# 四、网络嗅探器的设计与实现

参照附录4 raw socket编程例子，设计一个可以监视网络的状态，数据流动情况以及网络上传输的信息的网络嗅探器。

## 课程设计内容

C语言getop()函数:分析命令行参数

> 头文件 #include <unistd.h>
>
> 定义函数：int getopt(int argc, char * const argv[], const char * optstring);
>
> 函数说明：getopt()用来分析命令行参数。 1、参数argc 和argv 是由main()传递的参数个数和内容。 2、参数optstring 则代表欲处理的选项字符串。
>
> 此函数会返回在argv 中下一个的选项字母，此字母会对应参数optstring 中的字母。
>
> 如果选项字符串里的字母后接着冒号":"，则表示还有相关的参数，全域变量optarg 即会指向此额外参数。 如果getopt()找不到符合的参数则会印出错信息，并将全域变量optopt 设为"?"字符, 如果不希望getopt()印出错信息，则只要将全域变量opterr 设为0 即可。
>
> 返回值：如果找到符合的参数则返回此参数字母, 如果参数不包含在参数optstring 的选项字母则返回"?"字符, 分析结束则返回-1.
>
> 范例 #include <stdio.h> #include <unistd.h> int main(int argc, char **argv) { int ch; opterr = 0; while((ch = getopt(argc, argv, "a:bcde")) != -1) switch(ch) {  case 'a':  printf("option a:'%s'\n", optarg); break;  case 'b':  printf("option b :b\n"); break;  default:  printf("other option :%c\n", ch); } printf("optopt +%c\n", optopt); }
>
> 执行： $. /getopt -b option b:b $. /getopt -c other option:c $. /getopt -a other option :? $. /getopt -a12345 option a:'12345'

