\_List.InsertString(0, str + recvbuf);

// 更新通信对方地址

m\_peer = cliaddr;

break;

default:

break;

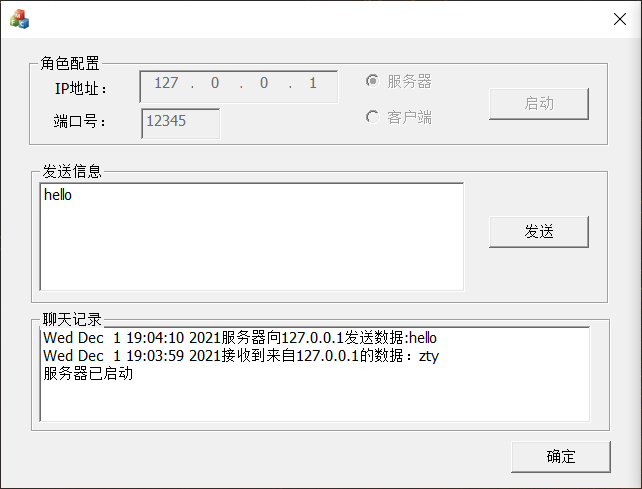
}

}

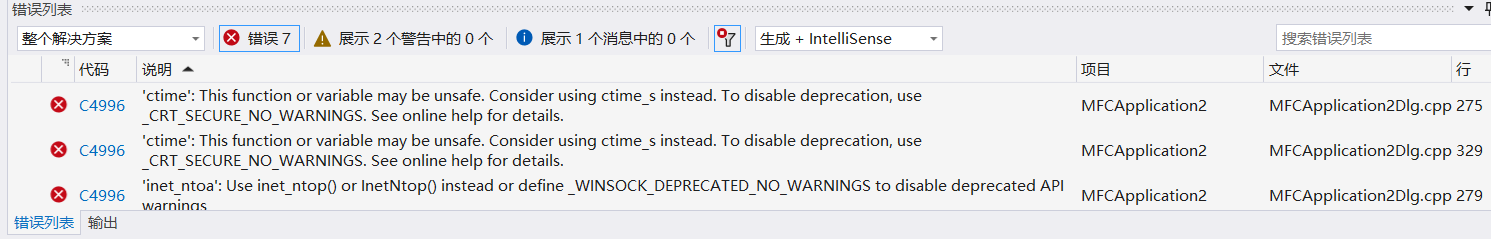
return 0;

}

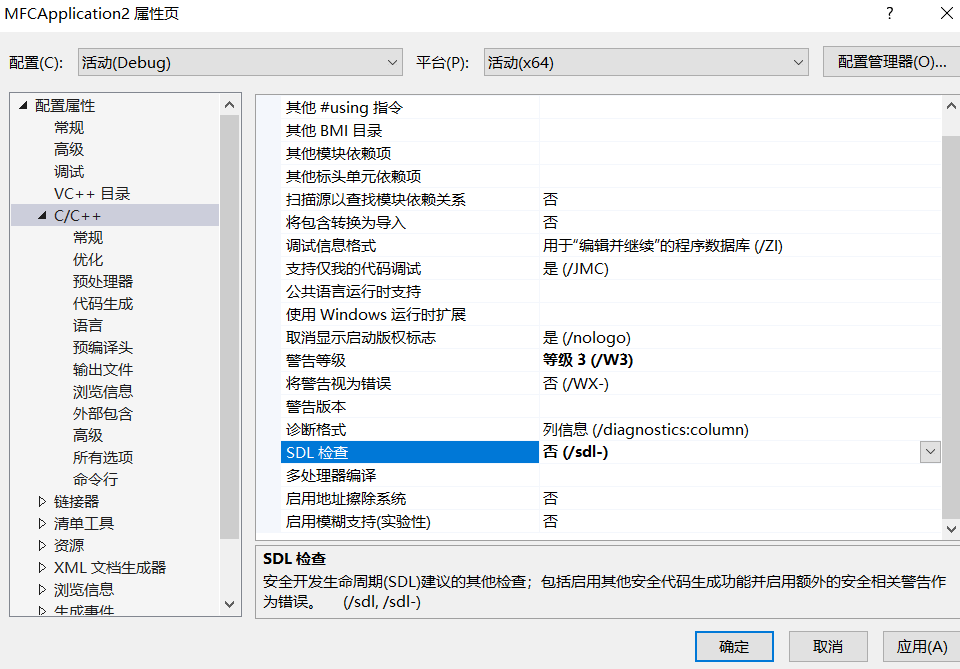
截图：



编译出现问题：



通过将SDL检查由是改为否解决：



【实验总结】

MFC框架中“消息映射”是通过巧妙的宏定义形成一张消息映射表格来完成的。消息一旦发生，框架就可以根据消息映射表格来进行消息映射和命令传递。