嵌入式LINUX网络驱动开发

华清远见
FARSIGHT
嵌入式培训专家

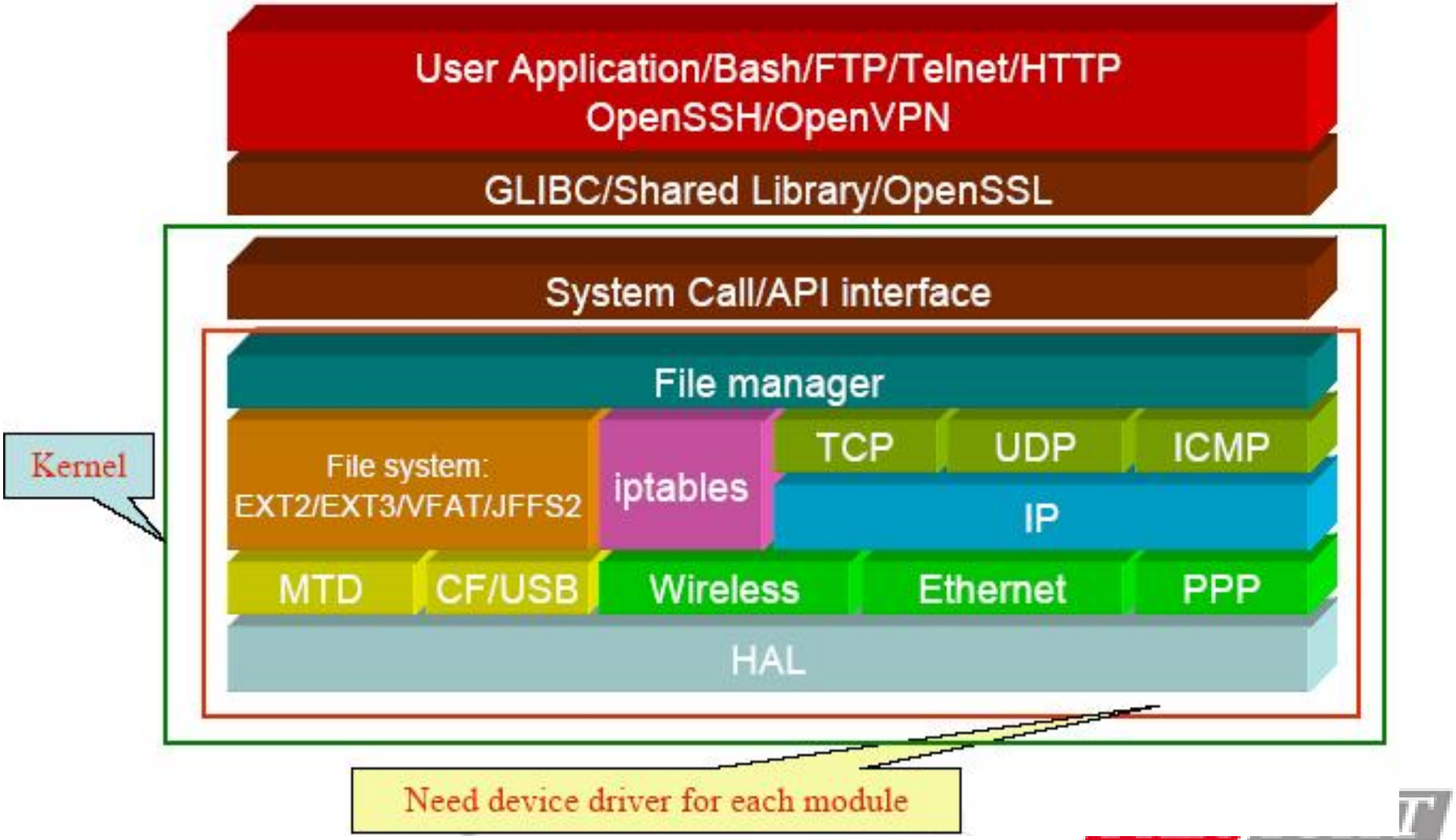
要点

- Linux网络设备驱动程序概述
- 计算机网络概述
- skbuf 数据结构介绍
- Linux网络设备驱动程序API介绍
- Linux网络设备驱动程序实现算法
- 剖析Linux网络设备驱动程序源代码

