**一、套接字概述**

原始套接字编程和之前的 UDP 编程差不多，无非就是创建一个套接字后，通过这个套接字接收数据或者发送数据。区别在于，原始套接字可以自行组装数据包（伪装本地 IP，本地 MAC），可以接收本机网卡上所有的数据帧（数据包）。另外，必须在管理员权限下才能使用原始套接字。

**二、原始套接字的创建：**

int socket ( int family, int type, int protocol );

参数：

family：协议族 这里写 PF\_PACKET

type：  套接字类，这里写 SOCK\_RAW

protocol：协议类别，指定可以接收或发送的数据包类型，不能写 “0”，取值如下，注意，传参时需要用 htons() 进行字节序转换。

ETH\_P\_IP：IPV4数据包

ETH\_P\_ARP：ARP数据包

ETH\_P\_ALL：任何协议类型的数据包

返回值：

成功( >0 )：套接字，这里为链路层的套接字

失败( <0 )：出错

实例如下：

#include <sys/socket.h>

#include <netinet/ether.h>

#include <stdio.h>  // perror

int main(int argc,charchar \*argv[])

{

    int sock\_raw\_fd = socket(PF\_PACKET, SOCK\_RAW, htons(ETH\_P\_ALL) );

    if(sock\_raw\_fd < 0){

        perror("socket");

        return -1;

    }

    return 0;

}

**三、获取链路层的数据包：**

ssize\_t recvfrom(  int sockfd,

void \*buf,

size\_t nbytes,

int flags,

struct sockaddr \*from,

socklen\_t \*addrlen );

参数：

sockfd:原始套接字

buf：接收数据缓冲区

nbytes:接收数据缓冲区的大小

flags：套接字标志(常为0)

from：这里没有用，写 NULL

addrlen：这里没有用，写 NULL

返回值：

成功：接收到的字符数

失败：-1

实例如下：

#include <stdio.h>

#include <netinet/in.h>

#include <sys/socket.h>

#include <netinet/ether.h>

int main(int argc,charchar \*argv[])

{

    unsigned char buf[1024] = {0};

    int sock\_raw\_fd = socket(PF\_PACKET, SOCK\_RAW, htons(ETH\_P\_ALL));

    //获取链路层的数据包

    int len = recvfrom(sock\_raw\_fd, buf, sizeof(buf), 0, NULL, NULL);

    printf("len = %d\n", len);

    return 0;

}

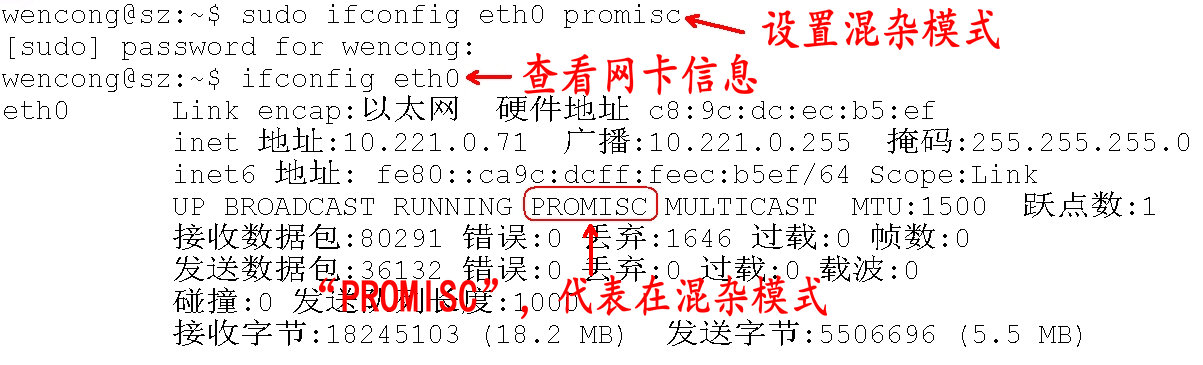
**四、混杂模式**

默认的情况下，我们接收数据，目的地址是本地地址，才会接收。有时候我们想接收所有经过网卡的所有数据流，而不论其目的地址是否是它，这时候我们需要设置网卡为混杂模式。

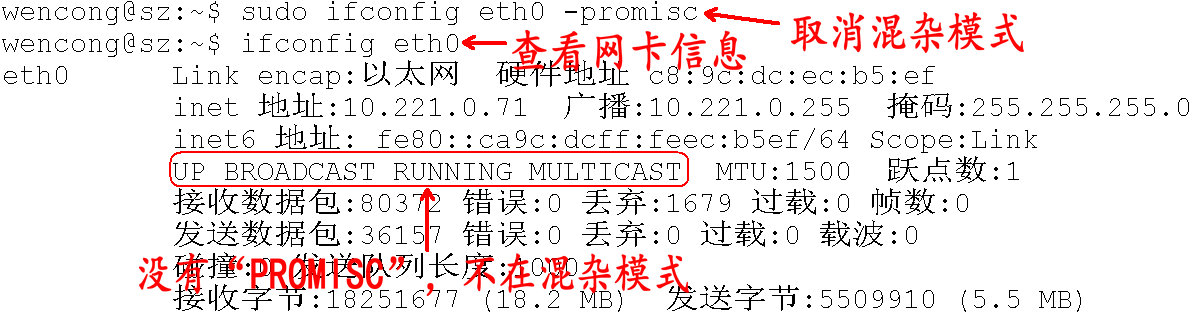
网卡的混杂模式一般在网络管理员分析网络数据作为网络故障诊断手段时用到，同时这个模式也被网络黑客利用来作为网络数据窃听的入口。在 Linux 操作系统中设置网卡混杂模式时需要管理员权限。在 Windows 操作系统和 Linux 操作系统中都有使用混杂模式的抓包工具，比如著名的开源软件 Wireshark。

