知识点1【概述】（了解）

知识点2【libpcap库】（了解）

1、pcap_lookupdev

2、创建用于pcap捕获数据包的句柄（重要）

3、查看网段信息

4、接收帧数据（重要）

5、关闭pcap句柄

6、pcap_loop循环接收数据

7、设置过滤规则

1、编译规则（将用户能阅读的规则 转换成 pcap能识别的规则）

功能：

2、设置规则（将pcap识别的规则 设置到 pcap中）

3、过滤规则

8、pcap和recvfrom接收数据的区别

知识点3【libnet发送帧数据】（了解）

1、安装libnet

2、头文件

3、开发流程

知识点4【libnet的API】（了解）

1、创建libnet句柄 并内存初始化

2、释放资源

3、构建udp报文

4、构建IP报文

5、构建mac报文

libnet_ptag_t libnet_build_ethernet(

6、发送帧数据

nt libnet_write(libnet_t * l)

知识点1【概述】（了解）

libnet发原始套接字数据的库

libpcap是用来接收数据的库

知识点2【libpcap库】（了解）

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

安装：

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

开发流程：

1、打开网络设备

2、设置过滤规则（非必须）

3、捕获数据

4、关闭网络设备

gcc编译时记得加库 -lpcap　而且可执行文件记得sudo运行

1、pcap_lookupdev

```
char *pcap_lookupdev(char *errbuf)
```

功能：

得到可用的网络设备名指针

参数：

errbuf: 存放相关的错误信息

返回值：

成功返回设备名指针；失败返回NULL

2、创建用于pcap捕获数据包的句柄（重要）

```
pcap_t *pcap_open_live(const char *device,
                       int snaplen,
                       int promisc,
```

```
4                       int to_ms,char *ebuf)
```

功能：

打开一个用于捕获数据的网络接口

返回值：

返回一个Libpcap句柄

参数：

device：网络接口的名字

snaplen：捕获数据包的长度

promise：1代表混杂模式,其它非混杂模式

to_ms：等待时间

ebuf：存储错误信息

```
1 pcap_t *pcap_handle = pcap_open_live("eth0", 1500, 1, 0, NULL);
```

3、查看网段信息

```
1 int pcap_lookupnet(char *device, bpf_u_int32 *netp, bpf_u_int32 *maskp,
2 char *errbuf)
```

功能：

获得指定网络设备的网络号和掩码

参数：

device：网络设备名

netp：存放网络号的指针

maskp：存放掩码的指针

errbuf：存放出错信息

返回值：

成功返回0，失败返回-1

4、接收帧数据（重要）

```
1 const u_char *pcap_next(pcap_t *p, struct pcap_pkthdr *h)
```

功能：

捕获一个网络数据包

参数：

p：Libpcap句柄

h：数据包头

返回值：

捕获的数据包的地址

```
struct pcap_pkthdr {
    struct timeval ts;  /* time stamp */
    bpf_u_int32 caplen; /* length of portion present */
    bpf_u_int32 len;    /* length this packet (off wire) */
};
```

5、关闭pcap句柄

void pcap_close(pcap_t *p)

功能：

关闭Libpcap操作，并销毁相应的资源

参数

p:需要关闭的Libpcap句柄

返回值：

无

6、pcap_loop循环接收数据

```
int pcap_loop(pcap_t *p,int cnt,
              pcap_handler callback,
              u_char *user)
```

功能:

循环捕获网络数据包，直到遇到错误或者满足退出条件；每次捕获一个数据包就会调用callback指示的回调函数，所以，可以在回调函数中进行数据包的处理操作

返回值：

成功返回0，失败返回负数

参数:

p：Libpcap句柄

cnt：指定捕获数据包的个数，如果是-1，就会永无休止的捕获

callback：回调函数

user：向回调函数中传递的参数

```
typedef void (*pcap_handler)(u_char *, const struct pcap_pkthdr *,
            const u_char *);
```

7、设置过滤规则

1、编译规则（将用户能阅读的规则 转换成 pcap能识别的规则）

```
int pcap_compile(pcap_t *p,
              struct bpf_program *program,
              char *buf,int optimize,
```

```
4               bpf_u_int32 mask)
```

