

用C语言实现Ping程序功能

梁俊辉 (zeusliang@21cn.com)

XML error: Please enter a value for the author element's jobtitle attribute, or the company-name element, or both.

简介： 大部分人用ping命令只是作为查看另一个系统的网络连接是否正常的一种简单方法。在这篇文章中，作者将介绍如何用C语言编写一个模拟ping命令功能的程序。

发布日期： 2001 年 10 月 01 日

级别： 初级

访问情况 2386 次浏览

建议:0 (添加评论)

★★★★★ 平均分 （共 1 个评分 ）

ping命令是用来查看网络上另一个主机系统的网络连接是否正常的工具。ping命令的工作原理是：向网络上的另一个主机系统发送ICMP报文，如果指定系统得到了报文，它将把报文一模一样地传回给发送者，这有点象潜水艇声纳系统中使用的发声装置。

例如，在Linux终端上执行ping localhost命令将会看到以下结果：

```
PING localhost.localdomain (127.0.0.1) from 127.0.0.1 : 56(84) bytes of data.
64 bytes from localhost.localdomain (127.0.0.1): icmp_seq=0 ttl=255 time=112 usec
64 bytes from localhost.localdomain (127.0.0.1): icmp_seq=1 ttl=255 time=79 usec
64 bytes from localhost.localdomain (127.0.0.1): icmp_seq=2 ttl=255 time=78 usec
64 bytes from localhost.localdomain (127.0.0.1): icmp_seq=3 ttl=255 time=82 usec
--- localhost.localdomain ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max/mdev = 0.078/0.087/0.112/0.018 ms
```

由上面的执行结果可以看到，ping命令执行后显示出被测试系统主机名和相应IP地址、返回给当前主机的ICMP报文顺序号、ttl生存时间和往返时间rtt（单位是毫秒，即千分之一秒）。要写一个模拟ping命令，这些信息有启示作用。

要真正了解ping命令实现原理，就要了解ping命令所使用到的TCP/IP协议。

ICMP(Internet Control Message,网际控制报文协议)是为网关和目标主机而提供了一种差错控制机制，使它们在遇到差错时能把错误报告给报文源发方。ICMP协议是IP层的一个协议，但是由于差错报告在发送给报文源发方时可能也要经过若干子网，因此牵涉到路由选择等问题，所以ICMP报文需通过IP协议来发送。ICMP数据报的数据发送前需要两级封装：首先添加ICMP报头形成ICMP报文，再添加IP报头形成IP数据报。如下图所示



IP报头格式

由于IP层协议是一种点对点的协议，而非端对端的协议，它提供无连接的数据报服务，没有端口的概念，因此很少使用bind()和connect()函数，若有使用也只是用于设置IP地址。发送数据使用sendto()函数，接收数据使用recvfrom()函数。IP报头格式如下图：

版本号 VER	IP 报头长度 IHL	服务类型 TOS	数据报长度 TL
报文标志 ID	报文标志 F	分段偏移量 FO	
生存时间 TTL	协议号 PORT	报头校验和	
源地址			
目标地址			
任选项和填充位			

在Linux中，IP报头格式数据结构(<netinet/ip.h>)定义如下：

```
struct ip
{
    #if __BYTE_ORDER == __LITTLE_ENDIAN
        unsigned int ip_hl:4;          /* header length */
        unsigned int ip_v:4;          /* version */
    #endif
    #if __BYTE_ORDER == __BIG_ENDIAN
        unsigned int ip_v:4;          /* version */
        unsigned int ip_hl:4;          /* header length */
    #endif
    u_int8_t ip_tos;                  /* type of service */
    u_short ip_len;                   /* total length */
    u_short ip_id;                    /* identification */
    u_short ip_off;                   /* fragment offset field */
    #define IP_RF 0x8000                /* reserved fragment flag */
    #define IP_DF 0x4000                /* dont fragment flag */
    #define IP_MF 0x2000                /* more fragments flag */
```

```
#define IP_OFFMASK 0x1fff          /* mask for fragmenting bits */
u_int8_t ip_ttl;                  /* time to live */
u_int8_t ip_p;                    /* protocol */
u_short ip_sum;                   /* checksum */
struct in_addr ip_src, ip_dst; /* source and dest address */
};
```