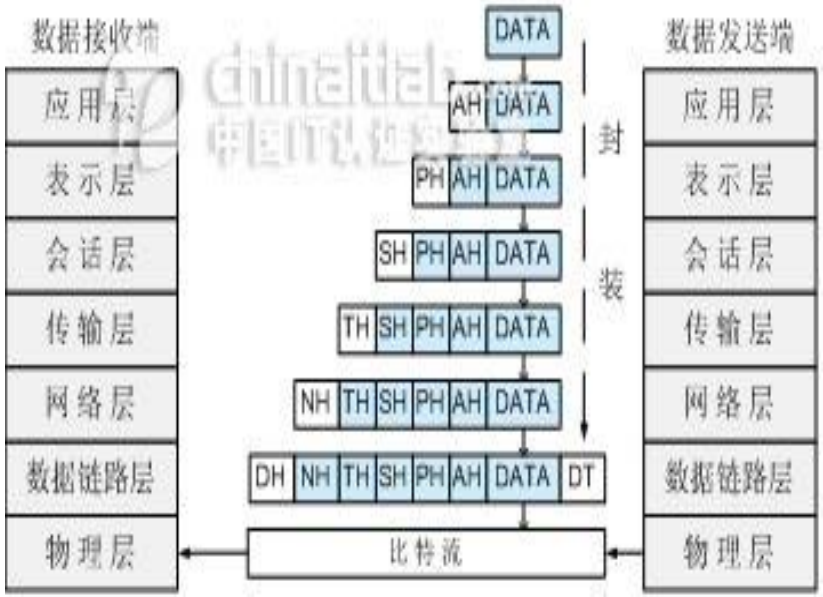
Linux网络设备驱动程序概述

- Linux网络驱动程序遵循通用的接口。设计时采用的是面向对象的方法。一个设备就是一个对象(net_device 结构)，它内部有自己的数据和方法。一个网络设备最基本的方法有初始化,发送和接收。
Linux网络驱动程序的体系结构可以划分为四层：
网络协议接口，网络设备接口，设备驱动功能，网络设备和网络媒介层
网络驱动程序，最主要的工作就是完成设备驱动功能层。在Linux中所有网络设备都抽象为一个接口，这个接口提供了对所有网络设备的操作集合。由数据结构struct net_device来表示网络设备在内核中的运行情况，即网络设备接口。它既包括纯软件网络设备接口，如环路（Loopback），也包括硬件网络设备接口，如以太网卡。而由以dev_base为头指针的设备链表来集体管理所有网络设备，该设备链表中的每个元素代表一个网络设备接口。数据结构net_device中有很多供系统访问和协议层调用的设备方法，包括初始化,打开和关闭网络设备的open和stop函数，处理数据包发送的hard_start_xmit函数，以及中断处理函数等。
网络设备在Linux里做专门的处理。Linux的网络系统主要是基于BSD unix的socket机制。在系统和驱动程序之间定义有专门的数据结构(sk_buff)进行数据的传递。系统里支持对发送数据和接收数据的缓存，提供流量控制机制，提供对多协议的支持。