功能：

编译BPF过滤规则

返回值：

成功返回0，失败返回-1

参数:

p：Libpcap句柄

program：bpf过滤规则（pcap能识别的规则）

buf：过滤规则字符串（用户能识别的规则）

optimize：优化

mask：掩码

2、设置规则（将pcap识别的规则 设置到 pcap中）

```
int pcap_setfilter(pcap *p,struct bpf_program*fp)
```

功能：

设置BPF过滤规则

返回值：

成功返回0，失败返回-1

参数:

p：Libpcap句柄

fp：BPF过滤规则

3、过滤规则

1、mac地址过滤

ether src 11:22:33:44:55:66表示捕获源mac地址为11:22:33:44:55:66的数据包

ether dst 11:22:33:44:55:66表示捕获目的mac地址为11:22:33:44:55:66的数据包

ether host 11:22:33:44:55:66表示捕获源/目的mac地址为11:22:33:44:55:66的数据包

2、IP地址过滤

src host 192.168.0.2表示捕获源地址为192.168.0.2的数据报文

dst host 192.168.0.2表示捕获目的地址为192.168.0.2的数据报文

host 192.168.0.2表示捕获源/目的地址为192.168.0.2的数据报文

ip src host 192.168.0.2表示捕获源地址为192.168.0.2的IP协议数据报。

ip dst host 192.168.0.2表示捕获目的地址为192.168.0.2的IP协议数据报。

3、端口过滤

src port 8000表示捕获源端口为8000的数据报文

dst port 8000表示捕获目的端口为8000的数据报文

port 8000表示捕获源/目的端口为8000的数据报文

tcp src port 8000表示捕获源端口为8000的TCP数据报文

tcp dst port 8000表示捕获目的端口为8000的TCP数据报文

udp src port 8000表示捕获源端口为8000的udp数据报文

udp dst port 8000表示捕获目的端口为8000的udp数据报文

4、协议过滤

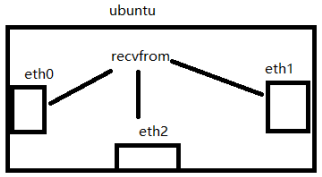
tcp udp tftp http icmp表示协议过滤

5、多条语句关系

and 表示同时成立

or 表示任意一个条件为真 则成立

8、pcap和recvfrom接收数据的区别

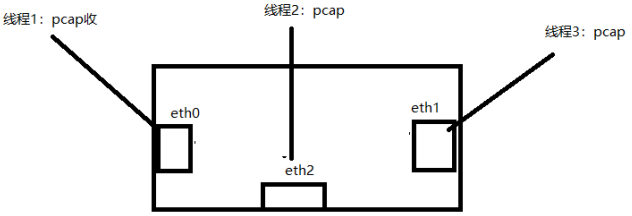


优点：