其中ping程序只使用以下数据:

- IP报头长度IHL (Internet Header Length) □□□□以4字节为一个单位来记录IP报头的长度, 是上述IP数据结构的ip_hl变量。
- 生存时间TTL (Time To Live) □□□□以秒为单位, 指出IP数据报能在网络上停留的最长时间, 其值由发送方设定, 并在经过路由的每一个节点时减一, 当该值为0时, 数据报将被丢弃, 是上述IP数据结构的ip_ttl变量。

ICMP报头格式

ICMP报文分为两种, 一是错误报告报文, 二是查询报文。每个ICMP报头均包含类型、编码和校验和这三项内容, 长度为8位, 8位和16位, 其余选项则随ICMP的功能不同而不同。

Ping命令只使用众多ICMP报文中的两种: "请求回送(ICMP_ECHO)和"请求回应(ICMP_ECHOREPLY)。在Linux中定义如下:

```
#define ICMP_ECHO 0
#define ICMP_ECHOREPLY 8
```

这两种ICMP类型报头格式如下:

类型 TYPE(8或0)	编码 CODE (没有使用)	校验和 CHECKSUM
标志符 Identifier	顺序号 Sequence NO	

在Linux中ICMP数据结构(<netinet/ip_icmp.h>)定义如下:

```
struct icmp
{
    u_int8_t icmp_type; /* type of message, see below */
    u_int8_t icmp_code; /* type sub code */
    u_int16_t icmp_cksum; /* ones complement checksum of struct */
    union
    {
        {
            u_char ih_pptr; /* ICMP_PARAMPROB */
            struct in_addr ih_gwaddr; /* gateway address */
            struct ih_idseq /* echo datagram */
            {
                u_int16_t icd_id;
                u_int16_t icd_seq;
            } ih_idseq;
            u_int32_t ih_void;
            /* ICMP_UNREACH_NEEDFRAG -- Path MTU Discovery (RFC1191) */
            struct ih_pmtu
            {
                u_int16_t ipm_void;
                u_int16_t ipm_nextmtu;
            } ih_pmtu;
            struct ih_rtradv
            {
                u_int8_t irt_num_addrs;
                u_int8_t irt_wpa;
                u_int16_t irt_lifetime;
            } ih_rtradv;
        } icmp_hun;
#define icmp_pptr icmp_hun.ih_pptr
#define icmp_gwaddr icmp_hun.ih_gwaddr
#define icmp_id icmp_hun.ih_idseq.icd_id
#define icmp_seq icmp_hun.ih_idseq.icd_seq
#define icmp_void icmp_hun.ih_void
#define icmp_pmvoid icmp_hun.ih_pmtu.ipm_void
#define icmp_nextmtu icmp_hun.ih_pmtu.ipm_nextmtu
#define icmp_num_addrs icmp_hun.ih_rtradv.irt_num_addrs
#define icmp_wpa icmp_hun.ih_rtradv.irt_wpa
#define icmp_lifetime icmp_hun.ih_rtradv.irt_lifetime
        union
        {
            struct
            {
                {
                    u_int32_t its_otime;
                    u_int32_t its_rtime;
                    u_int32_t its_ttime;
                } id_ts;
                struct
                {
                    struct ip idi_ip;
                    /* options and then 64 bits of data */
                } id_ip;
                struct icmp_ra_addr id_radv;
                u_int32_t id_mask;
            }
        }
    }
};
```

```

    u_int8_t    id_data[1];
} icmp_dun;
#define icmp_otime    icmp_dun.id_ts.its_otime
#define icmp_rtime    icmp_dun.id_ts.its_rtime
#define icmp_ttime    icmp_dun.id_ts.its_ttime
#define icmp_ip       icmp_dun.id_ip.idi_ip
#define icmp_radv     icmp_dun.id_radv
#define icmp_mask     icmp_dun.id_mask
#define icmp_data     icmp_dun.id_data
};

