<https://blog.csdn.net/yangzheng_yz/article/details/50593678>

运用C语言编写模拟常用网络命令ping命令实现一个基于linux原始套接字和ICMP协议的ping程序。该程序能用于检测主机或路由器工作是否正常。

程序中主要的函数

void alarm\_handler(int); /\*SIGALRM处理程序\*/

void int\_handler(int); /\*SIGINT处理程序\*/

void set\_sighandler(); /\*设置信号处理程序\*/

void send\_ping(); /\*发送ping消息\*/

void recv\_reply(); /\*接收ping应答\*/

u16 checksum(u8 \*buf, int len); /\*计算校验和\*/

int handle\_pkt(); /\*ICMP应答消息处理\*/

void get\_statistics(int, int); /\*统计ping命令的检测结果\*/

void bail(const char \*); /\*错误报告\*/

**ping.h文件：**

**[cpp]** [view plain](https://blog.csdn.net/yangzheng_yz/article/details/50593678) [copy](https://blog.csdn.net/yangzheng_yz/article/details/50593678)

1. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
2. \*      Copyright:  (C) 2016 Yang Zheng<yz2012ww@gmail.com>
3. \*                  All rights reserved.
4. \*
5. \*       Filename:  ping.h
6. \*    Description:  This head file
7. \*
8. \*        Version:  1.0.0(01/22/2016~)
9. \*         Author:  Yang Zheng <yz2012ww@gmail.com>
10. \*      ChangeLog:  1, Release initial version on "01/22/2016 04:51:41 PM"
11. \*
12. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/</span>

15. #define ICMP\_ECHOREPLY 0 /\* Echo应答\*/
16. #define ICMP\_ECHO   /\*Echo请求\*/
18. #define BUFSIZE 1500    /\*发送缓存最大值\*/
19. #define DEFAULT\_LEN 56  /\*\*ping消息数据默认大小/
21. /\*数据类型别名\*/
22. **typedef** unsigned **char** u8;
23. **typedef** unsigned **short** u16;
24. **typedef** unsigned **int** u32;
26. /\*ICMP消息头部\*/
27. **struct** icmphdr {
28. u8 type;     /\*定义消息类型\*/
29. u8 code;    /\*定义消息代码\*/
30. u16 checksum;   /\*定义校验\*/
31. **union**{
32. **struct**{
33. u16 id;
34. u16 sequence;
35. }echo;
36. u32 gateway;
37. **struct**{
38. u16 unsed;
39. u16 mtu;
40. }frag; /\*pmtu实现\*/
41. }un;
42. /\*ICMP数据占位符\*/
43. u8 data[0];
44. #define icmp\_id un.echo.id
45. #define icmp\_seq un.echo.sequence
46. };
47. #define ICMP\_HSIZE sizeof(struct icmphdr)
48. /\*定义一个IP消息头部结构体\*/
49. **struct** iphdr {
50. u8 hlen:4, ver:4;   /\*定义4位首部长度，和IP版本号为IPV4\*/
51. u8 tos;             /\*8位服务类型TOS\*/
52. u16 tot\_len;        /\*16位总长度\*/
53. u16 id;             /\*16位标志位\*/
54. u16 frag\_off;       /\*3位标志位\*/
55. u8 ttl;             /\*8位生存周期\*/
56. u8 protocol;        /\*8位协议\*/
57. u16 check;          /\*16位IP首部校验和\*/
58. u32 saddr;          /\*32位源IP地址\*/
59. u32 daddr;          /\*32位目的IP地址\*/
60. };
62. **char** \*hostname;             /\*被ping的主机名\*/
63. **int** datalen = DEFAULT\_LEN;  /\*ICMP消息携带的数据长度\*/
64. **char** sendbuf[BUFSIZE];      /\*发送字符串数组\*/
65. **char** recvbuf[BUFSIZE];      /\*接收字符串数组\*/
66. **int** nsent;                  /\*发送的ICMP消息序号\*/
67. **int** nrecv;                  /\*接收的ICMP消息序号\*/
68. pid\_t pid;                  /\*ping程序的进程PID\*/
69. **struct** timeval recvtime;    /\*收到ICMP应答的时间戳\*/
70. **int** sockfd;                 /\*发送和接收原始套接字\*/
71. **struct** sockaddr\_in dest;    /\*被ping的主机IP\*/
72. **struct** sockaddr\_in from;    /\*发送ping应答消息的主机IP\*/
73. **struct** sigaction act\_alarm;
74. **struct** sigaction act\_int;
76. /\*函数原型\*/
77. **void** alarm\_handler(**int**);        /\*SIGALRM处理程序\*/
78. **void** int\_handler(**int**);          /\*SIGINT处理程序\*/
79. **void** set\_sighandler();          /\*设置信号处理程序\*/
80. **void** send\_ping();               /\*发送ping消息\*/
81. **void** recv\_reply();              /\*接收ping应答\*/
82. u16 checksum(u8 \*buf, **int** len); /\*计算校验和\*/
83. **int** handle\_pkt();               /\*ICMP应答消息处理\*/
84. **void** get\_statistics(**int**, **int**);  /\*统计ping命令的检测结果\*/
85. **void** bail(**const** **char** \*);        /\*错误报告\*/

