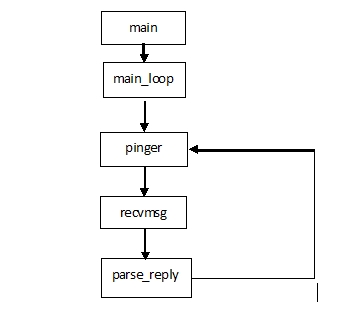
<https://blog.csdn.net/rosetta/article/details/7343533>

第一回分析源码

程序流程一览图

|  |
| --- |
|  |
|  |



感觉有点简单吧，ping的核心就是上面的图了，而其它都是对选项的处理，不仅仅是ping源码，其它开源代码一般都是一堆选项，如果一开始就被那些对分析程序核心没有帮助的选项所困扰，那么分析一个程序需要的时间和效率就可想而知。当然ping的很多选项都是值得分析的。

这个ping源码控制发包的时间间隔是通过main\_loop()比较复杂的时间计算控制，因为它要控制诸如-l这样的选项，如果纯粹是控制时间，那么注册个SIGALRM信号处理函数，再加个1秒的定时器alarm(1)或者就直接sleep(1)就可以很轻松的实现。-w选项就是通过alarm完成的,SIGALRM信号处理函数是在main()->setup()中设置的，具体请看setup函数分析。

1，准备工作，获得ping源码并学会编译

会编译了看代码其实就是后续的活，网上找的方法，挺好用……

[root@xxx study\_2]# type ping

ping is /bin/ping

[root@xxx study\_2]# rpm -qf /bin/ping

iputils-20020927-46.el5

从而得知ping属于iputils包，去下iputils源码包即可，其它linux 命令源码获取方法类似。

[root@xxx iputils]# make

Please, set correct KERNEL\_INCLUDE

make: \*\*\* [check-kernel] 错误 1

反正是一堆错误，改了这个来了那个，看了INSTALL说是被定制过的linux头文件可能会不支持，反正只要能编出ping就行，改了下Makefile，把有关KERNEL\_INCLUDE的都屏蔽了，编译成功。

2，源码分析

    只分析实现最简单的不带选项的ping程序。也花了一个星期的业余时间。

 112main(int argc, char \*\*argv)

 113{

 114    struct hostent \*hp;

 115    int ch, hold, packlen;

 116    int socket\_errno;

 117    u\_char \*packet;

 118    char \*target, hnamebuf[MAXHOSTNAMELEN];

 119    char rspace[3 + 4 \* NROUTES + 1];  /\* record route space \*/

 120#ifdef DO\_IPSEC

 121    char \*policy\_string = NULL;

 122#endif

 123

 124    icmp\_sock = socket(AF\_INET, SOCK\_RAW, IPPROTO\_ICMP); //创建icmp套接字

 125    socket\_errno = errno;

 126

 127    uid = getuid();//设置进程有效用户ID为实际用户uid，访问权限方面的知识。

 128    setuid(uid);

 129

 130    source.sin\_family = AF\_INET;

 131

 132    preload = 1;//这个参数默认为1，可由-l选项控制，一次发多个包而不管对方是否回应，其本质还是通过计算时间来实现，具体可看main\_loop().没兴趣不用管。

 133    while ((ch = getopt(argc, argv, COMMON\_OPTSTR "bRT:P:")) !=EOF) {

 134        switch(ch) {//getopt得好好学会。

 213        default:

 214            usage();

 215        }

 216    }

 217    argc -= optind;//可参考GETOPT(3) ，是个全局变量

 218    argv += optind;

 219

//这里对选项的处理，不管，option为0。

 220    printf("options :%d\n", options );

 221    if (argc == 0)

 222        usage();

 223    if (argc > 1) {

 224        if (options & F\_RROUTE)

 225            usage();

 226        else if (options & F\_TIMESTAMP) {

 227            if (ts\_type != IPOPT\_TS\_PRESPEC)

 228                 usage();

 229            if (argc > 5)

 230                 usage();

 231        } else {

 232            if (argc > 10)

 233                 usage();

 234            options |= F\_SOURCEROUTE;

 235         }

 236    }

//处理传给ping的参数

 237       while (argc > 0) {

//我只传一个目标IP，所以argc=1,这个循环只做一回，把IP地址给了target

 238        target = \*argv;