```

使用宏定义令表达更简洁,其中ICMP报头为 8 字节,数据报长度最大为64K字节。

1. 校验和算法□D□D这一算法称为网际校验和算法,把被校验的数据 1 6 位进行累加,然后取反码,若数据字节长度为奇数,则数据尾部补一个字节的 0 以凑成偶数。此算法适用于IPv4、ICMPv4、IGMPV4、ICMPv6、UDP和TCP校验和,更详细的信息请参考RFC1071,校验和字段为上述ICMP数据结构的icmp_cksum变量。
2. 标识符□D□D用于唯一标识ICMP报文,为上述ICMP数据结构的icmp_id宏所指的变量。
3. 顺序号□D□Dping命令的icmp_seq便由这里读出,代表ICMP报文的发送顺序,为上述ICMP数据结构的icmp_seq宏所指的变量。

ICMP数据报

Ping命令中需要显示的信息,包括icmp_seq和ttl都已有实现的办法,但还缺rtt往返时间。为了实现这一功能,可利用ICMP数据报携带一个时间戳。使用以下函数生成时间戳:

```

#include
int gettimeofday(struct timeval *tp,void *tzp)
其中timeval结构如下:
    struct timeval{
        long tv_sec;
        long tv_usec;
    }

```

其中tv_sec为秒数, tv_usec微秒数。在发送和接收报文时由gettimeofday分别生成两个timeval结构,两者之差即为往返时间,即ICMP报文发送与接收的时间差,而timeval结构由ICMP数据报携带,tzp指针表示时区,一般都不使用,赋NULL值。

数据统计

系统自带的ping命令当它接送完所有ICMP报文后,会对所有发送和所有接收的ICMP报文进行统计,从而计算ICMP报文丢失的比率。为达此目的,定义两个全局变量:接收计数器和发送计数器,用于记录ICMP报文接受和发送数目。丢失数目=发送总数-接收总数,丢失比率=丢失数目/发送总数。

现给出模拟Ping程序功能的代码如下:

```

/*****
 * 作者:梁俊辉
 * 时间:2001年10月
 * 名称: myping.c
 * 说明: 本程序用于演示ping命令的实现原理
 *****/
#include <stdio.h>
#include <signal.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <unistd.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/ip_icmp.h>
#include <netdb.h>
#include <setjmp.h>
#include <errno.h>
#define PACKET_SIZE    4096
#define MAX_WAIT_TIME    5
#define MAX_NO_PACKETS    3
char sendpacket[PACKET_SIZE];
char recvpacket[PACKET_SIZE];
int sockfd,datalen=56;
int nsend=0,nreceived=0;
struct sockaddr_in dest_addr;
pid_t pid;
struct sockaddr_in from;
struct timeval tvrecv;
void statistics(int signo);
unsigned short cal_chksum(unsigned short *addr,int len);
int pack(int pack_no);
void send_packet(void);
void recv_packet(void);
int unpack(char *buf,int len);
void tv_sub(struct timeval *out,struct timeval *in);
void statistics(int signo)
{
    printf("\n-----PING statistics-----\n");
    printf("%d packets transmitted, %d received , %%d lost\n",nsend,nreceived,

