**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。

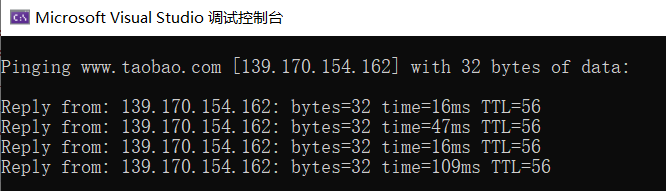


图1

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。

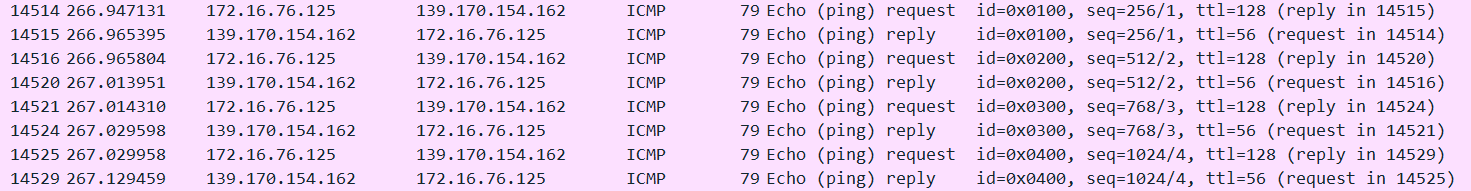


图2

【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。