**ping.c文件：**

**[cpp]** [view plain](https://blog.csdn.net/yangzheng_yz/article/details/50593678) [copy](https://blog.csdn.net/yangzheng_yz/article/details/50593678)

1. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
2. \*      Copyright:  (C) 2016 Yang Zheng<yz2012ww@gmail.com>
3. \*                  All rights reserved.
4. \*
5. \*       Filename:  ping.c
6. \*    Description:  This file
7. \*
8. \*        Version:  1.0.0(01/22/2016~)
9. \*         Author:  Yang Zheng <yz2012ww@gmail.com>
10. \*      ChangeLog:  1, Release initial version on "01/22/2016 04:51:12 PM"
11. \*
12. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/</span>
13. #include<stdio.h>
14. #include<stdlib.h>
15. #include<sys/time.h>  /\*是Linux系统的日期时间头文件\*/
16. #include<unistd.h>    /\*　是POSIX标准定义的unix类系统定义符号常量的头文件，包含了许多UNIX系统服务的函数原型，例如read函数、write函数和getpid函数\*/
17. #include<string.h>
18. #include<sys/socket.h>    /\*对与引用socket函数必须\*/
19. #include<sys/types.h>
20. #include<netdb.h> /\*定义了与网络有关的结构，变量类型，宏，函数。函数gethostbyname()用\*/
21. #include<errno.h> /\*sys/types.h中文名称为基本系统数据类型\*/
22. #include<arpa/inet.h> /\*inet\_ntoa()和inet\_addr()这两个函数，包含在 arpa/inet.h\*/
23. #include<signal.h>    /\*进程对信号进行处理\*/
24. #include<netinet/in.h>    /\*互联网地址族\*/
26. #include"ping.h"
27. #define IP\_HSIZE sizeof(struct iphdr)   /\*定义IP\_HSIZE为ip头部长度\*/
28. #define IPVERSION  4   /\*定义IPVERSION为4，指出用ipv4\*/
30. /\*设置的时间是一个结构体，倒计时设置，重复倒时，超时值设为1秒\*/
31. **struct** itimerval val\_alarm = {
32. .it\_interval.tv\_sec = 1,
33. .it\_interval.tv\_usec = 0,
34. .it\_value.tv\_sec = 0,
35. .it\_value.tv\_usec = 1
36. };
38. /\*argc表示隐形程序命令行中参数的数目，argv是一个指向字符串数组指针，其中每一个字符对应一个参数\*/
39. **int** main(**int** argc,**char** \*\*argv)
40. {
41. **struct** hostent      \*host; /\*该结构体属于include<netdb.h>\*/
42. **int**                 on = 1;
44. **if**( argc < 2)/\*判断是否输入了地址\*/
45. {
46. printf("Usage: %s hostname\n",argv[0]);
47. exit(1);
48. }
50. /\*gethostbyname()返回对应于给定主机名的包含主机名字和地址信息的结构指针,\*/
51. //if((host = getaddrinfo(argv[1])) == NULL)
52. **if**((host = gethostbyname(argv[1])) == NULL)
53. {
54. printf("usage:%s hostname/IP address\n", argv[0]);
55. exit(1);
56. }
58. hostname = argv[1]; /\*取出地址名\*/
60. memset(&dest,0,**sizeof** dest);    /\*将dest中前sizeof(dest)个字节替换为0并返回s,此处为初始化,给最大内存清零\*/
61. dest.sin\_family=PF\_INET;        /\*PF\_INET为IPV4，internet协议，在<netinet/in.h>中，地址族\*/
