努力成为 linux kernel hacker 的人李万鹏原创作品，为梦而战。转载请标明出处

努力成为 linux kernel hacker 的人李万鹏原创作品，为梦而战。转载请标明出处
http://blog.csdn.net/woshixingaaa/archive/2011/06/18/6552908.aspx

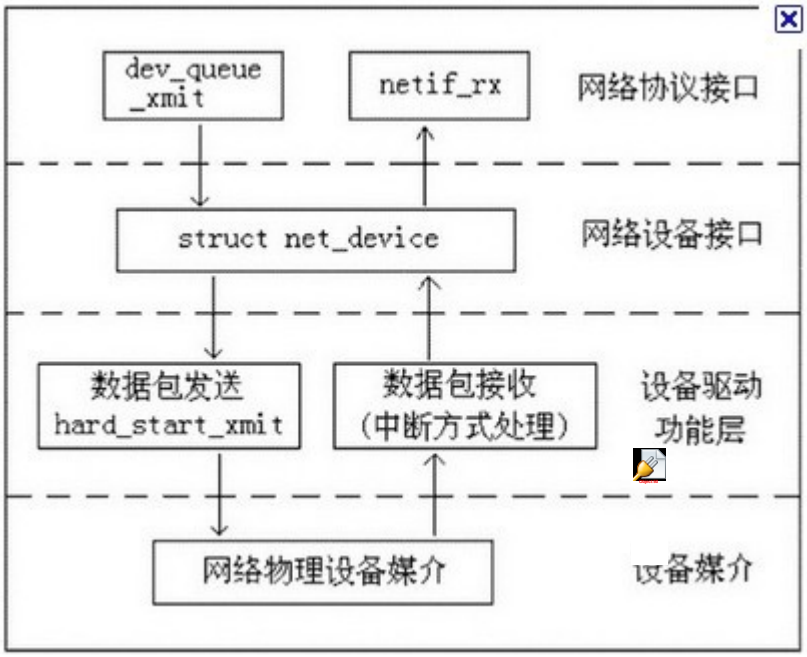
更多信息请看：

Linux 驱动修炼之道-DM9000A 网卡驱动框架源码分析(中)

Linux 驱动修炼之道-DM9000A 网卡驱动框架源码分析(下)

Linux 驱动修炼之道

首先分析一下 Linux 网络设备的结构，如下图：



1.网络协议接口层向网络层协议提供提供统一的数据包收发接口，不论上层协议为 ARP 还是 IP，都通过 dev_queue_ ‘)函数发送数据，并通过 netif_rx()函数接受数据。这一层的存在使得上层协议独立于具体的设备。

2.网络设备接口层向协议接口层提供统一的用于描述具体网络设备属性和操作的结构体 net_device，该结构体是设备驱动功能层中各函数的容器。实际上，网络设备接口层从宏观上规划了具体操作硬件的设备驱动功能层的结构。

3.设备驱动功能层各函数是网络设备接口层 net_device 数据结构的具体成员，是驱使网络设备硬件完成相应动作的程序，通过 hard_start_xmit()函数启动发送操作，并通过网络设备上的中断触发接受操作。

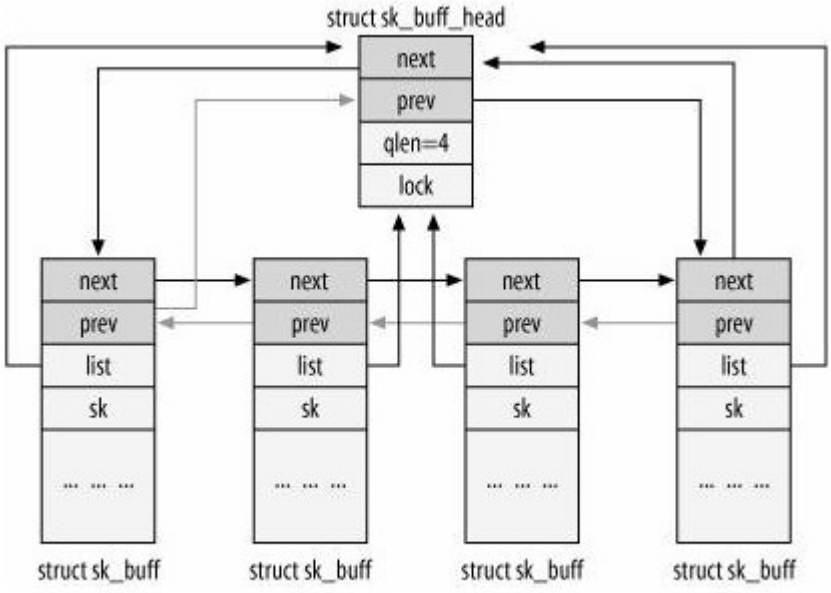
4.网络设备与媒介层是完成数据包发送/接受的物理实体，包括网络适配器和具体的传输媒介，网络适配器被驱动功能层中的函数物理上驱动。对于 Linux 系统而言，网络设备和媒介都可以是虚拟的。