不管有多少个网络接口（eth0,eth1,eth2）只需要一个recvfrom

缺点：

通过目的mac来确定是哪个具体的设备收到了帧数据



优点：

不用确定具体哪块网卡收到数据

缺点：

多个网络接口 就需要多个线程

知识点3【libnet发送帧数据】（了解）

1、安装libnet

```
sudo apt-get install libnet-dev
```

2、头文件

gcc编译的时候加 -lnet

```
#include<libnet.h>
```

3、开发流程

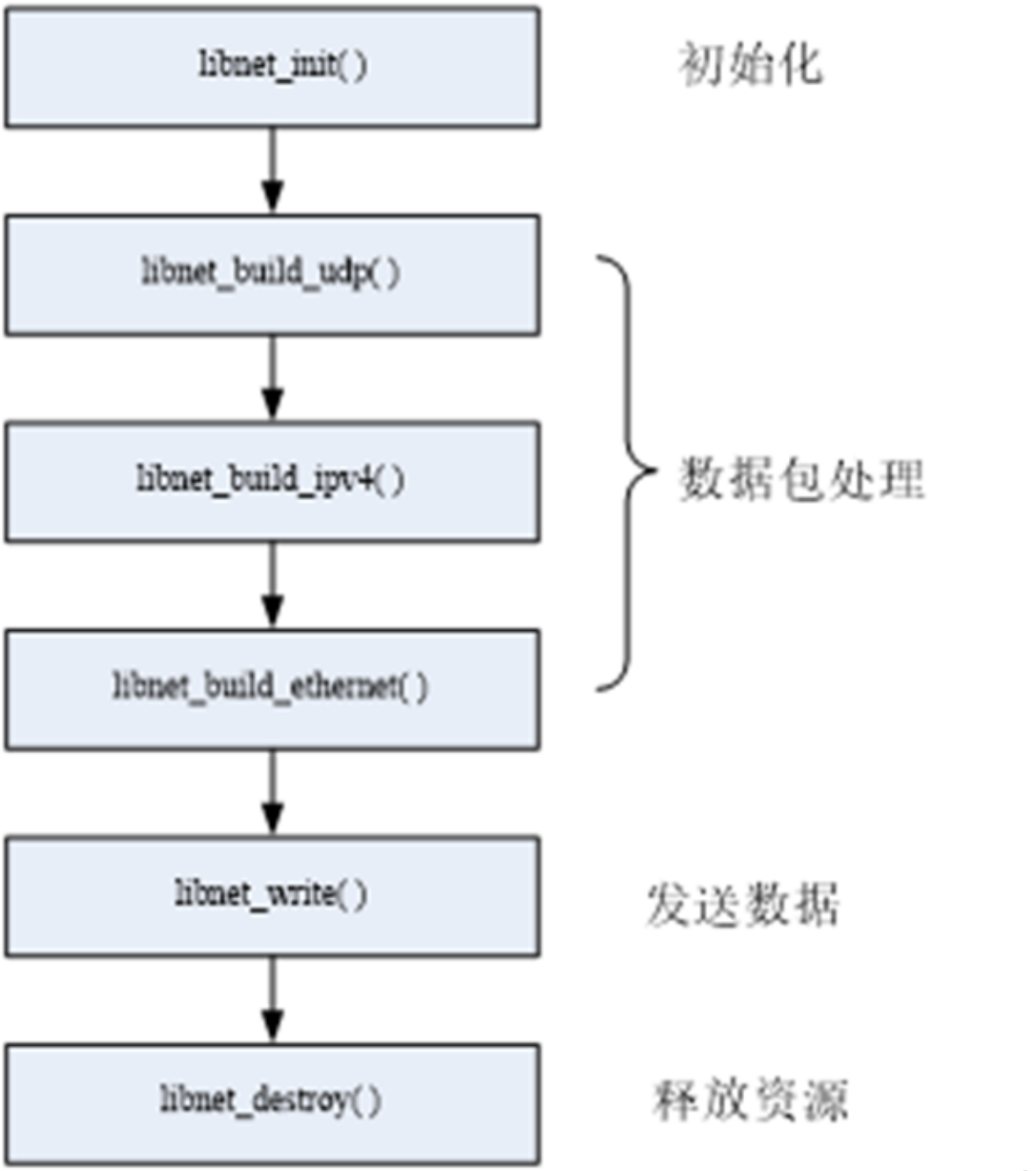
利用libnet函数库开发应用程序的基本步骤：

1、数据包内存初始化

2、构造数据包

3、发送数据

4、释放资源




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

//pcap_loop的回调函数
void get_pcap_data(u_char *user_data, const struct pcap_pkthdr *head, con
st u_char *pcap_msg)
{
    printf("帧数据的长度:%u\n", head->len);
    char src_mac[18] = "";
     char dst_mac[18] = "";
     sprintf(dst_mac, "%02x:%02x:%02x:%02x:%02x:%02x",
```

```
            pcap_msg[0], pcap_msg[1], pcap_msg[2], pcap_msg[3], pcap_msg
cap_msg[5]);
    sprintf(src_mac, "%02x:%02x:%02x:%02x:%02x:%02x",
            pcap_msg[0 + 6], pcap_msg[1 + 6], pcap_msg[2 + 6], pcap_msg[
cap_msg[4 + 6], pcap_msg[5 + 6]);
    printf("%s---->%s\n", src_mac, dst_mac);
}
int main(int argc, char const *argv[])
{
    //获取网络设备名（不要）
    char *dev_name = pcap_lookupdev(NULL);
    printf("dev_name=%s\n", dev_name);

    //创建pcap句柄
    pcap_t *pcap_handle = pcap_open_live("eth0", 1500, 1, 0, NULL);