 239        printf("target:%s\n", target);

 240

 241        bzero((char \*)&whereto, sizeof(whereto));

 242        whereto.sin\_family = AF\_INET;

 243        if (inet\_aton(target, &whereto.sin\_addr) == 1) {//如果是IP地址

 244            hostname = target;

 245            if (argc == 1)

 246                 options |= F\_NUMERIC;

 247        } else {//如果是其它域名

 248            hp = gethostbyname(target);

 249            if (!hp) {//如果解析域名不成功，返回

 250                 fprintf(stderr, "ping:unknown host %s\n", target);

 251                 exit(2);

 252            }

 253            memcpy(&whereto.sin\_addr, hp->h\_addr, 4);

 254             strncpy(hnamebuf, hp->h\_name,sizeof(hnamebuf) - 1);

 255            hnamebuf[sizeof(hnamebuf) - 1] = 0;

 256            hostname = hnamebuf;

 257        }

 258        if (argc > 1)

 259            route[nroute++] = whereto.sin\_addr.s\_addr;

 260        argc--;

 261        argv++;

 262    }

 //这部分只是对目标IP的一个connect尝试，或者判断是否是广播地址,connect失败退出，

//如果成功再获取本机struct sockaddr结构到source，

 264    if (source.sin\_addr.s\_addr == 0) {

 265        int alen;

 266        struct sockaddr\_in dst = whereto;

 267        int probe\_fd = socket(AF\_INET, SOCK\_DGRAM, 0);

 268

 269        if (probe\_fd < 0) {

 270            perror("socket");

 271            exit(2);

……

……

……

 322         alen = sizeof(source);

 323        if (getsockname(probe\_fd, (struct sockaddr\*)&source, &alen) ==-1) {

 324            perror("getsockname");

 325            exit(2);

 326        }

 327        source.sin\_port = 0;

 328        close(probe\_fd);

 329    } while (0);

 330

//如果目标IP为0.0.0.0，则赋值目标IP为127.0.0.1,即ping本地回环地址。

 331     if (whereto.sin\_addr.s\_addr == 0)

 332        whereto.sin\_addr.s\_addr = source.sin\_addr.s\_addr;

 333

//如果icmp\_sock套接字创建失败，返回。

 334    if (icmp\_sock < 0) {

 335        errno = socket\_errno;

 336        perror("ping: icmp open socket");

 337         exit(2);

 338    }

 339

//device是加-I选项选择出口接口，这回没加-I选项。

 338    if (device) {

……

……

 349    }

……

//这部分都是对一些选项处理, 省略。

……

//,

 446

 447    /\* Estimate memory eaten by single packet. It is rough estimate.

 448     \* Actually, for small datalen's it depends on kernel side a lot. \*/

 449    hold = datalen + 8;

 450    hold += ((hold+511)/512)\*(optlen + 20 + 16 + 64 + 160);

 451    sock\_setbufs(icmp\_sock, hold);//设置此套接字接收和发送缓冲区大小。

 515    printf("PING %s (%s) ", hostname, inet\_ntoa(whereto.sin\_addr));

 516    if (device || (options&F\_STRICTSOURCE))

 517        printf("from %s %s: ", inet\_ntoa(source.sin\_addr), device ?:"");

 518    printf("%d(%d) bytes of data.\n", datalen,datalen+8+optlen+20);

 519

 520    setup(icmp\_sock);

 521

 522    main\_loop(icmp\_sock, packet, packlen);

//到这里main就结束了。现在来分析setup和main\_loop。

//setup()函数

……

478    set\_signal(SIGINT, sigexit);

479    set\_signal(SIGALRM, sigexit);

480    set\_signal(SIGQUIT, sigstatus);

481

482    gettimeofday(&start\_time, NULL);

483

484    if (deadline) {

485        struct itimerval it;

486

487        it.it\_interval.tv\_sec = 0;

488        it.it\_interval.tv\_usec = 0;

489        it.it\_value.tv\_sec = deadline;

490        it.it\_value.tv\_usec = 0;

491        setitimer(ITIMER\_REAL, &it, NULL);

492    }

……

set\_signal(SIGALRM, sigexit);

static void sigexit(int signo)

{

   exiting = 1;

