**实验四 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：

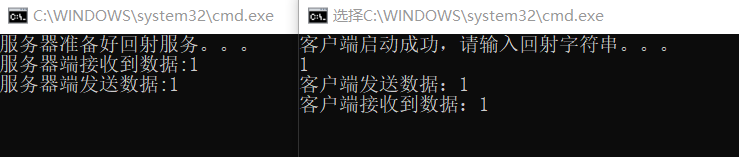


图1 运行结果

出现的问题：

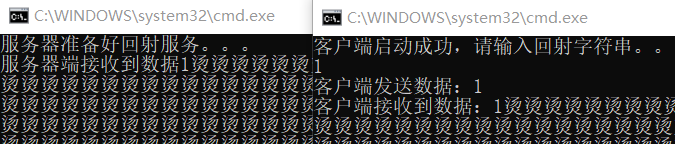


图2 缓冲区未清空的运行结果

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 INADDR\_ANY (ULONG)0x00000000

#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：

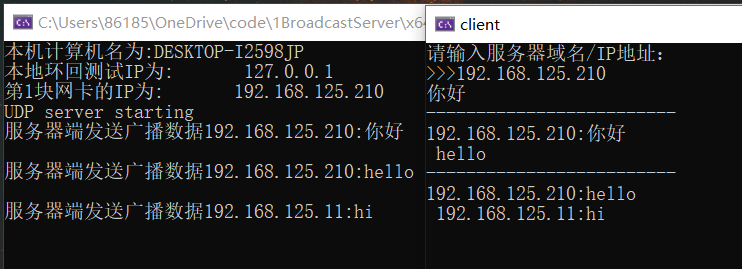


图3 服务端：虚拟机1，客户端：虚拟机1运行结果

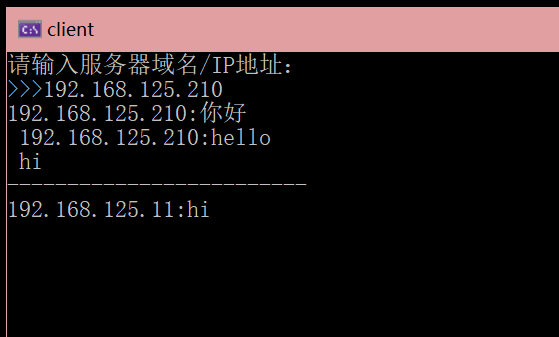


图4 客户端：虚拟机2运行结果

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。

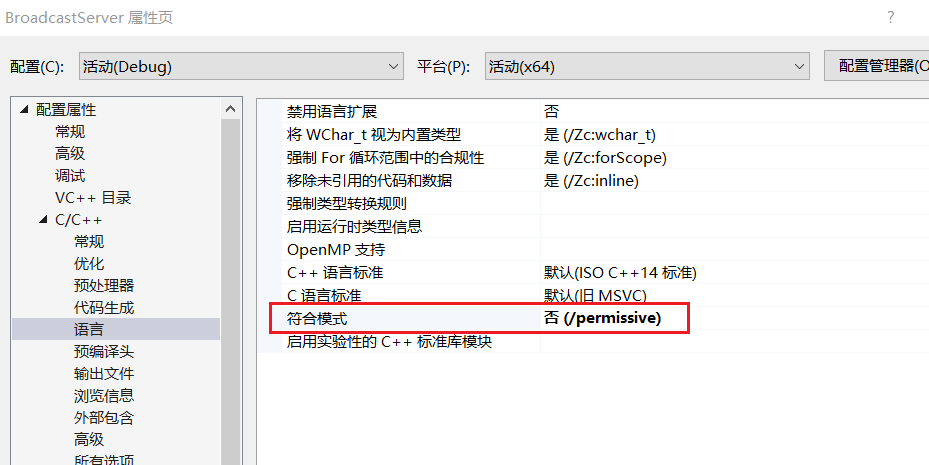


图6 修改符合模式选项

【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。