网络协议接口层：

这里主要进行数据包的收发，使用函数原型为：

```
dev_queue_xmit(struct sk_buff *skb);int netif_rx(struct sk_buff *skb);
```

这里使用了一个 skb_buff 结构体，定义于 include/linux/skbuff.h 中，它的含义为“套接字缓冲区”，用于在 Linux 网络子系统各层间传输数据。他是一个双向链表，在老的内核中会有一个 list 域指向 sk_buff_head 也就是链表头，但是在我研究的 linux2.6.30.4 内核中已经不存在了，如下图：



操作套接字缓冲区的函数：

1.分配

```
struct sk_buff *alloc_skb(unsigned int len, int priority);struct sk_buff *dev_alloc_skb(unsigned int len);
```

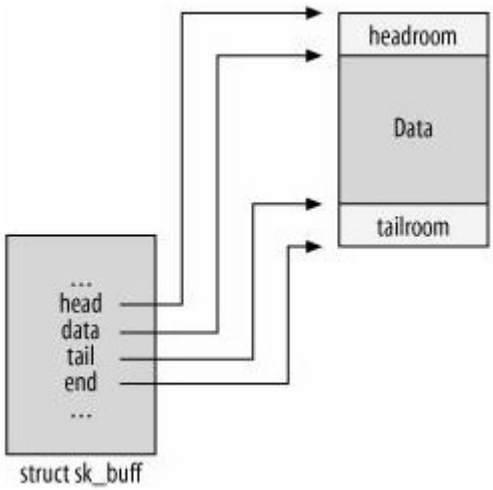
分配一个缓冲区。alloc_skb 函数分配一个缓冲区并初始化 skb->data 和 skb->tail 为 skb->head。参数 len 为数据缓冲区的空间大小，通常以 L1_CACHE_BYTES 字节(对 ARM 为 32)对齐，参数 priority 为内存分配的优先级。dev_alloc_skb()函数以 GFP_ATOMIC 优先级进行 skb 的分配。

2.释放

```
void kfree_skb(struct sk_buff *skb);void dev_kfree_skb(struct sk_buff *skb);
```

