虚拟网卡 TUN/TAP 驱动程序设计原理

麻利辉 (mlhdyx@21cn.com), 研究生，电子科技大学计算机学院研究生部

简介： 本文将介绍 TUN/TAP 驱动的使用并分析虚拟网卡 TUN/TAP 驱动程序在 Linux 环境下的设计思路。

发布日期： 2004 年 11 月 01 日
级别： 初级
访问情况 ： 9941 次浏览
评论： 0 (查看 | 添加评论 - 登录)

平均分 (18个评分)
为本文评分

简介

虚拟网卡Tun/tap驱动是一个开源项目，支持很多的类UNIX平台，OpenVPN和Vtun都是基于它实现隧道包封装。本文将介绍tun/tap驱动的使用并分析虚拟网卡tun/tap驱动程序在linux环境下的设计思路。

tun/tap驱动程序实现了虚拟网卡的功能，tun表示虚拟的是点对点设备，tap表示虚拟的是以太网设备，这两种设备针对网络包实施不同的封装。利用tun/tap驱动，可以将tcp/ip协议栈处理好的网络分包传给任何一个使用tun/tap驱动的进程，由进程重新处理后再发到物理链路中。开源项目openvpn（ http://openvpn.sourceforge.net）和Vtun(http://vtun.sourceforge.net)都是利用tun/tap驱动实现的隧道封装。

使用tun/tap驱动

在linux 2.4内核版本及以后版本中，tun/tap驱动是作为系统默认预先编译进内核中的。在使用之前，确保已经装载了tun/tap模块并建立设备文件：

```
#modprobe tun
#mknod /dev/net/tun c 10 200
```

参数c表示是字符设备，10和200分别是主设备号和次设备号。

这样，我们就可以在程序中使用该驱动了。

使用tun/tap设备的示例程序(摘自openvpn开源项目 http://openvpn.sourceforge.net，tun.c文件)

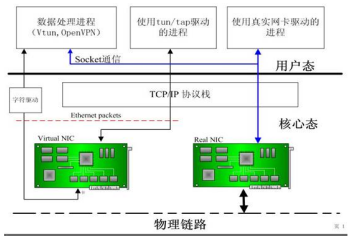
```
int open_tun (const char *dev, char *actual, int size)
{
  struct ifreq ifr;
  int fd;
  char *device = "/dev/net/tun";
  if ((fd = open (device, O_RDWR)) < 0) //创建描述符
    msg (M_ERR, "Cannot open TUN/TAP dev %s", device);
  memset (&ifr, 0, sizeof (ifr));
  ifr.ifr_flags = IFF_NO_PI;
  if (!strncmp (dev, "tun", 3)) {
      ifr.ifr_flags |= IFF_TUN;
   }
  else if (!strncmp (dev, "tap", 3)) {
      ifr.ifr_flags |= IFF_TAP;
    }
  else {
    msg (M_FATAL, "I don't recognize device %s as a TUN or TAP device",dev);
    }
  if (strlen (dev) > 3)         /* unit number specified? */
    strncpy (ifr.ifr_name, dev, IFNAMSIZ);
  if (ioctl (fd, TUNSETIFF, (void *) &ifr) < 0) //打开虚拟网卡
    msg (M_ERR, "Cannot ioctl TUNSETIFF %s", dev);
  set_nonblock (fd);
  msg (M_INFO, "TUN/TAP device %s opened", ifr.ifr_name);
  strncpynt (actual, ifr.ifr_name, size);
  return fd;
}
```

调用上述函数后，就可以在shell命令行下使用ifconfig 命令配置虚拟网卡了，通过生成的字符设备描述符，在程序中使用read和write函数就可以读取或者发送给虚拟的网卡数据了。

Tun/tap驱动程序工作原理

做为虚拟网卡驱动，Tun/tap驱动程序的数据接收和发送并不直接和真实网卡打交道，而是通过用户态来转交。在linux下，要实现核心态和用户态数据的交互，有多种方式：可以通用socket创建特殊套接字，利用套接字实现数据交互；通过proc文件系统创建文件来进行数据交互；还可以使用设备文件的方式，访问设备文件会调用设备驱动相应的例程，设备驱动本身就是核心态和用户态的一个接口，Tun/tap驱动就是利用设备文件实现用户态和核心态的数据交互。

从结构上来说，Tun/tap驱动并不单纯是实现网卡驱动，同时它还实现了字符设备驱动部分。以字符设备的方式连接用户态和核心态。下面是示意图：



Tun/tap驱动程序中包含两个部分，一部分是字符设备驱动，还有一部分是网卡驱动部分。利用网卡驱动部分接收来自TCP/IP协议栈的网络分包并发送或者反过来将接收到的网络分包传给协议栈处理，而字符驱动部分则将网络分包在内核与用户态之间传送，模拟物理链路的数据接收和发送。Tun/tap驱动很好的实现了两种驱动的结合。