```

```

        (nsend-nreceived)/nsend*100);
    close(sockfd);
    exit(1);
}
/*校验和算法*/
unsigned short cal_chksum(unsigned short *addr,int len)
{
    int nleft=len;
    int sum=0;
    unsigned short *w=addr;
    unsigned short answer=0;

    /*把ICMP报头二进制数据以2字节为单位累加起来*/
    while(nleft>1)
    {
        sum+=*w++;
        nleft-=2;
    }
    /*若ICMP报头为奇数个字节,会剩下最后一字节。把最后一个字节视为一个2字节数据的高字节,这个2字节数据的低字节为0,继续累加*/
    if( nleft==1)
    {
        *(unsigned char *)(&answer)=*(unsigned char *)w;
        sum+=answer;
    }
    sum=(sum>>16)+(sum&0xffff);
    sum+=(sum>>16);
    answer=~sum;
    return answer;
}
/*设置ICMP报头*/
int pack(int pack_no)
{
    int i,packsize;
    struct icmp *icmp;
    struct timeval *tval;
    icmp=(struct icmp*)sendpacket;
    icmp->icmp_type=ICMP_ECHO;
    icmp->icmp_code=0;
    icmp->icmp_cksum=0;
    icmp->icmp_seq=pack_no;
    icmp->icmp_id=pid;
    packsize=8+datalen;
    tval= (struct timeval *)icmp->icmp_data;
    gettimeofday(tval,NULL); /*记录发送时间*/
    icmp->icmp_cksum=cal_chksum( (unsigned short *)icmp,packsize); /*校验算法*/
    return packsize;
}
/*发送三个ICMP报文*/
void send_packet()
{
    int packsize;
    while( nsend<MAX_NO_PACKETS)
    {
        nsend++;
        packsize=pack(nsend); /*设置ICMP报头*/
        if( sendto(sockfd,sendpacket,packsize,0,
            (struct sockaddr *)&dest_addr,sizeof(dest_addr) )<0 )
        {
            perror("sendto error");
            continue;
        }
        sleep(1); /*每隔一秒发送一个ICMP报文*/
    }
}
/*接收所有ICMP报文*/
void recv_packet()
{
    int n,fromlen;
    extern int errno;
    signal(SIGALRM,statistics);
    fromlen=sizeof(from);
    while( nreceived<nsend)
    {
        alarm(MAX_WAIT_TIME);
        if( (n=recvfrom(sockfd,recvpacket,sizeof(recvpacket),0,
            (struct sockaddr *)&from,&fromlen)) <0)
        {
            if(errno==EINTR)continue;
            perror("recvfrom error");
            continue;
        }
        gettimeofday(&tvrecv,NULL); /*记录接收时间*/
        if(unpack(recvpacket,n)==-1)continue;
        nreceived++;
    }
}
/*剥去ICMP报头*/
int unpack(char *buf,int len)
{
    int i,iphdrlen;
    struct ip *ip;
    struct icmp *icmp;
    struct timeval *tvsend;
    double rtt;
    ip=(struct ip *)buf;
    iphdrlen=ip->ip_hl<<2; /*求ip报头长度,即ip报头的长度标志乘4*/
    icmp=(struct icmp *) (buf+iphdrlen); /*越过ip报头,指向ICMP报头*/
    len-=iphdrlen; /*ICMP报头及ICMP数据报的总长度*/
    if( len<8) /*小于ICMP报头长度则不合理*/
    {
        printf("ICMP packets\'s length is less than 8\n");
    }
}