}

主要是注册了几个信号量。为-w选项设置定时器,如果超时，置exiting为1，而在main\_loop()的主循环中一开始就对这个变量做了判断，如果exiting为真跳出循环，程序退出。

for (;;) {

       /\* Check exit conditions. \*/

       if (exiting)

           break;

……

……

}

//main\_loop()函数,关键东西都在这里。

504 void main\_loop(int icmp\_sock, \_\_u8\*packet, int packlen)

505 {

506    char addrbuf[128];

507    char ans\_data[4096];

508    struct iovec iov;

509    struct msghdr msg;

510    struct cmsghdr \*c;

511    int cc;

512    int next;

513    int polling;

514

515    iov.iov\_base = (char \*)packet;

516

517    for (;;) {

518        /\* Check exit conditions. \*/

519        if (exiting)

520             break;

521        if (npackets && nreceived + nerrors >= npackets)

522             break;

523        if (deadline && nerrors)

524             break;

525        /\* Check for and do special actions. \*/

526        if (status\_snapshot)

527             status();

528

529        /\* Send probes scheduled to this time. \*/

530        do {

531             next = pinger();//发送ICMP包

532             next = schedule\_exit(next);

533        } while (next <= 0);

534

535        /\* "next" is time to send next probe, if positive.

536         \* If next<=0 send now or assoon as possible. \*/

537

538        /\* Technical part. Looks wicked. Could be dropped,

539         \* if everyone used the newest kernel. :-)

540         \* Its purpose is:

541         \* 1. Provide intervals less than resolution of scheduler.

542         \*    Solution: spinning.

543         \* 2. Avoid use of poll(), when recvmsg() can provide

544         \*    timed waiting (SO\_RCVTIMEO).\*/

545        polling = 0;

546        if ((options & (F\_ADAPTIVE|F\_FLOOD\_POLL)) ||next<SCHINT(interval)) {//控制发送时间间隔。

547             int recv\_expected = in\_flight();

548

549             /\* If we are here, recvmsg() isunable to wait for

550              \* required timeout. \*/

551             if (1000\*next <=1000000/(int)HZ) {

552                /\* Very shorttimeout... So, if we wait for

553                  \* something, we sleep forMININTERVAL.

554                  \* Otherwise, spin! \*/

555                 if (recv\_expected) {

556                     next = MININTERVAL;

557                } else {

558                     next = 0;

559                     /\* When spinning, noreasons to poll.

560                      \* Use nonblockingrecvmsg() instead. \*/

561                     polling = MSG\_DONTWAIT;

562                     /\* But yield yet. \*/

563                     sched\_yield();

564                 }

565             }

566

567             if (!polling &&

568                 ((options &(F\_ADAPTIVE|F\_FLOOD\_POLL)) || interval)) {

569                 struct pollfd pset;

570                pset.fd = icmp\_sock;

571                 pset.events = POLLIN|POLLERR;

572                 pset.revents = 0;

573                 if (poll(&pset, 1, next)< 1 ||

574                    !(pset.revents&(POLLIN|POLLERR)))

575                     continue;

576                 polling = MSG\_DONTWAIT;

577             }

578        }

579

580        for (;;) {//接收ICMP\_ECHOREPLY回应包

581             struct timeval \*recv\_timep = NULL;

582             struct timeval recv\_time;

583             int not\_ours = 0; /\* Raw socketcan receive messages

584                        \* destined to otherrunning pings. \*/

585

586             iov.iov\_len = packlen;

587             msg.msg\_name = addrbuf;

588             msg.msg\_namelen = sizeof(addrbuf);

589             msg.msg\_iov = &iov;

590             msg.msg\_iovlen = 1;

591             msg.msg\_control = ans\_data;

592             msg.msg\_controllen =sizeof(ans\_data);

593

594             cc = recvmsg(icmp\_sock, &msg,polling);//接收

595             polling = MSG\_DONTWAIT;

596

597             if (cc < 0) {

598                 if (errno == EAGAIN || errno== EINTR)

599                     break;

600                 if (!receive\_error\_msg()) {

601                     if (errno) {

602                         perror("ping: recvmsg");

603                         break;

604                     }

605                     not\_ours = 1;

606                 }

607             } else {

608

609 #ifdef SO\_TIMESTAMP

610      for (c = CMSG\_FIRSTHDR(&msg); c; c = CMSG\_NXTHDR(&msg, c)) {

611                     if (c->cmsg\_level !=SOL\_SOCKET ||

612                         c->cmsg\_type !=SO\_TIMESTAMP)