下面是定义的tun/tap设备结构：

```
struct tun_struct {
        char                    name[8]; //设备名
        unsigned long           flags; //区分tun和tap设备
        struct fasync_struct    *fasync; //文件异步通知结构
        wait_queue_head_t       read_wait; //等待队列
        struct net_device       dev;  //linux 抽象网络设备结构
        struct sk_buff_head     txq; //网络缓冲区队列
    struct net_device_stats     stats; //网卡状态信息结构
};
```

struct net_device结构是linux内核提供的统一网络设备结构，定义了系统统一的访问接口。

Tun/tap驱动中实现的网卡驱动的处理例程：

```
static int tun_net_open(struct net_device *dev);
static int tun_net_close(struct net_device *dev);
static int tun_net_xmit(struct sk_buff *skb, struct net_device *dev); //数据包发送例程
static void tun_net_mclist(struct net_device *dev); //设置多点传输的地址链表
static struct net_device_stats *tun_net_stats(struct net_device *dev); //当一个应用程序需要知道 网络接口的一些统计数据时，可调用该函数，如ifconfig、netstat等。
int tun_net_init(struct net_device *dev); //网络设备初始例程
```

字符设备部分：

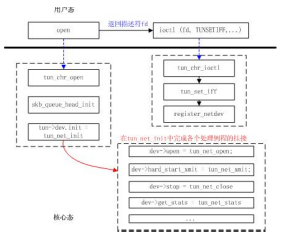
在Linux中，字符设备和块设备统一以文件的方式访问，访问它们的接口是统一的，都是使用open()函数打开设备文件或普通文件，用read()和write()函数实现读写文件等等。Tun/tap驱动定义的字符设备的访问接口如下：

```
static struct file_operations tun_fops = {
owner: THIS_MODULE,
llseek: tun_chr_lseek,
read tun_chr_read,
write: tun_chr_write,
poll: tun_chr_poll,
ioctl: tun_chr_ioctl,
open: tun_chr_open,
release: tun_chr_close,
fasync: tun_chr_fasync
};
```

在内核中利用misc_register() 函数将该驱动注册为非标准字符设备驱动，提供字符设备具有的各种程序接口。代码摘自linux-2.4.20\linux-2.4.20\drivers\net\tun.c

```
static struct miscdevice tun_miscdev=
{
        TUN_MINOR,
        "net/tun",
        &tun_fops
};
int __init tun_init(void)
{
        ...
        if (misc_register(&tun_miscdev)) {
                printk(KERN_ERR "tun: Can't register misc device %d\n", TUN_MINOR);
                return -EIO;
        }
        return 0;
}
```

当打开一个tun/tap设备时，open 函数将调用tun_chr_open()函数，其中将完成一些重要的初始化过程，包括设置网卡驱动部分的初始化函数以及网络缓冲区链表的初始化和等待队列的初始化。Tun/tap驱动中网卡的注册被嵌入了字符驱动的ioctl例程中，它是通过对字符设备文件描述符利用自定义的ioctl设置标志TUNSETIFF完成网卡的注册的。下面是函数调用关系示意图：



使用ioctl()函数操作字符设备文件描述符,将调用字符设备中tun_chr_ioctl 来设置已经open好的tun/tap设备，如果设置标志为TUNSETIFF，则调用tun_set_iff() 函数，此函数将完成很重要的一步操作，就是对网卡驱动进行注册register_netdev(&tun->dev)，网卡驱动的各个处理例程的挂接在open操作时由tun_chr_open()函数初始化好了。

Tun/tap设备的工作过程：

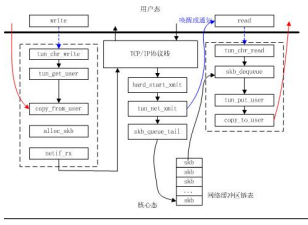
Tun/tap设备提供的虚拟网卡驱动，从tcp/ip协议栈的角度而言，它与真实网卡驱动并没有区别。从驱动程序的角度来说，它与真实网卡的不同表现在tun/tap设备获取的数据不是来自物理链路，而是来自用户区，Tun/tap设备驱动通过字符设备文件来实现数据从用户区的获取。发送数据时tun/tap设备也不是发送到物理链路，而是通过字符设备发送至用户区，再由用户区程序通过其他渠道发送。

发送过程：

使用tun/tap网卡的程序经过协议栈把数据传送给驱动程序，驱动程序调用注册好的hard_start_xmit函数发送，hard_start_xmit函数又会调用tun_net_xmit函数，其中skb将会被加入skb链表，然后唤醒被阻塞的使用tun/tap设备字符驱动读数据的进程，接着tun/tap设备的字符驱动部分调用其tun_chr_read()过程读取skb链表，并将每一个读到的skb发往用户区，完成虚拟网卡的数据发送。

接收数据的过程：

当我们使用write()系统调用向tun/tap设备的字符设备文件写入数据时，tun_chr_write函数将被调用，它使用tun_get_user从用户区接受数据，其中将数据存入skb中，然后调用关键的函数netif_rx(skb) 将skb送给tcp/ip协议栈处理，完成虚拟网卡的数据接收。



小结

tun/tap驱动很巧妙的将字符驱动和网卡驱动糅合在一起，本文重点分析两种驱动之间的衔接，具体的驱动处理细节没有一一列出，请读者参考相关文档。

参考资料

- 《Linux 设备驱动程序 第二版》，(美)Alessando Rubini
- http://openvpn.sourceforge.net
- Linux串口上网的简单实现
- linux 源代码