```

```

        return -1;
    }
    /*确保所接收的是我所发的ICMP的回应*/
    if( (icmp->icmp_type==ICMP_ECHOREPLY) && (icmp->icmp_id==pid) )
    {
        tvsend=(struct timeval *)icmp->icmp_data;
        tv_sub(&tvrecv,tvsend); /*接收和发送的时间差*/
        rtt=tvrecv.tv_sec*1000+tvrecv.tv_usec/1000; /*以毫秒为单位计算rtt*/
        /*显示相关信息*/
        printf("%d byte from %s: icmp_seq=%u ttl=%d rtt=%.3f ms\n",
            len,
            inet_ntoa(from.sin_addr),
            icmp->icmp_seq,
            ip->ip_ttl,
            rtt);
    }
    else return -1;
}
main(int argc,char *argv[])
{
    struct hostent *host;
    struct protoent *protocol;
    unsigned long inaddr=0l;
    int waittime=MAX_WAIT_TIME;
    int size=50*1024;
    if(argc<2)
    {
        printf("usage:%s hostname/IP address\n",argv[0]);
        exit(1);
    }
    if( (protocol=getprotobyname("icmp")) ==NULL)
    {
        perror("getprotobyname");
        exit(1);
    }
    /*生成使用ICMP的原始套接字,这种套接字只有root才能生成*/
    if( (sockfd=socket(AF_INET,SOCK_RAW,protocol->p_proto) )<0)
    {
        perror("socket error");
        exit(1);
    }
    /* 回收root权限,设置当前用户权限*/
    setuid(getuid());
    /*扩大套接字接收缓冲区到50K这样做主要为了减小接收缓冲区溢出的
    的可能性,若无意中ping一个广播地址或多播地址,将会引来大量应答*/
    setsockopt(sockfd,SOL_SOCKET,SO_RCVBUF,&size,sizeof(size) );
    bzero(&dest_addr,sizeof(dest_addr));
    dest_addr.sin_family=AF_INET;
    /*判断是主机名还是ip地址*/
    if( inaddr=inet_addr(argv[1])==INADDR_NONE)
    {
        if((host=gethostbyname(argv[1]))==NULL) /*是主机名*/
        {
            perror("gethostbyname error");
            exit(1);
        }
        memcpy( (char *)&dest_addr.sin_addr,host->h_addr,host->h_length);
    }
    else /*是ip地址*/
        memcpy( (char *)&dest_addr,(char *)&inaddr,host->h_length);
    /*获取main的进程id,用于设置ICMP的标志符*/
    pid=getpid();
    printf("PING %s(%s): %d bytes data in ICMP packets.\n",argv[1],
        inet_ntoa(dest_addr.sin_addr),datalen);
    send_packet(); /*发送所有ICMP报文*/
    recv_packet(); /*接收所有ICMP报文*/
    statistics(SIGALRM); /*进行统计*/
    return 0;
}
/*两个timeval结构相减*/
void tv_sub(struct timeval *out,struct timeval *in)
{
    if( (out->tv_usec=-in->tv_usec)<0)
    {
        --out->tv_sec;
        out->tv_usec+=1000000;
    }
    out->tv_sec-=in->tv_sec;
}
/*----- The End -----*/

```

特别注意

只有root用户才能利用socket()函数生成原始套接字,要让Linux的一般用户能执行以上程序,需进行如下的特别操作:

用root登陆,编译以上程序: gcc -o myping myping.c, 其目的有二: 一是编译,二是让myping属于root用户。

再执行chmod u+s myping, 目的是把myping程序设成SUID的属性。

退出root, 用一般用户登陆, 执行./myping www.cn.ibm.com, 有以下执行结果:

```

PING www.cn.ibm.com(202.95.2.148): 56 bytes data in ICMP packets.
64 byte from 202.95.2.148: icmp_seq=1 ttl=242 rtt=3029.000 ms
64 byte from 202.95.2.148: icmp_seq=2 ttl=242 rtt=2020.000 ms
64 byte from 202.95.2.148: icmp_seq=3 ttl=242 rtt=1010.000 ms

```

```
-----PING statistics-----  
3 packets transmitted, 3 received , %0 lost
```

由于myping.c是发送完所有的ICMP报文才去接收，因此第一、第二和第三个ICMP报文的往返时间依此是3秒， 2 秒， 1 秒，上述结果中rtt信息正反映这一事实。

关于作者

梁俊辉,对Linux的网络应用和程序设计有浓厚兴趣,并且专注于这一方面研究，在IBM developerWorks [D](#)Linux专区上发表过《NEWT程序设计指南》一文。

商标 | [My developerWorks 使用条款与条件](#)