通过命令给 Linux 网卡设置混杂模式（需要管理员权限）

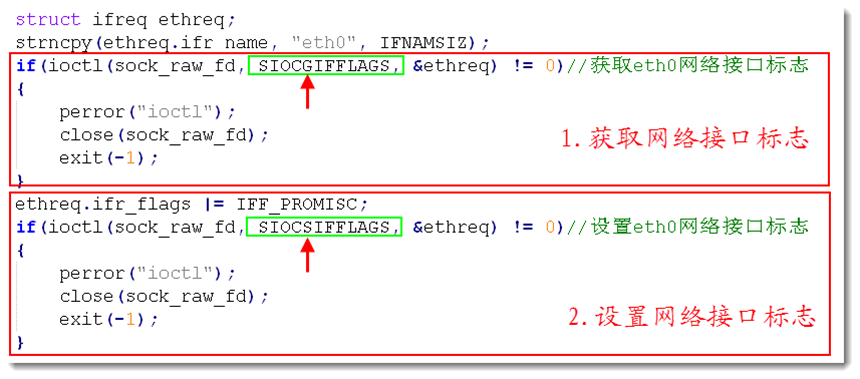
设置混杂模式：ifconfig eth0 promisc



取消混杂模式：ifconfig eth0 -promisc



通过代码给 Linux 网卡设置混杂模式



代码如下：

struct ifreq ethreq;    //网络接口地址

strncpy(ethreq.ifr\_name, "eth0", IFNAMSIZ);         //指定网卡名称

if(-1 == ioctl(sock\_raw\_fd, SIOCGIFINDEX, &ethreq)) //获取网络接口

{

    perror("ioctl");

    close(sock\_raw\_fd);

    exit(-1);

}

ethreq.ifr\_flags |= IFF\_PROMISC;

if(-1 == ioctl(sock\_raw\_fd, SIOCSIFINDEX, &ethreq)) //网卡设置混杂模式

{

    perror("ioctl");

    close(sock\_raw\_fd);

    exit(-1);

}

**五、发送自定义的数据包：**

ssize\_t sendto(     int sockfd,

const void \*buf,

size\_t nbytes,int flags,

const struct sockaddr \*to,

socklen\_t addrlen );

参数：

sockfd：原始套接字

buf：发送数据缓冲区

nbytes: 发送数据缓冲区的大小

flags：一般为 0

to：本机网络接口，指发送的数据应该从本机的哪个网卡出去，而不是以前的目的地址

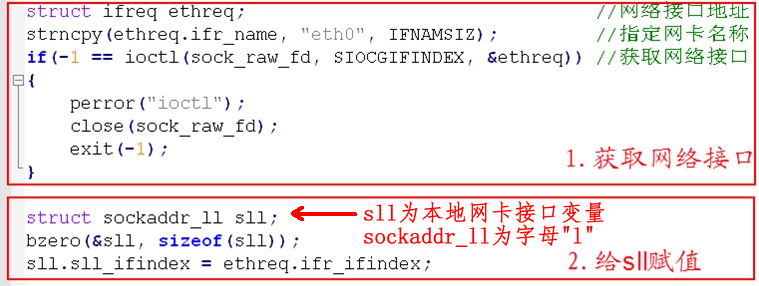
addrlen：to 所指向内容的长度

返回值：

成功：发送数据的字符数

失败： -1

本机网络接口的定义



发送完整代码如下：

struct sockaddr\_ll sll;                 //原始套接字地址结构

struct ifreq ethreq;                    //网络接口地址

strncpy(ethreq.ifr\_name, "eth0", IFNAMSIZ);         //指定网卡名称

if(-1 == ioctl(sock\_raw\_fd, SIOCGIFINDEX, ðreq))    //获取网络接口

{

    perror("ioctl");

    close(sock\_raw\_fd);

    exit(-1);

}

/\*将网络接口赋值给原始套接字地址结构\*/

bzero(&sll, sizeof(sll));

sll.sll\_ifindex = ethreq.ifr\_ifindex;

// 发送数据

// send\_msg, msg\_len 这里还没有定义，模拟一下

int len = sendto(sock\_raw\_fd, send\_msg, msg\_len, 0 , (struct sockaddr \*)&sll, sizeof(sll));

if(len == -1)

{

    perror("sendto");

}

这里头文件情况如下：

#include <net/if.h>// struct ifreq

#include <sys/ioctl.h> // ioctl、SIOCGIFADDR

#include <sys/socket.h> // socket

#include <netinet/ether.h> // ETH\_P\_ALL

#include <netpacket/packet.h> // struct sockaddr\_ll