<https://blog.csdn.net/chenpidaxia/article/details/77803388>

今天翻硬盘时，发现自己很久之前写的ping程序。突然想不如把ping程序放上来做一下也算让之前自己写的东西重见天日，当时还是花了挺多时间来写，感触良多。

首先呢，ping用到的协议是网络层的ICMP协议，发送/接收的是ICMP报文，最终的形式还是以一个IP报文在网络中传送。   
下面先定义一下IP头和ICMP协议的相关数据结构。

IP头

IP报文格式

0 8 16 19 31

+------------+------------+-------------------------+

| ver + hlen | 服务类型 | 总长度 |

+------------+------------+----+--------------------+

| 标识位 |flag| 分片偏移(13位) |

+------------+------------+----+--------------------+

| 生存时间 | 高层协议号 | 首部校验和 |

+------------+------------+-------------------------+

| 源 IP 地址 |

+---------------------------------------------------+

| 目的 IP 地址 |

+---------------------------------------------------+

数据结构，使用了位域来兼容协议格式，最终编译的时候使用1字节对齐

struct IPhead

{

//这里使用了C语言的位域，也就是说像version变量它的大小在内存中是占4bit，而不是8bit

uint8\_t version : 4; //IP协议版本

uint8\_t headLength : 4;//首部长度

uint8\_t serverce;//区分服务

uint16\_t totalLength;//总长度

uint16\_t flagbit;//标识

uint16\_t flag : 3;//标志

uint16\_t fragmentOffset : 13;//片偏移

char timetoLive;//生存时间（跳数）

uint8\_t protocol;//使用协议

uint16\_t headcheckSum;//首部校验和

uint32\_t srcIPadd;//源IP

uint32\_t dstIPadd;//目的IP

//可选项和填充我就不定义了

};

ICMP协议相关：

ICMP 请求报文

+--------+--------+---------------+

| 类型 | 代码 | 描述 |

+--------+--------+---------------+

| 8 | 0 | 回显请求 |

+--------+--------+---------------+

| 10 | 0 | 路由器请求 |

+--------+--------+---------------+

| 13 | 0 | 时间戳请求 |

+--------+--------+---------------+

| 15 | 0 | 信息请求(废弃)|

+--------+--------+---------------+

| 17 | 0 | 地址掩码请求 |

+--------+--------+---------------+

ICMP 应答报文

+--------+--------+---------------+

| 类型 | 代码 | 描述 |

+--------+--------+---------------+

| 0 | 0 | 回显回答 |

+--------+--------+---------------+

| 9 | 0 | 路由器回答 |

+--------+--------+---------------+

| 14 | 0 | 时间戳回答 |

+--------+--------+---------------+

| 16 | 0 | 信息回答(废弃)|

+--------+--------+---------------+

| 18 | 0 | 地址掩码回答 |

+--------+--------+---------------+

ICMP协议请求/回答报文格式

0 8 16 32

+--------+--------+-----------------+

| 类型 | 代码 | 校验和 |

+--------+--------+-----------------+

| 标识符 | 序号 |

+-----------------+-----------------+

| 回显数据 |

+-----------------+-----------------+

ICMP协议我只定义了请求/回答功能的报文格式。因为不同的类型和代码报文的内容组成不一样,ICMP分请求报文，回答报文，差错报文。差错报文我没写出来，因为用不着

ICMP相关数据结构

//ICMP头

struct ICMPhead

{

uint8\_t type;//类型

uint8\_t code;//代码

uint16\_t checkSum;//校验和

uint16\_t ident;//进程标识符

uint16\_t seqNum;//序号

};

//ICMP回显请求报文(发送用)

struct ICMPrecvReq

{

ICMPhead icmphead;//头部

uint32\_t timeStamp;//时间戳

char data[32];//数据

};

//ICMP应答报文(接收用)

struct ICMPansReply

{

IPhead iphead;//IP头

ICMPrecvReq icmpanswer;//ICMP报文

char data[1024];//应答报文携带的数据缓冲区

};

需要实现的函数也不多，只需要6个就能够满足基本的ping程序用了

//计算校验和，参数1：协议头的指针，需要转换成空指针。参数2：计算的协议类型。返回校验和

uint16\_t getCheckSum(void\* protocol,char\* type);

//ping程序，参数1：为目的IP地址。返回是否发送成功.参数2：只能写-t,或者不写

bool ping(const char\* dstIPaddr,const char\* param='\0');

//发送ICMP请求回答报文，参数1：套接字.参数2：目的地址.参数3：第num次发送

bool sendICMPReq(SOCKET &mysocket, sockaddr\_in &dstAddr, unsigned int num);

