<https://blog.csdn.net/xujuan0815/article/details/38397473>

相信大家一定遇到过上不了网的情形，都知道用个ping命令。这不小王就是这样的女孩，老是上不了网，老是找我，我就先ping一下，逐步找找问题在哪儿，有的放矢，不至于盲目抓瞎(说心里话，我真不愿意帮小王弄，每次弄好了，她就和那个叫寒烟的Q友，使劲聊天，唉，心里哇凉啊.）都说实践是最好的老师，小王不和老师我聊天，说明我这个老师没做好，没关系，我有技术我怕谁，抓住小王的心，我还是有把握的。不知谁问：刀是什么样的刀，朱锈断环刀；剑是什么样的剑，流光翠月剑！不要问我为啥这大反差，没办法，谁叫小王喜欢呢。唉，不说了，小王把眨巴着眼催我快点呢.

     好，今天就给大家讲讲有关linux下ping程序的实现，以后谁再找我修网，对不起，该不奉陪，我只为小王，嘿嘿。

     先来说说ping程序的原理吧，其实挺简单，就是一个主机系统向另外一个主机系统说：I love you(ICMP报文)，然后那个主机如果相信你或者说想和你通信，和你心知心，那它就把收到的I love you(ICMP)报文原样返回.好嘛,源主机收到这个回应后，就happy了，因为对方是和自己心连心的。如果对方没有收到这个消息，或者对你不感冒，不愿意理你，不回你这个报文，或者说些不知云是云雾是雾的话，对不起啦，感情是两个人的事情哦.

     要想深刻了解，需有入目三分的实力，这个ping也一样，咱们先来看看它采用的TCP/IP协议，我刚说了，它发送的是ICMP回显请求，回答的是回显应答报文。谈起这个ICMP(Internet Control Message,网际控制报文协议)是为网关和目标主机而提供的一种差错控制机制，使它们在遇到差错时能把错误报告给报文源发方.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 |
| ICMP报头 |
| ICMP数据报 |

     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。

     TCP/IP的经典大作《TCP/IP协议详解.卷一》清晰的告诉我，IP报头格式如下：



    详细的，小王那懒的人都知道翻翻上面提到的书，我也就不详细介绍了，我这里给出linux中的数据结构实现：

|  |
| --- |
| 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 \*/  }; |

     别看这多，其实ping程序用到的没几个：

    (1)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。

    (2)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报文分为两种：查询报文和差错报文。每个ICMP报头均包含类型、编码和校验和这三项内容，其余选项则随ICMP的功能不同而不同。ICMP报文格式如下：

ICMP

    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 |

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

|  |
| --- |
| linux中ICMP数据结构 |

    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;  } |

    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。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：

|  |
| --- |
| 必要的头文件,宏定义和函数说明 |

|  |
| --- |
| 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) //发送MAX\_NO\_PACKETS个ICMP报文  {  nsend++;  packetsize=pack(nsend); /\*设置ICMP报头\*/  //sendpacket为要发送的内容，由pack()函数设定，dest\_addr是目的地址，  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**()用来设置信号SIGALRM在经过参数seconds指定的秒数后传送给目前的进程  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;  //求ip报头长度,即ip报头的长度标志乘4,头长度指明头中包含的4字节字的个数。可接受  //的最小值是5，最大值是15  iphdrlen=ip->ip\_hl<<2;  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; //#define MAX\_WAIT\_TIME 5  int size=50\*1024;  if(argc<2)  { printf("usage:%s hostname/IP address\n",argv[0]);  exit(1);  }  //getprotobyname("icmp")返回对应于给定协议名的包含名字和协议号的protoent结构指针。  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\*/  dest\_addr.sin\_addr.s\_addr = inet\_addr(argv[1]);  /\*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;  } |

好了，编译，运行。