Linux网络编程：libpcap 移植及使用

原创 錦鈥銀 于 2021-06-16 17:57:48 发布 1133 收藏 12 版权

分类专栏： 网络编程 文章标签： 网络通信 socket

网络编程 专栏收录该内容

目录

参考文章：

1. Linux下移植libpcap抓包库到arm平台
2. Linux 网络编程—— libpcap 详解
3. libpcap使用
4. Libpcap库编程指南–数据包捕获
5. libpcap函数库详细介绍

一、libpcap 库下载

1. http://www.tcpdump.org/
2. https://github.com/the-tcpdump-group/libpcap/releases

二、libpcap库交叉编译 安装

1. 配置交叉编译环境
2. 下载最新版本 libpcap-1.10.1.tar.gz，解压缩到当前目录：

```
tar -xzvf ./libpcap-1.10.1.tar.gz -C ./
```

3. 配置安装目录和交叉编译环境

```
./configure --prefix=/xxx/xxx/install/ --host=arm-linux-gnueabihf
```

4. 编译

```
make
```

5. 安装

```
make install
```

在 install/lib/ 目录下生成如下文件：

6. cd 进入 install 安装目录，打包lib目录下动态库文件

```
tar -zcvf libpcap-1.10.1-install.tar.gz lib/*.so*
```

7. 将 libpcap-1.10.1-install.tar.gz 压缩包拷贝到开发板上，解压
   新建文件夹：/usr/local/lib/libpcap ， 然后解压到该文件夹中

```
sudo mkdir /usr/local/lib/libpcap
```

```
sudo tar -zxf ./libpcap-1.10.1-install.tar.gz --strip-components 1 -C /usr/local/lib/libpcap
```

8. 开发板上添加库文件搜索路径
   打开ld.so.conf文件

```
sudo vi /etc/ld.so.conf.d/libc.conf
```

在 /etc/ld.so.conf 文件中添加库的搜索路径

```
/usr/local/lib/libpcap //根据自己的库路径添加
```

然后 `ldconfig` 生成/etc/ld.so.cache， `ldconfig -v` 查看

```
sudo ldconfig
```

三、应用程序交叉编译

交叉编译应用程序：**需要加 -lpcap 选项，并指定头文件及动态库路径**

```
arm-linux-gnueabihf-gcc ./libpcap_test.c -o ./libpcap_test -lpcap -I/xxx/include/ -L/xxx/lib/
```

1. 查看头文件及动态库路径
   Libpcap 安装为一个库和一组包含文件。在您的程序中使用的主要包含文件是：

```
#include <pcap.h>
```

要获得头文件和库文件的正确搜索路径，请使用标准pkg-config工具：

```
pkg-config --libs --static --cflags libpcap
```

结果：

```
-I/usr/local/include -L/usr/local/lib -lpcap
```

/usr/local/此处显示的路径为默认值。configure时，可以使用 --prefix 选项指定不同的路径。

2. 编译需要添加 -lpcap 选项

```
gcc test.c -o test -lpcap
```

3. 基于 GNU autotools 的项目，请在以下内容中使用 `configure.ac` ：

```
# Check for required libraries
PKG_CHECK_MODULES([libpcap], [libpcap>= 1.2])
```

并在您的 Makefile.am：

```
proggy_CFLAGS = $(libpcap_CFLAGS)
proggy_LDADD  = $(libpcap_LIBS)
```