    //查看网段信息（不要）
    unsigned int net_id = 0;
    unsigned int mask_id = 0;
    pcap_lookupnet("eth0", &net_id, &mask_id, NULL);
    char net_buf[16] = "";
    char mask_buf[16] = "";
    inet_ntop(AF_INET, &net_id, net_buf, 16);
    inet_ntop(AF_INET, &mask_id, mask_buf, 16);
    printf("网段信息: %s 掩码信息: %s\n", net_buf, mask_buf);

    //编译过滤规则
    struct bpf_program bpf;
    char *p = "ip src 10.9.21.211 and udp src port 8000";
    pcap_compile(pcap_handle, &bpf, p, 0, 0xffffff00);

    //设置过滤规则
    pcap_setfilter(pcap_handle, &bpf);

#if 0
    //接收数据
    while (1)
    {
        struct pcap_pkthdr head;
        unsigned char *pcap_msg = pcap_next(pcap_handle, &head);
        printf("len = %u\n", head.len);
```

```
51
52          char src_mac[18] = "";
53          char dst_mac[18] = "";
54          sprintf(dst_mac, "%02x:%02x:%02x:%02x:%02x:%02x",
55                  pcap_msg[0], pcap_msg[1], pcap_msg[2], pcap_msg[3], pcap
_msg[4], pcap_msg[5]);
56          sprintf(src_mac, "%02x:%02x:%02x:%02x:%02x:%02x",
57                  pcap_msg[0 + 6], pcap_msg[1 + 6], pcap_msg[2 + 6], pcap_
msg[3 + 6], pcap_msg[4 + 6], pcap_msg[5 + 6]);
58          printf("%s---->%s\n", src_mac, dst_mac);
59      }
60  #else
61      //接收数据 阻塞
62      pcap_loop(pcap_handle, -1, get_pcap_data, "hehe");
63  #endif
64      //关闭
65      pcap_close(pcap_handle);
66
67      return 0;
68  }
69
```

知识点4【libnet的API】（了解）

1、创建libnet句柄 并内存初始化

```
1  libnet_t *libnet_init(int injection_type, char *device, char *err_buf)
```

功能：

数据包内存初始化及环境建立

参数：

injection_type: 构造的类型(LIBNET_LINK,LIBNET_RAW4，LIBNET_LINK_ADV,LIBNET_RAW4_ADV)

device：网络接口，如"eth0",或IP地址，亦可为NULL(自动查询搜索)

err_buf: 存放出错的信息

返回值：

成功返回一个libnet句柄；失败返回NULL

2、释放资源

```
1  void libnet_destroy(libnet_t *l);
```

功能：

释放资源

参数：

l: libnet句柄

返回值：

无

3、构建udp报文

```
libnet_ptag_t libnet_build_udp(
        u_int16_t sp,u_int16_t dp,
        u_int16_t len,u_int16_t sum,
        u_int8_t *payload,u_int32_t payload_s,
        libnet_t *l,libnet_ptag_t ptag)
```

功能：

构造udp数据包

返回值：

成功返回协议标记；失败返回-1

参数：

sp: 源端口号

dp：目的端口号

len：udp包总长度

sum：校验和，设为0，libnet自动填充

payload：负载，可设置为NULL

payload_s：负载长度，或为0

l: libnet句柄

ptag：协议标记（第一次创建报文记得赋值为0）

返回值：

成功返回协议标记；失败返回-1

4、构建IP报文

```
libnet_ptag_t libnet_build_ipv4(
        u_int16_t ip_len,u_int8_t tos,
        u_int16_t id,u_int16_t flag,
        u_int8_t ttl,u_int8_t prot,
        u_int16 sum,u_int32_t src,
```

```
        u_int32_t dst,u_int8_t *payload,
        u_int32_t payload_s,
        libnet_t *l,libnet_ptag_t ptag)
```

功能：

构造一个IPv4数据包

参数：

ip_len：ip包总长

tos：服务类型

id：ip标识

flag：片偏移

ttl：生存时间

prot：上层协议

sum：校验和，设为0，libnet自动填充

src: 源ip地址

dst：目的ip地址

payload：负载，可设置为NULL

payload_s：负载长度，或为0

l: libnet句柄

ptag：协议标记

返回值：

成功返回协议标记；失败返回-1

5、构建mac报文