62. dest.sin\_port=ntohs(0);         /\*端口号,ntohs()返回一个以主机字节顺序表达的数。\*/
63. dest.sin\_addr=\*(**struct** in\_addr \*)host->h\_addr\_list[0];/\*host->h\_addr\_list[0]是地址的指针.返回IP地址，初始化\*/
65. /\*PF\_INEI套接字协议族，SOCK\_RAW套接字类型，IPPROTO\_ICMP使用协议，
66. 调用socket函数来创建一个能够进行网络通信的套接字。这里判断是否创建成功\*/
67. **if**((sockfd = socket(PF\_INET, SOCK\_RAW, IPPROTO\_ICMP)) < 0)
68. {
69. perror("RAW socket created error");
70. exit(1);
71. }
73. /\*设置当前套接字选项特定属性值，sockfd套接字，IPPROTO\_IP协议层为IP层，
74. IP\_HDRINCL套接字选项条目，套接字接收缓冲区指针，sizeof(on)缓冲区长度的长度\*/
75. setsockopt(sockfd, IPPROTO\_IP, IP\_HDRINCL, &on, **sizeof**(on));
77. /\*getuid()函数返回一个调用程序的真实用户ID,setuid()是让普通用户
78. 可以以root用户的角色运行只有root帐号才能运行的程序或命令。\*/
79. setuid(getuid());
80. pid = getpid(); /\*getpid函数用来取得目前进程的进程识别码\*/
82. set\_sighandler();/\*对信号处理\*/
83. printf("Ping %s(%s): %d bytes data in ICMP packets.\n", argv[1], inet\_ntoa(dest.sin\_addr), datalen);
85. **if**((setitimer(ITIMER\_REAL, &val\_alarm, NULL)) == -1) /\*定时函数\*/
86. {
87. bail("setitimer fails.");
88. }
90. recv\_reply(); /\*接收ping应答\*/
92. **return** 0;
93. }
95. /\*发送ping消息\*/
96. **void** send\_ping(**void**)
97. {
98. **struct** iphdr        \*ip\_hdr;   /\*iphdr为IP头部结构体\*/
99. **struct** icmphdr      \*icmp\_hdr;   /\*icmphdr为ICMP头部结构体\*/
100. **int**                 len;
101. **int**                 len1;
103. /\*ip头部结构体变量初始化\*/
104. ip\_hdr=(**struct** iphdr \*)sendbuf; /\*字符串指针\*/
105. ip\_hdr->hlen=**sizeof**(**struct** iphdr)>>2;  /\*头部长度\*/
106. ip\_hdr->ver=IPVERSION;   /\*版本\*/
107. ip\_hdr->tos=0;   /\*服务类型\*/
108. ip\_hdr->tot\_len=IP\_HSIZE+ICMP\_HSIZE+datalen; /\*报文头部加数据的总长度\*/
109. ip\_hdr->id=0;    /\*初始化报文标识\*/
110. ip\_hdr->frag\_off=0;  /\*设置flag标记为0\*/
111. ip\_hdr->protocol=IPPROTO\_ICMP;/\*运用的协议为ICMP协议\*/
112. ip\_hdr->ttl=255; /\*一个封包在网络上可以存活的时间\*/
113. ip\_hdr->daddr=dest.sin\_addr.s\_addr;  /\*目的地址\*/
114. len1=ip\_hdr->hlen<<2;  /\*ip数据长度\*/
115. /\*ICMP头部结构体变量初始化\*/
116. icmp\_hdr=(**struct** icmphdr \*)(sendbuf+len1);  /\*字符串指针\*/
117. icmp\_hdr->type=8;    /\*初始化ICMP消息类型type\*/
118. icmp\_hdr->code=0;    /\*初始化消息代码code\*/
119. icmp\_hdr->icmp\_id=pid;   /\*把进程标识码初始给icmp\_id\*/
120. icmp\_hdr->icmp\_seq=nsent++;  /\*发送的ICMP消息序号赋值给icmp序号\*/
121. memset(icmp\_hdr->data,0xff,datalen);  /\*将datalen中前datalen个字节替换为0xff并返回icmp\_hdr-dat\*/
123. gettimeofday((**struct** timeval \*)icmp\_hdr->data,NULL); /\* 获取当前时间\*/