Linux 内核内部使用 kfree_skb()函数，而网络设备驱动程序中则最好使用 dev_kfree_skb()。

sk_buff 中比较重要的成员是指向数据包中数据的指针，如下图所示：

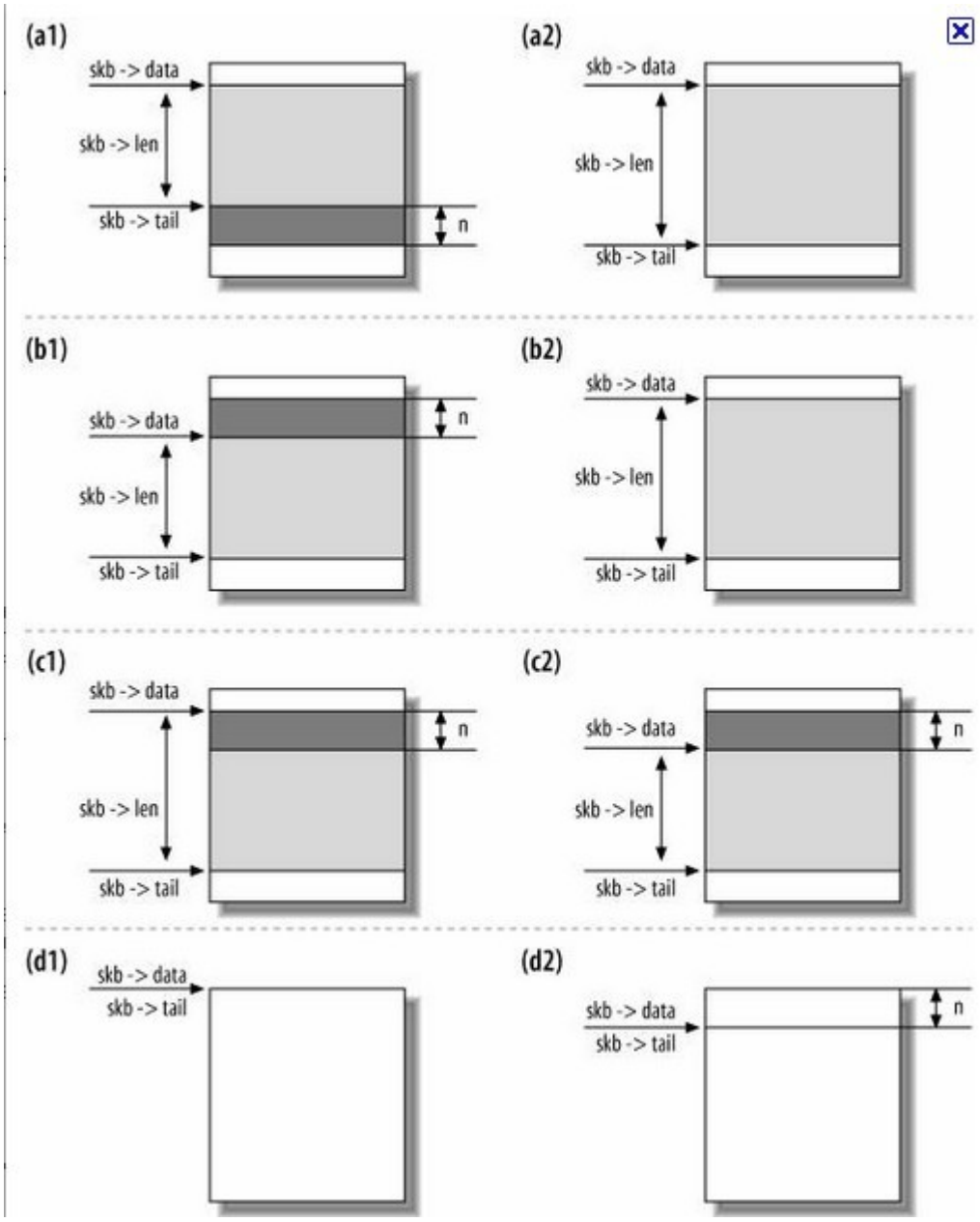


用于寻址数据包中数据的指针，head 指向已分配空间开头，data 指向有效的 octet 开头，tail 指向有效的 octet 结尾，而 end 指向 tail 可以到达的最大地址。如果不这样做而分配一个大小固定的缓冲区，如果 buffer 不够用，则要申请一个更大的 buffer，拷贝进去再增加，这样降低了性能。

3.变更

1. unsigned **char** *skb_put(**struct** sk_buff *skb, **int** len);unsigned **char** *skb_push(**struct** sk_buff *skb, **int** len);unsigned **char** *skb_pull(**struct** sk_buff *skb, **int** len);**void** skb_reserve(**struct** sk_buff ×skb, **int** len);

下图分别对应了这四个函数，看了这张图应该对这 4 个函数的作用了然于胸。



网络设备接口层：

网络设备接口层的主要功能是为千变万化的网络设备定义了统一，抽象的数据结构 net_device 结构体，以不变应万变，实现多种硬件在软件层次上的统一。

首先看打开和关闭网络设备的函数：

```
1. int (*open)(struct net_device *dev);int (*close)(struct net_device *dev);
```

要注意的是 ifconfig 是 interface config 的缩写，通常我们在用户空间输入:

```
1. ifconfig eth0 up
```

会调用这里的 open 函数。

在用户空间输入：

```
1. ifconfig eth0 down
```

会调用这里的 stop 函数。

在使用 ifconfig 向接口赋予地址时，要执行两个任务。首先，它通过 ioctl(SIOCSIFADDR)(Socket I/O Control Set Interface Address)赋予地址，然后通过 ioctl(SIOCSIFFLAGS)(Socket I/O Control Set Interface Flags)设置 dev->flag 中的 IFF_UP 标志以打开接口。这个调用会使得设备的 open 方法得到调用。类似的，在接口关闭时，ifconfig 使用 ioctl(SIOCSIFFLAGS)来清理 IFF_UP 标志，然后调用 stop 函数。

```
1. int (*hard_header)(struct sk_buff *skb, struct net_device *dev, unsigned short type, void *dadd
   r, void *saddr, unsigned len);
```

该方法根据先前检索到的源和目的硬件地址建立硬件头。

```
1. int (*rebuild_header)(struct sk_buff *skb);
```

以太网的 mac 地址是固定的，为了高效，第一个包去询问 mac 地址，得到对应的 mac 地址后就会作为 cache 把 mac 地址保存起来。以后每次发包不用询问了，直接把包的地址拷贝出来。

```
1. void (*tx_timeout)(struct net_device *dev);
```

如果数据包发送在超时时间内失败，这时该方法被调用，这个方法应该解决失败的问题，并重新开始发送数据。

```
1. struct net_device_stats *(*get_stats)(struct net_device *dev);
```

当应用程序需要获得接口的统计信息时，这个方法被调用。

```
1. int (*set_config)(struct net_device *dev, struct ifmap *map);
```

改变接口的配置，比如改变 I/O 端口和中断号等，现在的驱动程序通常无需该方法。

```
1. int (*do_ioctl)(struct net_device *dev, struct ifmap *map);
```

用来实现自定义的 ioctl 命令，如果不需要可以为 NULL。

```
1. void (*set_multicast_list)(struct net_device *dev);
```

当设备的组播列表改变或设备标志改变时，该方法被调用。

```
1. int (*set_mac_address)(struct net_device *dev, void *addr);
```

如果接口支持 mac 地址改变，则可以实现该函数。

设备驱动接口层：

net_device 结构体的成员(属性和函数指针)需要被设备驱动功能层的具体数值和函数赋予。对具体的设置 xxx，工程师应该编写设备驱动功能层的函数，这些函数型如 xxx_open(),xxx_stop(),xxx_tx(),xxx_hard_header(),xxx_get_stats(),xxx_tx_timeout()等。

网络设备与媒介层：

网络设备与媒介层直接对应于实际的硬件设备。