## 课程设计结果及结果分析 心得体会

通过原生套接字的方式，监听所有本地主机收发的数据链路层帧结构，然后解析数据包的类型，并记录到日志文件。实现一个轻量级的网络嗅探器。

通过完成这个基于 Linux 操作系统的网络嗅探器，可以更加深入的了解套接字，尤其是原始套接字。另外也可以对 TCP/IP 协议栈有更深入的了解。

### 需要注意的地方:

> 原始套接字**SOCKET_RAW**的获取必须获得**root**权限，否则返回-1

## 程序源文件主要代码展示

**程序运行截图**


**程序主要代码展示**

```
int snifer(int sock, int pPort,  char *pprototype, char *pipaddr)
{
    if (pPort > 0)
        printf("pPort:%d\n", pPort);
    if (pprototype != NULL)
        printf("pprototype:%s\n", pprototype);
//    if (pipaddr != NULL)
//        printf("pipaddr:%s\n", pipaddr);

    int n_read, proto;
    char buffer[BUFFER_MAX];
    char *ethhead, *iphead, *tcphead, *udphead, *icmphead, *p;
    unsigned int sport = 0; // source port
    unsigned int dport = 0; // destination prot

    n_read = recvfrom(sock, buffer, 2048, 0, NULL, NULL);
    /*
    14  6(dest)+6(source)+2(type or length) // MAC dst(6) + src(6) + type(2)  (p96)
    +
    20  ip header                           // IP static(20)
    +
    8  icmp, tcp or udp header             // UDP static(8)
    = 42
    */
    if (n_read < 42)
    {
        fprintf(stdout, "Incomplete header, packet corrupt\n");
```

```
        return -2;
    }

    ethhead = buffer;
    p = ethhead;
    int n = 0XFF; // 255

    // MAC address
    // printf("MAC:%.2X:%.02X:%02X:%02X:%02X:%02X=>"
    //        "%.2X:%.2X:%.2X:%.2X:%.2X:%.2X\n",
    //        p[6]&n, p[7]&n, p[8]&n, p[9]&n, p[10]&n, p[11]&n, p[0]&n, p[1]&n, p[2]&n,
p[3]&n, p[4]&n, p[5]&n);
    int mac_src_addr[6];
    int mac_dst_addr[6];
    mac_src_addr[0] = p[6] & n;
    mac_src_addr[1] = p[7] & n;
    mac_src_addr[2] = p[8] & n;
    mac_src_addr[3] = p[9] & n;
    mac_src_addr[4] = p[10] & n;
    mac_src_addr[5] = p[11] & n;

    mac_dst_addr[0] = p[0] & n;
    mac_dst_addr[1] = p[1] & n;
    mac_dst_addr[2] = p[2] & n;
    mac_dst_addr[3] = p[3] & n;
    mac_dst_addr[4] = p[4] & n;
    mac_dst_addr[5] = p[5] & n;

    iphead = ethhead + 14; // position IP frame
    p = iphead + 12; // position src & dst

    // IP address
    // printf("IP:%d.%d.%d.%d => %d.%d.%d.%d\n",
    //        p[0] & 0XFF, p[1] & 0XFF, p[2] & 0XFF, p[3] & 0XFF, p[4] & 0XFF, p[5] &
0XFF, p[6] & 0Xff, p[7] & 0XFF);
    int ip_src_addr[4];
    int ip_dst_addr[4];
    ip_src_addr[0] = p[0] & n;
    ip_src_addr[1] = p[1] & n;
    ip_src_addr[2] = p[2] & n;
    ip_src_addr[3] = p[3] & n;
    ip_dst_addr[0] = p[4] & n;
    ip_dst_addr[1] = p[5] & n;
    ip_dst_addr[2] = p[6] & n;
    ip_dst_addr[3] = p[7] & n;

    // diff with pipaddr
    if (pipaddr != NULL)
    {
        char str_src[15] = {'\0'};
        char str_dst[15] = {'\0'};
        int i, L_src, L_dst;
        for (i = 0; i < 4; i++)
```

```
        {
            L_src = strlen(str_src);
            L_dst = strlen(str_dst);
            if (i < 3)
            {
                sprintf(str_src + L_src, "%d.", ip_src_addr[i]);
                sprintf(str_dst + L_dst, "%d.", ip_dst_addr[i]);
            }
            else
            {
                sprintf(str_src + L_src, "%d", ip_src_addr[i]);
                sprintf(str_dst + L_dst, "%d", ip_dst_addr[i]);
            }
        };
        // if str_src equal pipaddr or str_dst equal pipaddr
        if (strcmp(str_src, pipaddr) != 0 && strcmp(str_dst, pipaddr) != 0)
        {
            return -1;
        }
    }

    proto = (iphead + 9)[0]; // position Protocol (p130)
    p = iphead + 20; // position TCP/UDP frame

    // Protocol
    // get Port
    if (proto == IPPROTO_TCP)
    {
        sport = (p[0] << 8) & 0XFF00 | p[1] & 0XFF;
        dport = p[2] << 8 & 0XFF00 | p[3] & 0XFF;
    }
    else if (proto == IPPROTO_UDP)
    {
        sport = (p[0] << 8) & 0XFF00 | p[1] & 0XFF;
        dport = p[2] << 8 & 0XFF00 | p[3] & 0XFF;
    }

    printf("Protocol:");
    switch (proto) // int
    {
    case IPPROTO_ICMP:
        printf("\033[47;35mICMP\033[0m\n");
        break;
    case IPPROTO_IGMP:
        printf("\033[47;35mIGMP\033[0m\n");
        break;
    case IPPROTO_IPIP:
        printf("\033[47;35mIPIP\033[0m\n");
        break;
    case IPPROTO_TCP:
    case IPPROTO_UDP:
        if (pPort > 0)
        {
```

```
            if (pPort != sport)
                if (pPort != dport)
                    return -1;
        }
        printf("\033[47;35m%s\033[0m,", proto == IPPROTO_TCP ? "TCP" : "UDP");
        printf("source port:\033[;33m%u\033[0m,", sport);
        printf("dest port:\033[;33m%u\033[0m\n", dport);
        break;
    case IPPROTO_RAW:
        printf("\033[47;35mRAW\033[0m\n");
        break;
    default:
        printf("\033[47;41mUnkown\033[0m, please query in include/linux/in.h\n");
    }

    // Output
    // MAC adrress
    printf("MAC:%.2X:%.02X:%02X:%02X:%02X:%02X => "
           "%.2X:%.2X:%.2X:%.2X:%.2X:%.2X\n",
           mac_src_addr[0], mac_src_addr[1], mac_src_addr[2], mac_src_addr[3],
mac_src_addr[4], mac_src_addr[5], mac_dst_addr[0], mac_dst_addr[1], mac_dst_addr[2],
mac_dst_addr[3], mac_dst_addr[4], mac_dst_addr[5]);

    // IP address
    printf("IP:%d.%d.%d.%d => %d.%d.%d.%d\n",
           ip_src_addr[0], ip_src_addr[1], ip_src_addr[2], ip_src_addr[3],
ip_dst_addr[0], ip_dst_addr[1], ip_dst_addr[2], ip_dst_addr[3]);


    return 0;
}
```