#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <netinet/in.h>

#include <netdb.h>

#include <unistd.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <sys/types.h>

#include <sys/ioctl.h>

#include <net/if.h>

#include <time.h>

#include "fill\_packet.h"

#include "pcap.h"

pid\_t pid;

void if\_ip(char \*, char\*);

static const char\* dev = "eth0";

int main(int argc, char\* argv[])

{

int sockfd;

int on = 1,i,sequence\_num=1,pcap\_check=0;

char c,\*target\_ip,\*copy\_target\_ip,\*gateway,data[56];

struct timeval tm;

struct sockaddr\_in dst;

struct ifreq req;

pid = getpid();

myicmp \*packet = (myicmp\*)malloc(PACKET\_SIZE);//PACKET\_SIZE 92

int count = DEFAULT\_SEND\_COUNT; //DEFAULT\_SEND\_COUNT 3

int timeout = DEFAULT\_TIMEOUT; //DEFAULT\_TIMEOUT 1500

/\*

\* in pcap.c, initialize the pcap

\*/

while((c = getopt(argc,argv,"g:w:c:"))!=-1){

switch(c){

case 'g':

gateway = malloc(strlen(optarg)+1);

strcpy(gateway,optarg);

break;

case 'w':

timeout = atoi(optarg);

break;

case 'c':

count = atoi(optarg);

break;

default :

printf("Usage ./myping -g [gateway] -w [timeout] -c [count]\n");

exit(0);

}

}

/\*

extern char \*optarg; //選項的參數指針

extern int optind, //下一次調用getopt的時，從optind存儲的位置處重新開始檢查選項。

extern int opterr, //當opterr=0時，getopt不向stderr輸出錯誤信息。

extern int optopt; //當命令行選項字符不包括在optstring中或者選項缺少必要的參數時，該選項存儲在optopt中，getopt返回'？ '

\*/

target\_ip = (char \*)malloc(strlen(argv[optind])+1);

copy\_target\_ip = (char \*)malloc(strlen(argv[optind])+1);

strcpy(target\_ip,argv[optind]);

strcpy(copy\_target\_ip,target\_ip);

pcap\_init( copy\_target\_ip , timeout);

if((sockfd = socket(AF\_INET, SOCK\_RAW , IPPROTO\_RAW)) < 0){

perror("socket");

exit(1);

}

/\*

Raw socket是linux network programming一個很進階的技巧，大致上說起來就是，跳過socket所在的transport layer，直接往下撈"封包"，也不能說封包(packet)，因為raw socket撈到的是frame

\*/

/\*

如果protocol是IPPROTO\_RAW(255)，這時候，這個socket只能用來發送IP包，而不能接收任何的數據。發送的數據需要自己填充IP包頭，並且自己計算校驗和。

\*/

if(setsockopt( sockfd, IPPROTO\_IP, IP\_HDRINCL, &on, sizeof(on)) < 0){

perror("setsockopt");

exit(1);

}

/\*

對於protocol為0（IPPROTO\_IP)的raw socket。用於接收任何的IP數據包

\*/

/\*

當需要編寫自己的IP數據包首部時，可以在原始套接字上設置套接字選項IP\_HDRINCL.在不設置這個選項的情況下，IP協議自動填充IP數據包的首部。

\*/

//get self\_ip and fill in packt src

strncpy(req.ifr\_name,dev,IFNAMSIZ);

if(ioctl(sockfd,SIOCGIFADDR,&req)<0){

perror("ioctl");

exit(1);

}

packet->ip\_hdr.ip\_src = ((struct sockaddr\_in \*)&req.ifr\_addr)->sin\_addr;

//fill packet

fill\_iphdr(&(packet->ip\_hdr),gateway);

fill\_icmphdr(&(packet->icmp\_hdr));

//dst\_addr set

dst.sin\_family = AF\_INET;

inet\_aton(gateway,&dst.sin\_addr);

//icmp payload

memset(packet->data,'\0',56);

strcpy(packet->data,"Data");

//ip\_option

packet->ip\_option[0] = 0x83;

packet->ip\_option[1] = 0x07;

packet->ip\_option[2] = 0x04;

if\_ip(packet->ip\_option+3,target\_ip);

packet->ip\_option[7] = 0;

/\*

\* Use "sendto" to send packets, and use "pcap\_get\_reply"(in pcap.c)

\* to get the "ICMP echo response" packets.

\* You should reset the timer every time before you send a packet.

\*/

printf("\nPing %s (data\_size = 56 , id = 0x%x , timeout = %d ,count = %d) :\n",copy\_target\_ip,pid,timeout,count);

for(i=1;i<=count;i++){

//fill sequence\_num and get checksum

packet->icmp\_hdr.un.echo.sequence = sequence\_num;

fill\_cksum(&(packet->icmp\_hdr));

//send packet

if(sendto(sockfd, packet, PACKET\_SIZE, 0, (struct sockaddr \*)&dst, sizeof(dst)) < 0){

perror("sendto");

exit(1);

}

gettimeofday(&tm,NULL);

if((pcap\_check = pcap\_get\_reply(sequence\_num)) != 0 ){

printf("Reply from %s = %.3f ms\n",copy\_target\_ip,(double)(pcap\_check - tm.tv\_usec)/1000);

printf(" Router : %s\n",gateway);

}else{

printf("Reply from %s = \*\n",copy\_target\_ip);

}

sequence\_num++;

fflush(stdout);

}

free(packet);

return 0;

}

void if\_ip(char \*incoming\_ip , char\* ip){

char delim[] = ". \n\t";

char \*token=0;

int i;

for(i=0;i<=4;i++){

if(i==0)

token = strtok(ip,delim);

else if(i<=3){

token = strtok(NULL,delim);

if(token == NULL){

printf("Wrong filter ip address\n");

exit(1);

}

}else{

token = strtok(NULL,delim);

if(token != NULL){

printf("Wrong filter ip address\n");

exit(1);

}

}

if(i<=3){

if(atoi(token)>=0 && atoi(token) <256 ){

incoming\_ip[i]=strtol(token,NULL,0);

}else{

printf("Wrong filter ip address\n");

exit(0);

}

}

}

}