613                         continue;

614                     if (c->cmsg\_len <CMSG\_LEN(sizeof(struct timeval)))

615                         continue;

616                     recv\_timep = (structtimeval\*)CMSG\_DATA(c);

617                 }

618 #endif

619

620                 if ((options&F\_LATENCY) ||recv\_timep == NULL) {

621                     if((options&F\_LATENCY) ||

622                         ioctl(icmp\_sock, SIOCGSTAMP,&recv\_time))

623                        gettimeofday(&recv\_time, NULL);

624                     recv\_timep =&recv\_time;

625                 }

626

627                 not\_ours =parse\_reply(&msg, cc, addrbuf, recv\_timep);//处理回应包

628             }

629

630             /\* See? ... someone runs anotherping on this host. \*/

631             if (not\_ours)

632                 install\_filter();

633

634             /\* If nothing is in flight,"break" returns us to pinger. \*/

635             if (in\_flight() == 0)

636                 break;

637

638             /\* Otherwise, try to recvmsg()again. recvmsg()

639              \* is nonblocking after the firstiteration, so that

640              \* if nothing is queued, it will receiveEAGAIN

641              \* and return to pinger. \*/

642        }

643    }

644    finish();//收尾

645 }

//pinger()

int pinger(void)

{

       staticint oom\_count;

       staticint tokens;

       inti;

       /\*Have we already sent enough? If we have, return an arbitrary positive value. \*/

       if(exiting || (npackets && ntransmitted >= npackets &&!deadline))

              return 1000;

       /\*Check that packets < rate\*time + preload \*/

       if(cur\_time.tv\_sec == 0) {//这部分对时间的控制

              gettimeofday(&cur\_time, NULL);

              tokens = interval\*(preload-1);//preload就是通过-l选项送进来的，默认为1。

       }else {

              long ntokens;

              struct timeval tv;

              gettimeofday(&tv, NULL);

              ntokens = (tv.tv\_sec -cur\_time.tv\_sec)\*1000 +

                     (tv.tv\_usec-cur\_time.tv\_usec)/1000;

              if (!interval) {

                     /\*Case of unlimited flood is special;

                      \* if we see no reply, they are limited to100pps \*/

                     if(ntokens < MININTERVAL && in\_flight() >= preload)

                            return MININTERVAL-ntokens;

              }

              ntokens += tokens;

              if (ntokens > interval\*preload)

                     ntokens= interval\*preload;

              if (ntokens < interval)

                     returninterval - ntokens;

              cur\_time = tv;

              tokens = ntokens - interval;

       }

resend:

       i= send\_probe();//发送数据包的核心。

       if(i == 0) {//发送数据包成功。

              oom\_count = 0;

              advance\_ntransmitted();

              if (!(options & F\_QUIET) &&(options & F\_FLOOD)) {

                     /\*Very silly, but without this output with

                      \* high preload or pipe size is very confusing.\*/

                     if((preload < screen\_width && pipesize < screen\_width) ||

                         in\_flight() < screen\_width)

                            write(STDOUT\_FILENO, ".", 1);

              }

              return interval - tokens;

       }

//发送失败，错误处理。

       /\*And handle various errors... \*/

       if(i > 0) {

              /\* Apparently, it is some fatal bug. \*/

              abort();

       }else if (errno == ENOBUFS || errno == ENOMEM) {

              ……

……

}

       returnSCHINT(interval);

}

//send\_probe()

int send\_probe()

{

       structicmphdr \*icp;//icmp报文头部结构。

       intcc;

       inti;

       icp= (struct icmphdr \*)outpack;

       icp->type= ICMP\_ECHO; //request请求类型。

       icp->code= 0;

       icp->checksum= 0;

       icp->un.echo.sequence= htons(ntransmitted+1);//数据包序列。可以把htons去掉抓包对比。

       icp->un.echo.id= ident;               /\* ID \*/ //进程ID。

       CLR((ntransmitted+1)% mx\_dup\_ck);

       if(timing) {

              if (options&F\_LATENCY) {

                     staticvolatile int fake\_fucked\_egcs = sizeof(struct timeval);

                     structtimeval tmp\_tv;

                     gettimeofday(&tmp\_tv,NULL);//获得当前时间发送给对方，为计算来回时间差RTT

                     /\*egcs is crap or glibc is crap, but memcpy

                        does not copy anything, if len is constant!\*/

                     memcpy(icp+1,&tmp\_tv, fake\_fucked\_egcs);