//读取ICMP应答报文，参数1：套接字.参数2：接收方地址.参数3：跳数

uint32\_t readICMPanswer(SOCKET &mysocket, sockaddr\_in &srcAddr, char &TTL);

//等待socket是否由数据需要读取

int waitForSocket(SOCKET &mysocket);

//执行一次ping操作,参数1：套接字.参数2：源地址.参数3：目的地址。参数4序号ping报文序号

void doPing(SOCKET &mysocket,sockaddr\_in &srcAddr,sockaddr\_in &dstAddr,int num);

//判断参数是否为空，空的话返回一个NULL指针，否侧提取参数存入到param中，同时返回

char\* isParamEmpty(char \*buffer, char \*param);

//捕获终止信号函数

void get\_ctrl\_stop(int signal);

本程序实在windows平台下实现的额，使用winsock的话需要添加一个预处理命令   
#pragma comment(lib,"ws2\_32.lib")来使能winsock的lib的使用   
接下来就是正题了，ICMP报文不经过TCP或者UDP的封装，而我们有知道sock层的API是面向用户（程序员）的，一般的sock是当用户读取时，系统会帮我们摘除IP头的信息，这样的话，我们就拿不到传输层以下的协议的相关信息了，所以我们不能使用普通的sock来编写ping程序，而要用原始套接字SOCK\_RAW，使用原始套接字的话，当我们想sock请求数据的时候，系统不会对sock做过多处理，我们最低可以拿到数据链路层的数据，因此在网络层的IP报头的数据也可以拿到了。

整个ping程序的基本流程如下：   
创建RAW套接字->接收待ping地址和参数->提取参数->根据参数指向相应的ping（普通的ping4次，还是无限循环ping）->向待ping地址发送一个ICMP回显请求报文->接收回复报文->输出相应信息。当中其实还有一个处理无限ping中捕获Ctrl+C退出命令的信号函数，那个其实并不影响大流程就不在流程上说了。

因为也就8个API就可以实现ping程序了，所以我打算一个一个API的放出来，具体的工程代码就放在github上，有需要的人自己拿吧。

参数检查和提取

char \*isParamEmpty(char \*buffer, char \*param)

{

char \*temp = NULL;

temp = buffer;

while (\*temp != '\0')

{

if (\*temp == ' ')

{

\*temp = '\0';

param = ++temp;

}

temp++;

}

return param;

}

计算校验和

uint16\_t getCheckSum(void \* protocol, char \* type)

{

uint32\_t checkSum = 0;

uint16\_t\* word = (uint16\_t\*)protocol;

uint32\_t size = 0;

if (type == "ICMP") {//计算有多少个字节

size = (sizeof(ICMPrecvReq));

}

while (size >1)//用32位变量来存是因为要存储16位数相加可能发生的溢出情况，将溢出的最高位最后加到16位的最低位上

{

checkSum += \*word++;

size -=2;

}

if (size == 1)

{

checkSum += \*(uint8\_t\*)word;

}

//二进制反码求和运算，先取反在相加和先相加在取反的结果是一样的，所以先全部相加在取反

//计算加上溢出后的结果

while (checkSum >> 16) checkSum = (checkSum >> 16) + (checkSum & 0xffff);

//取反

return (~checkSum);

}

ping流程选择

bool ping(const char \* dstIPaddr,const char\* param)

{

SOCKET rawSocket;//socket

sockaddr\_in srcAddr;//socket源地址

sockaddr\_in dstAddr;//socket目的地址

int Ret;//捕获状态值

char TTL = '0';//跳数

//生成一个套接字

//TCP/IP协议族,RAW模式，ICMP协议

//RAW创建的是一个原始套接字，最低可以访问到数据链路层的数据，也就是说在网络层的IP头的数据也可以拿到了。

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

//设置目标IP地址

dstAddr.sin\_addr.S\_un.S\_addr = inet\_addr(dstIPaddr);

//端口

dstAddr.sin\_port = htons(0);

//协议族

dstAddr.sin\_family = AF\_INET;

//提示信息

std::cout << "正在 Ping " << inet\_ntoa(dstAddr.sin\_addr) << " 具有 " << sizeof(ICMPrecvReq::data) << " 字节的数据:" << std::endl;

if(!strcmp(param,"-t"))

{

uint32\_t i=0;

while (keepping)

{

//无限执行ping操作

doPing(rawSocket, srcAddr, dstAddr,i);

//睡眠1ms

Sleep(1000);

i++;

}

keepping = 1;

}

else if(!strcmp(param,"no\_param"))

{ //执行4次ping

for (int i = 0; i < 4; i++)

{

doPing(rawSocket, srcAddr, dstAddr, i);