```
libnet_ptag_t libnet_build_ethernet(
            u_int8_t *dst,u_int8_t *src,
            u_int16_t type,
            u_int8_t *payload,
            u_int32_t payload_s,
            libnet_t *l,libnet_ptag_t ptag)
```

功能：

构造一个以太网数据包

参数：

dst：目的mac

src：源mac

type：上层协议类型

payload：负载，即附带的数据

payload_s：负载长度

l：libnet句柄

ptag：协议标记

返回值：

成功返回协议标记；失败返回-1

6、发送帧数据

nt libnet_write(libnet_t * l)

功能：

发送数据到网络

参数：

l：libnet句柄

返回值：

失败返回-1，成功返回其他

7、综合案例

```c
#include <stdio.h>
#include <libnet.h>
#include <string.h>
int main(int argc, char const *argv[])
{
    //初始化内存
    libnet_t *libnet = libnet_init(LIBNET_LINK_ADV, "eth0", NULL);

    libnet_ptag_t udp_tag = 0;
     libnet_ptag_t ip_tag = 0;
     libnet_ptag_t eth_tag = 0;
     while (1)
     {
         //获取数据
         char data[128] = "";
```

```
        printf("请输入你要发送的数据:");
        fgets(data, sizeof(data), stdin);
        data[strlen(data) - 1] = 0;
        int data_len = strlen(data);

        //判断退出
        if (strcmp(data, "bye") == 0)
            break;

        //构建udp报文
        udp_tag = libnet_build_udp(
            8000,          /*源端口*/
            8000,          /*目的端口*/
            8 + data_len, /*udp总长度*/
            0,             /*libnet自动填充校验和*/
            data,          /*需要发送的数据*/
            data_len,      /*需要发送的数据的长度*/
            libnet,        /*libnet的句柄*/
            udp_tag        /*第一构建报文，协议标记记得写0*/
        );

        //构建IP报文
        ip_tag = libnet_build_ipv4(
            20 + 8 + data_len,         /*IP报文总长度*/
            0,                         /*服务类型*/
            0,                         /*标识*/
            0,                         /*片偏移*/
            128,                       /*ttl生命周期*/
            17,                        /*上层协议*/
            0,                         /*IP头部校验，写0自动填充*/
            inet_addr("10.9.21.201"), /*源IP地址 ubuntu*/
            inet_addr("10.9.21.211"), /*目的IP地址windows*/
            NULL,                      /*负载 填空 自动拼接udp报文*/
            0,                         /*负载长度写0,自动填充*/
            libnet,                    /*libnet句柄*/
            ip_tag                     /*协议标记 第一构建写0*/
        );

        //构建链路层报文
        unsigned char src_mac[6] = {0x00, 0x0c, 0x29, 0x6e, 0x18, 0x47};
ubuntu的mac地址
```

```
56          unsigned char dst_mac[6] = {0x3c, 0x7c, 0x3f, 0x5f, 0x60, 0x7c};
windows的mac地址
57          eth_tag = libnet_build_ethernet(
58              dst_mac, /*目的主机的mac地址*/
59              src_mac, /*源mac地址*/
60              0x0800,  /*上层协议为ip报文*/
61              NULL,    /*负载*/
62              0,       /*负载长度0*/
63              libnet,  /*libnet句柄*/
64              eth_tag  /*协议标记*/
65          );
66
67          //发送帧数据
68          libnet_write(libnet);
69      }
70
71      //释放句柄的资源
72      libnet_destroy(libnet);
73      return 0;
74  }
```

```
edu@edu:~/work/net/day05$ cd ..
edu@edu:~/work/net$ ls
day01  day02  day03  day04  day05  day06
edu@edu:~/work/net$ cd day06
edu@edu:~/work/net/day06$ ls
00_pcap.c  01_libnet.c  a.out
edu@edu:~/work/net/day06$ gcc 01_libnet.c  -lnet
edu@edu:~/work/net/day06$ sudo ./a.out
edu@edu:~/work/net/day06$ gcc 01_libnet.c  -lnet
edu@edu:~/work/net/day06$ sudo ./a.out
请输入你要发送的数据:nihao
请输入你要发送的数据:hehe haha
请输入你要发送的数据:asdfas
请输入你要发送的数据:dfasd
请输入你要发送的数据:fasd
请输入你要发送的数据:fasdfa
请输入你要发送的数据:sdfas
请输入你要发送的数据:dfasdf
请输入你要发送的数据:bye
edu@edu:~/work/net/day06$
```