Ping（gcc实现）

1、实验思路

本实验使用gcc环境模拟实现Ping的功能，由于Ping是基于ICMP报文的，所以我们需要自己构造ICMP报文，由于牵扯到路由选择的问题，要将ICMP报文通过IP协议来发送。向对方发送使用sendto，接收使用recvfrom。ICMP报文的类型分为ICMP_ECHO和ICMP_ECHOREPLY，分别对应0和8，在构造ICMP报文发送时，需要指定icmp->type=ICMP_ECHO和序号，同时使用校验和函数计算校验和并填充校验和字段。在接收对方的回应报文时，如果ICMP的type字段为ICMP_ECHOREPLY以及序号等于我们定义的序号，则表明接收到的是自己发的ICMP的回应。然后将接收到的报文分别转换为IP格式和ICMP格式（跳过IP头部），通过IP结构体可以得到ttl，通过ICMP结构体可以得到seq序号。rtt的获取可以使用timeval结构体来计算得出。

2、实验环境及步骤

- Linux下gcc环境
- gcc -o myping myping.c
- ./myping 10.103.89.201（这是我的主机IP）
- ./myping 192.168.76.33（不存在的IP）

3、实验结果及分析

1、./myping 10.103.89.201

```
[root@localhost 桌面]# ./myping  10.103.89.201
PING 10.103.89.201(10.103.89.201): 56 bytes data in ICMP packets.
64 byte from 10.103.89.201:  icmp_seq=1  ttl=128 rtt=3002.000 ms
64 byte from 10.103.89.201:  icmp_seq=2  ttl=128 rtt=2000.000 ms
64 byte from 10.103.89.201:  icmp_seq=3  ttl=128 rtt=1000.000 ms

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

ping主机IP可以ping通，seq为自己设置的3个序号，信息显示均正确。

2、./myping 192.168.76.33

```
[root@localhost 桌面]# ./myping  192.168.76.33
PING 192.168.76.33(192.168.76.33): 56 bytes data in ICMP packets.

--------------------PING statistics-------------------
3 packets transmitted, 0 received , %100 lost
```

ping不存在的IP不能ping通。

从实验结果可以看出，该实验模拟实现了ping的功能。

4、实验程序

- ICMP结构体

```
typedef struct _icmphdr{
unsigned char i_type; //8位类型，本实验用 8: ECHO 0:ECHO REPLY
unsigned char i_code; //8位代码，本实验置零
unsigned short i_cksum; //16位校验和，从TYPE开始,直到最后一位用户数据,如果为字节数为奇数则补充一位
unsigned short i_id ; //识别号（一般用进程号作为识别号），用于匹配ECHO和ECHO REPLY包
unsigned short i_seq ; //报文序列号，用于标记ECHO报文顺序
unsigned int timestamp; //时间戳
}ICMP_HEADER;
```

- 总体代码

```
#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 packetsize;
    while( nsend<MAX_NO_PACKETS) {
        nsend++;
        packetsize=pack(nsend); /*设置ICMP报头*/
        if( sendto(sockfd,sendpacket,packetsize,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;
}
int 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;

    if((host=gethostbyname(argv[1]) )==NULL) { /*是主机名*/
        perror("gethostbyname error");
        exit(1);
    }
    dest_addr.sin_addr=*((struct in_addr *)host->h_addr);
    /*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 -----------*/
```