              } else {

                     memset(icp+1,0, sizeof(struct timeval));

              }

       }

       cc= datalen + 8;                  /\* skips ICMPportion \*/

       /\*compute ICMP checksum here \*/

       icp->checksum= in\_cksum((u\_short \*)icp, cc, 0);

//数据校验，不分析了，《计算机组成原理》貌似讲过原理，可以参考TCPv1，TCPv2或者其它书籍。

       if(timing && !(options&F\_LATENCY)) {

              static volatile int fake\_fucked\_egcs =sizeof(struct timeval);

               struct timeval tmp\_tv;

              gettimeofday(&tmp\_tv, NULL);

              /\* egcs is crap or glibc is crap, butmemcpy

                does not copy anything, if len is constant! \*/

              memcpy(icp+1, &tmp\_tv,fake\_fucked\_egcs);

              icp->checksum = in\_cksum((u\_short\*)(icp+1), fake\_fucked\_egcs, ~icp->checksum);

       }

       do {

              static struct iovec iov = {outpack, 0};

              static struct msghdr m = { &whereto,sizeof(whereto),

                                            &iov, 1, &cmsg, 0, 0 };

              m.msg\_controllen = cmsg\_len;

              iov.iov\_len = cc;//可选数据为56字节再加8字节ICMP报头=64字节。所以整个icmp包就是64字节，最终产生64+20字节IPv4首部=84字节的IP数据包。

              i = sendmsg(icmp\_sock, &m, confirm);//终于发出去了。

              confirm = 0;

       }while (0);

       return(cc == i ? 0 : i);

}

//parse\_reply()，最后一部分对回应包的处理，这也是核心，需要对icmp包和ip数据包很好了解。

parse\_reply(struct msghdr \*msg, int cc,void \*addr, struct timeval \*tv)

{

       structsockaddr\_in \*from = addr;

       \_\_u8\*buf = msg->msg\_iov->iov\_base;

       structicmphdr \*icp;

       structiphdr \*ip;

       inthlen;

       intcsfailed;

       /\*Check the IP header \*/

       ip= (struct iphdr \*)buf;

       hlen= ip->ihl\*4;//这个为什么\*4可看TCPv1，它其实就是IP首部以字节为单位的长度。

       if(cc < hlen + 8 || ip->ihl < 5) {

              if (options & F\_VERBOSE)

                     fprintf(stderr,"ping: packet too short (%d bytes) from %s\n", cc,

                            pr\_addr(from->sin\_addr.s\_addr));

              return 1;

       }

       /\*Now the ICMP part \*/

       cc-= hlen;

       icp= (struct icmphdr \*)(buf + hlen);//指针往后移ip首部长度就是ICMP头部了。

       csfailed= in\_cksum((u\_short \*)icp, cc, 0);//数据校验。

       if(icp->type == ICMP\_ECHOREPLY) {//如果是回应报

              if (icp->un.echo.id != ident)//判断是否是自己发的请求包。如果不是返回。

                     return1;                /\* 'Twas not our ECHO \*/

              if (gather\_statistics((\_\_u8\*)(icp+1), cc,

                                 ntohs(icp->un.echo.sequence),

                                 ip->ttl, 0, tv, pr\_addr(from->sin\_addr.s\_addr)))

//计算来回时间差，并打印。

                     return0;//返回到main\_loop，继续循环做时间计算-->发包-->收包。

       }else {//如果不是，做其它处理。。。。

                     return0;

}

4，总结。

本次ping源码分析知识点：

源码获得方法、

Linu网络编程、

Linux系统编程信号使用、

Linux时间相关系统调用、

getopt对选项处理过程（可以看下功能更强大的getopt\_long）

ICMP报文组成理解、

本地和网络字节序理解、

抓包分析能力

附常用抓包命令：

tcpdump抓包常用命令：

tcpdump -ni any icmp

tcpdump -ni any host 192.168.147.1

tcpdump -ni any port 500or4500500or4500

tcpdump -ni any arp -s0 -w arp.cap

当然windows一下的wireshark也是非常不错的抓包分析工具。