四、Ubuntu 系统安装 libpcap（非交叉编译）

1. libpcap 的安装

```
sudo apt-get install libpcap-dev
```

2. 应用程序编译

```
gcc libpcap_test.c -o libpcap_test -lpcap
```

五、libpcap使用

参考：

1. libpcap使用
2. Linux 网络编程—— libpcap 详解
3. Libpcap库编程指南–数据包捕获
4. libpcap函数库详细介绍

六、开发板上测试

使用 libpcap 循环捕获网络数据包（libpcap_test.c）：

```
#include <stdio.h>
#include <pcap.h>
#include <arpa/inet.h>
#include <time.h>
#include <stdlib.h>

#define BUFSIZE 1514

struct ether_header
{
        unsigned char ether_dhost[6];   //目的mac
        unsigned char ether_shost[6];   //源mac
        unsigned short ether_type;              //以太网类型
};

/*****************************回调函数***********************************/
void ethernet_protocol_callback(unsigned char *argument,const struct pcap_pkthdr *packet_heaher,const unsigned char *packet_cont
{
        unsigned char *mac_string;                              //
        struct ether_header *ethernet_protocol;
        unsigned short ethernet_type;                   //以太网类型
        printf("-----------------------------------------------\n");
        printf("%s\n", ctime((time_t *)&(packet_heaher->ts.tv_sec))); //转换时间
        ethernet_protocol = (struct ether_header *)packet_content;

        mac_string = (unsigned char *)ethernet_protocol->ether_shost;//获取源mac地址
        printf("Mac Source Address is %02x:%02x:%02x:%02x:%02x:%02x\n",*(mac_string+0),*(mac_string+1),*(mac_string+2),*(mac_str
        mac_string = (unsigned char *)ethernet_protocol->ether_dhost;//获取目的mac
        printf("Mac Destination Address is %02x:%02x:%02x:%02x:%02x:%02x\n",*(mac_string+0),*(mac_string+1),*(mac_string+2),*(ma

        ethernet_type = ntohs(ethernet_protocol->ether_type);//获得以太网的类型
        printf("Ethernet type is :%04x\n",ethernet_type);
        switch(ethernet_type)
        {
                case 0x0800:printf("The network layer is IP protocol\n");break;//ip
                case 0x0806:printf("The network layer is ARP protocol\n");break;//arp
                case 0x0835:printf("The network layer is RARP protocol\n");break;//rarp
                default:break;

```

```
        }
        usleep(800*1000);
}

int main(int argc, char *argv[])
{
        char error_content[100];        //出错信息
        pcap_t * pcap_handle;
        unsigned char *mac_string;
        unsigned short ethernet_type;                   //以太网类型
        char *net_interface = NULL;                                     //接口名字
        struct pcap_pkthdr protocol_header;
        struct ether_header *ethernet_protocol;

        //获取网络接口
        net_interface = pcap_lookupdev(error_content);
        if(NULL == net_interface)
        {
                perror("pcap_lookupdev");
                exit(-1);
        }

        pcap_handle = pcap_open_live(net_interface,BUFSIZE,1,0,error_content);//打开网络接口

        if(pcap_loop(pcap_handle,-1,ethernet_protocol_callback,NULL) < 0)
        {
                perror("pcap_loop");
        }

        pcap_close(pcap_handle);
        return 0;
}
```

交叉编译：

```
arm-linux-gnueabihf-gcc ./libpcap_test.c -o ./libpcap_test -lpcap -I/home/osrc/Projects/tools/libpcap/install/include/ -L/home/
```

在开发板上运行，**需要root账户权限**：

```
sudo ./libpcap_test
```

结果如下：

```
root@stretch-armhf:/home/qudoor# ./libpcap_test
device eth0 entered promiscuous mode
---------------------------------------------------
Wed Jun 16 09:40:49 2021

Mac Source Address is dc:fe:18:c7:97:9f
Mac Destination Address is ff:ff:ff:ff:ff:ff
Ethernet type is :0806
The network layer is ARP protocol
---------------------------------------------------
Wed Jun 16 09:40:49 2021

Mac Source Address is e4:54:e8:ac:27:65
Mac Destination Address is 33:33:00:00:00:0c
Ethernet type is :86dd
---------------------------------------------------
Wed Jun 16 09:40:49 2021

Mac Source Address is 9c:e3:74:91:c8:fb
Mac Destination Address is ff:ff:ff:ff:ff:ff
Ethernet type is :0806
The network layer is ARP protocol
---------------------------------------------------
Wed Jun 16 09:40:49 2021

Mac Source Address is dc:fe:18:c7:97:9f
Mac Destination Address is ff:ff:ff:ff:ff:ff
Ethernet type is :0806
The network layer is ARP protocol
---------------------------------------------------
```