125. len=ip\_hdr->tot\_len; /\*报文总长度赋值给len变量\*/
126. icmp\_hdr->checksum=0;    /\*初始化\*/
127. icmp\_hdr->checksum=checksum((u8 \*)icmp\_hdr,len);  /\*计算校验和\*/
129. sendto(sockfd,sendbuf,len,0,(**struct** sockaddr \*)&dest,**sizeof** (dest)); /\*经socket传送数据\*/
130. }
132. /\*接收程序发出的ping命令的应答\*/
133. **void** recv\_reply()
134. {
135. **int**         n;
136. **int**         len;
137. **int**         errno;
139. n = 0;
140. nrecv = 0;
141. len = **sizeof**(from);   /\*发送ping应答消息的主机IP\*/
143. **while**(nrecv < 4)
144. {
145. /\*经socket接收数据,如果正确接收返回接收到的字节数，失败返回0.\*/
146. **if**((n=recvfrom(sockfd, recvbuf, **sizeof**(recvbuf), 0, (**struct** sockaddr \*)&from, &len))<0)
147. {
148. **if**(errno==EINTR)  /\*EINTR表示信号中断\*/
149. **continue**;
150. bail("recvfrom error");
151. }
153. gettimeofday(&recvtime, NULL);   /\*记录收到应答的时间\*/
155. **if**(handle\_pkt())    /\*接收到错误的ICMP应答信息\*/
156. **continue**;
158. nrecv++;
159. }
161. get\_statistics(nsent, nrecv);     /\*统计ping命令的检测结果\*/
162. }
164. /\*计算校验和\*/
165. u16 checksum(u8 \*buf,**int** len)
166. {
167. u32 sum = 0;
168. u16 \*cbuf;
170. cbuf = (u16 \*)buf;
172. **while**(len > 1)
173. {
174. sum += \*cbuf++;
175. len -= 2;
176. }
178. **if**(len)
179. {
180. sum += \*(u8 \*)cbuf;
181. }
183. sum = (sum >> 16) + (sum & 0xffff);
184. sum += (sum >> 16);
186. **return** ~sum;
187. }
189. /\*ICMP应答消息处理\*/
190. **int** handle\_pkt()
191. {
192. **struct** iphdr        \*ip;
193. **struct** icmphdr      \*icmp;
194. **int**                 ip\_hlen;
195. u16                 ip\_datalen; /\*ip数据长度\*/
196. **double**              rtt; /\* 往返时间\*/
197. **struct** timeval      \*sendtime;
199. ip = (**struct** iphdr \*)recvbuf;
201. ip\_hlen = ip->hlen << 2;
202. ip\_datalen = ntohs(ip->tot\_len) - ip\_hlen;
204. icmp = (**struct** icmphdr \*)(recvbuf + ip\_hlen);
206. **if**(checksum((u8 \*)icmp, ip\_datalen)) /\*计算校验和\*/
207. **return** -1;
209. **if**(icmp->icmp\_id != pid)
210. **return** -1;
212. sendtime = (**struct** timeval \*)icmp->data; /\*发送时间\*/
213. rtt = ((&recvtime)->tv\_sec - sendtime->tv\_sec) \* 1000 + ((&recvtime)->tv\_usec - sendtime->tv\_usec)/1000.0; /\* 往返时间\*/
214. /\*打印结果\*/
215. printf("%d bytes from %s:icmp\_seq=%u ttl=%d rtt=%.3f ms\n",  \
216. ip\_datalen,                 /\*IP数据长度\*/
217. inet\_ntoa(from.sin\_addr),   /\*目的ip地址\*/
218. icmp->icmp\_seq,              /\*icmp报文序列号\*/