Sleep(1000);

}

}

Ret = closesocket(rawSocket);

if (Ret == SOCKET\_ERROR)

{

std::cerr << "socket关闭时发生错误:" << WSAGetLastError() << std::endl;

}

return true;

}

执行一次ping

void doPing(SOCKET & mysocket, sockaddr\_in & srcAddr, sockaddr\_in & dstAddr, int num)

{

uint32\_t timeSent;//发送时的时间

uint32\_t timeElapsed;//延迟时间

char TTL;//跳数

//发送ICMP回显请求

sendICMPReq(mysocket, dstAddr, num);

//等待数据

int Ret = waitForSocket(mysocket);

if (Ret == SOCKET\_ERROR)

{

std::cerr << "socket发生错误:" << WSAGetLastError() << std::endl;

return;

}

if (!Ret)

{

std::cout << "请求超时:" << std::endl;

return;

}

timeSent = readICMPanswer(mysocket, srcAddr, TTL);

if (timeSent != -1) {

timeElapsed = GetTickCount() - timeSent;

//输出信息，注意TTL值是ASCII码，要进行转换

std::cout << "来自 " << inet\_ntoa(srcAddr.sin\_addr) << " 的回复: 字节= " << sizeof(ICMPrecvReq::data) << " 时间= " << timeElapsed << "ms TTL= " << fabs((int)TTL) << std::endl;

}

else {

std::cout << "请求超时" << std::endl;

}

}

发送ICMP报文

bool sendICMPReq(SOCKET &mysocket,sockaddr\_in &dstAddr,unsigned int num)

{

//创建ICMP请求回显报文

//设置回显请求

ICMPrecvReq myIcmp;//ICMP请求报文

myIcmp.icmphead.type = 8;

myIcmp.icmphead.code = 0;

//设置初始检验和为0

myIcmp.icmphead.checkSum = 0;

//获得一个进程标识

myIcmp.icmphead.ident = (uint16\_t)GetCurrentProcessId();

//设置当前序号为0

myIcmp.icmphead.seqNum = ++num;

//保存发送时间

myIcmp.timeStamp = GetTickCount();

//计算并且保存校验和

myIcmp.icmphead.checkSum = getCheckSum((void\*)&myIcmp, "ICMP");

//发送报文

int Ret = sendto(mysocket, (char\*)&myIcmp, sizeof(ICMPrecvReq), 0, (sockaddr\*)&dstAddr, sizeof(sockaddr\_in));

if (Ret == SOCKET\_ERROR)

{

std::cerr << "socket 发送错误:" <<WSAGetLastError()<< std::endl;

return false;

}

return true;

}

等待ICMP报文的回复

int waitForSocket(SOCKET &mysocket)

{

//5S 等待套接字是否由数据

timeval timeOut;

fd\_set readfd;

readfd.fd\_count = 1;

readfd.fd\_array[0] = mysocket;

timeOut.tv\_sec = 5;

timeOut.tv\_usec = 0;

return (select(1, &readfd, NULL, NULL, &timeOut));

}

读取回复的ICMP报文

uint32\_t readICMPanswer(SOCKET &mysocket, sockaddr\_in &srcAddr, char &TTL)

{

ICMPansReply icmpReply;//接收应答报文

int addrLen = sizeof(sockaddr\_in);

//接收应答

int Ret = recvfrom(mysocket, (char\*)&icmpReply, sizeof(ICMPansReply), 0, (sockaddr\*)&srcAddr, &addrLen);

if (Ret == SOCKET\_ERROR)

{

std::cerr << "socket 接收错误:" << WSAGetLastError() << std::endl;

return false;

}

//读取校验并重新计算对比

uint16\_t checkSum = icmpReply.icmpanswer.icmphead.checkSum;

//因为发出去的时候计算的校验和是0

icmpReply.icmpanswer.icmphead.checkSum = 0;

//重新计算

if (checkSum== getCheckSum((void\*)&icmpReply.icmpanswer, "ICMP")) {

//获取TTL值

TTL = icmpReply.iphead.timetoLive;

return icmpReply.icmpanswer.timeStamp;

}

return -1;

}

处理退出信号函数

static volatile int keepping = 1;

//捕获终止信号函数,专门处理无限ping时的操作

void get\_ctrl\_stop(int signal)

{

if (signal == SIGINT)

{

keepping = 0;

}

}

如果要全部代码的话，可以在下面的链接里拿   
[ping程序完整工程](https://github.com/kitonchen/ping-procedure)

版权声明：本文为博主原创文章，未经博主允许不得转载。 https://blog.csdn.net/chenpidaxia/article/details/77803388