使用 libpcap 过滤目的端口为8080的数据包，循环捕获（libpcap_filter_test.c）；

```c
#include <pcap.h>
#include <time.h>
#include <stdlib.h>
#include <stdio.h>

/*
* src host 192.168.1.177 //只接收源 ip 地址是 192.168.1.177 的数据包
* dst port 80 //只接收 tcp/udp 的目的端口是 80 的数据包
* not tcp //只接收不使用 tcp 协议的数据包
* tcp[13] == 0x02 and (dst port 22 or dst port 23) //只接收 SYN 标志位置位且目标端口是 22 或 23 的数据包 ( tcp 首部开始的第 13 个字节
* icmp[icmptype] == icmp-echoreply or icmp[icmptype] == icmp-echo //只接收 icmp 的 ping 请求和 ping 响应的数据包
* ehter dst 00:e0:09:c1:0e:82 //只接收以太网 mac 地址是 00:e0:09:c1:0e:82 的数据包
*/
#define PACP_FILTER     "dst port 8080"


struct ether_header
{
        unsigned char ether_dhost[6];   //目的mac
        unsigned char ether_shost[6];   //源mac
        unsigned short ether_type;              //以太网类型
};

void getPacket(u_char * arg, const struct pcap_pkthdr * pkthdr, const u_char * packet)
{
        int * id = (int *)arg;
        unsigned char *mac_string;                              //
        struct ether_header *ethernet_protocol;
        unsigned short ethernet_type;                   //以太网类型
        printf("---------------------------------------------------\n");

        printf("id: %d\n", ++(*id));
        printf("Packet length: %d\n", pkthdr->len);
        printf("Number of bytes: %d\n", pkthdr->caplen);
        printf("Recieved time: %s", ctime((const time_t *)&pkthdr->ts.tv_sec));

        ethernet_protocol = (struct ether_header *)packet;
        mac_string = (unsigned char *)ethernet_protocol->ether_dhost;//获取目的mac
        printf("Mac Destination Address is %02x:%02x:%02x:%02x:%02x:%02x\n",*(mac_string+0),*(mac_string+1),*(mac_string+2),*(ma
        mac_string = (unsigned char *)ethernet_protocol->ether_shost;//获取源mac地址
```

```
        printf("Mac Source Address is %02x:%02x:%02x:%02x:%02x:%02x\n",*(mac_string+0),*(mac_string+1),*(mac_string+2),*(mac_st

        ethernet_type = ntohs(ethernet_protocol->ether_type);//获得以太网的类型
        printf("Ethernet type is :%04x\n",ethernet_type);
        switch(ethernet_type)
        {
                case 0x0800:printf("The network layer is IP protocol\n");break;//ip
                case 0x0806:printf("The network layer is ARP protocol\n");break;//arp
                case 0x0835:printf("The network layer is RARP protocol\n");break;//rarp
                default:break;
        }


        int i;
        for(i=0; i<pkthdr->len; ++i)
        {
                printf(" %02x", packet[i]);
                if( (i + 1) % 16 == 0 )
                {
                        printf("\n");
                }
        }

        printf("\n\n");
}



void getOnePacket(pcap_t * device)
{
        struct pcap_pkthdr protocol_header;
        struct ether_header *ethernet_protocol;
        const unsigned char *p_packet_content = NULL;           // 保存接收到的数据包的起始地址
        unsigned char *p_mac_string = NULL;                     // 保存mac的地址，临时变量
        unsigned short ethernet_type = 0;                       // 以太网类型
        p_packet_content = pcap_next(device,&protocol_header);

        printf("---------------------------------------------------------------------------\n");
        printf("Capture Time is :%s",ctime((const time_t *)&protocol_header.ts.tv_sec));
        printf("Packet Lenght is :%d\n",protocol_header.len);

        /*
        *分析以太网中的 源mac、目的mac
        */
        ethernet_protocol = (struct ether_header *)p_packet_content;
        p_mac_string = (unsigned char *)ethernet_protocol->ether_shost;//获取源mac
        printf("Mac Source Address is %02x:%02x:%02x:%02x:%02x:%02x\n",*(p_mac_string+0),*(p_mac_string+1),*(p_mac_string+2),*(
        p_mac_string = (unsigned char *)ethernet_protocol->ether_dhost;//获取目的mac
        printf("Mac Destination Address is %02x:%02x:%02x:%02x:%02x:%02x\n",*(p_mac_string+0),*(p_mac_string+1),*(p_mac_string+

        /*
        *获得以太网的数据包的地址，然后分析出上层网络协议的类型
        */
        ethernet_type = ntohs(ethernet_protocol->ether_type);
        printf("Ethernet type is :%04x\t",ethernet_type);
        switch(ethernet_type)
        {
                case 0x0800:printf("The network layer is IP protocol\n");break;//ip
                case 0x0806:printf("The network layer is ARP protocol\n");break;//arp
                case 0x0835:printf("The network layer is RARP protocol\n");break;//rarp
                default:printf("The network layer unknow!\n");break;
        }
}


int main()
{
        char errBuf[PCAP_ERRBUF_SIZE], * devStr;

        /* get a device */
        devStr = pcap_lookupdev(errBuf);
```

```
        if(devStr)
        {
                printf("success: device: %s\n", devStr);
        }
        else
        {
                printf("error: %s\n", errBuf);
                exit(1);
        }

        /* open a device, wait until a packet arrives */
        pcap_t * device = pcap_open_live(devStr, 65535, 1, 1, errBuf); //设置超时时间为1ms，超时后获取数据包的函数就会立即返回

        if(!device)
        {
                printf("error: pcap_open_live(): %s\n", errBuf);
                exit(1);
        }

        /* construct a filter */
        struct bpf_program filter;
        pcap_compile(device, &filter, PACP_FILTER, 1, 0);
        pcap_setfilter(device, &filter);

        /* wait loop forever */
        int id = 0;
        pcap_loop(device, -1, getPacket, (u_char*)&id);

//      while(1)
//      {
//              getOnePacket(device);
//      }

        pcap_close(device);

        return 0;
}
```

说明：

pcap_open_live 函数的 to_ms 参数指定超时时间（毫秒）可控制 pcap_loop 函数中的 callback 函数回调时间。