219. ip->ttl,                 /\*生存时间\*/
220. rtt);                       /\*往返时间\*/
222. **return** 0;
223. }
225. /\*设置信号处理程序\*/
226. **void** set\_sighandler()
227. {
228. act\_alarm.sa\_handler = alarm\_handler;
229. /\*sigaction()会依参数signum指定的信号编号来设置该信号的处理函数。参数signum指所要捕获信号或忽略的信号，
230. &act代表新设置的信号共用体，NULL代表之前设置的信号处理结构体。这里判断对信号的处理是否成功。\*/
231. **if**(sigaction(SIGALRM, &act\_alarm, NULL) == -1)
232. {
233. bail("SIGALRM handler setting fails.");
234. }
236. act\_int.sa\_handler = int\_handler;
237. **if**(sigaction(SIGINT, &act\_int, NULL) == -1)
238. {
239. bail("SIGALRM handler setting fails.");
240. }
241. }
243. /\*统计ping命令的检测结果\*/
244. **void** get\_statistics(**int** nsent,**int** nrecv)
245. {
246. printf("--- %s ping statistics ---\n",inet\_ntoa(dest.sin\_addr)); /\*将网络地址转换成“.”点隔的字符串格式。\*/
247. printf("%d packets transmitted, %d received, %0.0f%% ""packet loss\n",  \
248. nsent,nrecv,1.0\*(nsent-nrecv)/nsent\*100);
249. }
251. /\*错误报告\*/
252. **void** bail(**const** **char** \* on\_what)
253. {
254. /\*:向指定的文件写入一个字符串（不写入字符串结束标记符‘\0’）。成功写入一个字符串后，
255. 文件的位置指针会自动后移，函数返回值为0；否则返回EOR(符号常量，其值为-1)。\*/
256. fputs(strerror(errno),stderr);
257. fputs(":",stderr);
258. fputs(on\_what,stderr);
259. fputc('\n',stderr); /\*送一个字符到一个流中\*/
261. exit(1);
262. }
264. /\*SIGINT（中断信号）处理程序\*/
265. **void** int\_handler(**int** sig)
266. {
267. get\_statistics(nsent,nrecv);    /\*统计ping命令的检测结果\*/
268. close(sockfd);  /\*关闭网络套接字\*/
269. exit(1);
270. }
272. /\*SIGALRM（终止进程）处理程序\*/
273. **void** alarm\_handler(**int** signo)
274. {
275. send\_ping();    /\*发送ping消息\*/
277. }<strong>
278. </strong>

**运行结果：**

# ./ping 192.168.1.1  
Ping 192.168.1.1(192.168.1.1): 56 bytes data in ICMP packets.  
64 bytes from 192.168.1.1:icmp\_seq=0 ttl=64 rtt=113.510 ms  
64 bytes from 192.168.1.1:icmp\_seq=1 ttl=64 rtt=19.854 ms  
--- 192.168.1.1 ping statistics ---  
2 packets transmitted, 2 received, 0% packet loss  
   
# ./ping baidu.com  
Ping baidu.com(220.181.57.217): 56 bytes data in ICMP packets.  
64 bytes from 220.181.57.217:icmp\_seq=0 ttl=50 rtt=44.679 ms  
64 bytes from 220.181.57.217:icmp\_seq=1 ttl=50 rtt=58.446 ms  
64 bytes from 220.181.57.217:icmp\_seq=2 ttl=50 rtt=47.933 ms  
64 bytes from 220.181.57.217:icmp\_seq=3 ttl=50 rtt=44.946 ms  
--- 220.181.57.217 ping statistics ---  
4 packets transmitted, 4 received, 0% packet loss