Linux与网络架构

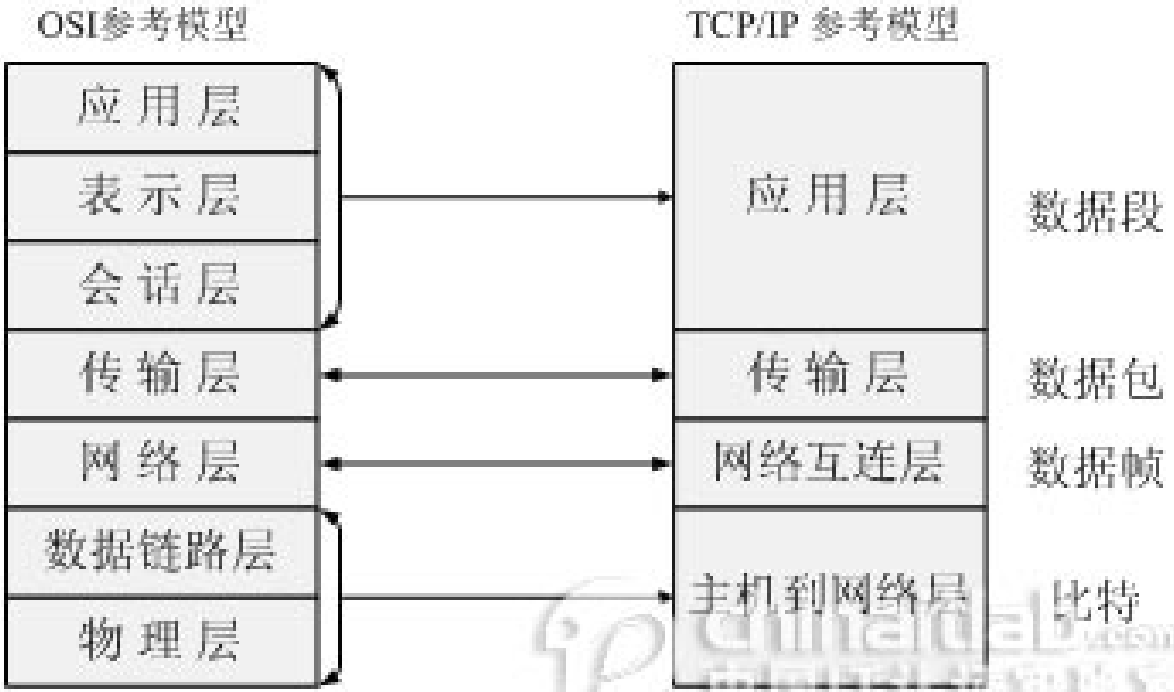


OSI网络参考模型



华清远见
FARSIGHT
嵌入式培训专家

OSI VS TCP/IP



华清远见
FARSIGHT
嵌入式培训专家

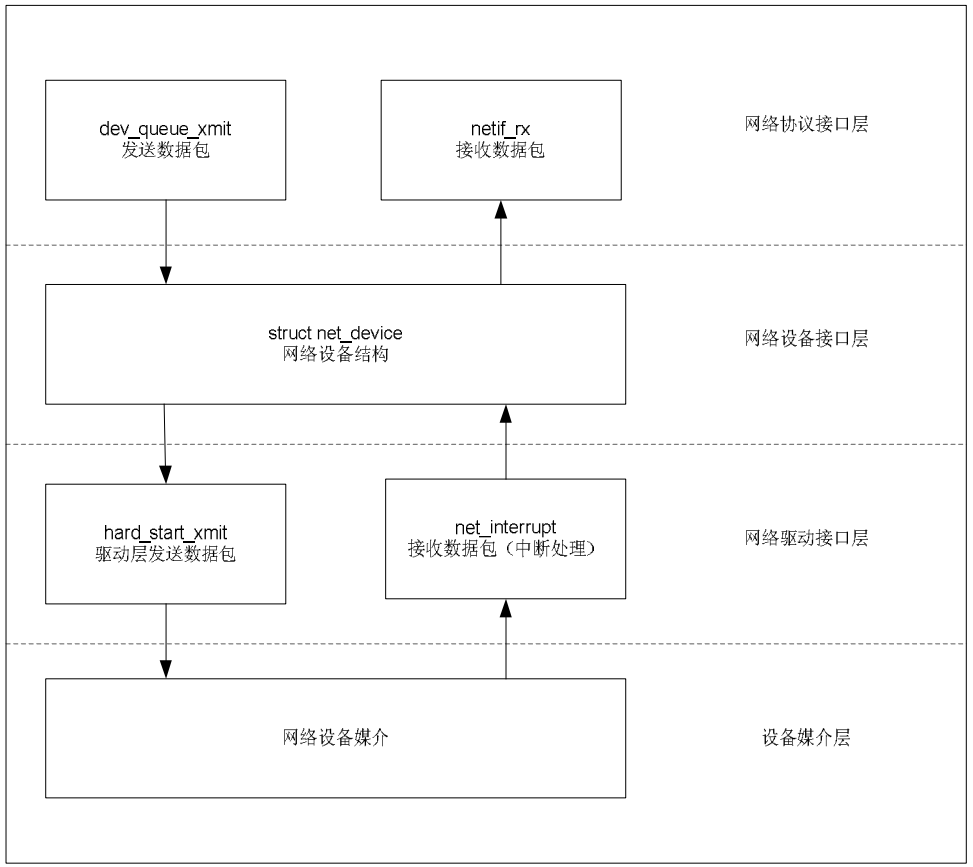
Linux网络设备

- 网络设备，又叫网络接口是Linux第三类标准设备
- 网络设备和块设备类似，在内核的特定数据结构中注册自己
- 当发生网络数据交换时，网络设备驱动程序注册的方法将被内核调用
- 网络设备不会在/dev下存在一个设备入口，它使用保留的内部设备名

网络设备的特点

- 网络设备异步的接收外来的数据包，有别于其他设备
- 网络设备主动的“请求”将硬件获得的数据包压入内核，而其他设备例如块设备被“请求”向内核发送缓冲区
- 网络设备同时要执行大量的管理任务
  - 设置地址
  - 修改传输参数
  - 维护流量和流量控制
  - 错误统计和报告
- 网络子系统是完全与协议无关的，网络驱动程序与内核其余部分之间的每次交互处理的都是一个网络数据包

网络设备驱动体系结构



- 套接字socket简介

华清远见
FARSIGHT
嵌入式培训专家

套接字socket简介

- 客户端与服务器都围绕着通信端点(commnication endpoint)的概念，即套接字。
- 一个套接字通过使用socket()函数惟一确定了一个端点。该端点的细节可以进一步由connect()或bin()函数定义。最终，客户端定义的端点将与服务器定义的端点进行连接和通信。
- 使用UDP和TCP时，端点就是本地或远程IP地址与端口的组合。 Socket广泛用在网络编程中，使用socket有固定的流程。

套接字socket简介（2）

- 服务器方
  - 调用socket
  - 调用bind连接网络地址
  - 启动监听listen
  - 等待连接accept
  - recv，send交换数据
  - close socket
- 客户端方
  - 调用socket
  - 调用connet连接远程主机
  - recv，send交换数据
  - close socket

sk_buff结构分析

华清远见
FARSIGHT
嵌入式培训专家

sk_buff结构

- 套接字缓冲区（sk_buff）结构是Linux内核网络子系统的核心内容，在<linux/skbuff.h>中被定义
- sk_buff结构中的重要字段:

  struct net_device *input_dev;

  – struct net_device *dev;

  分别为接收和发送缓冲区的设备

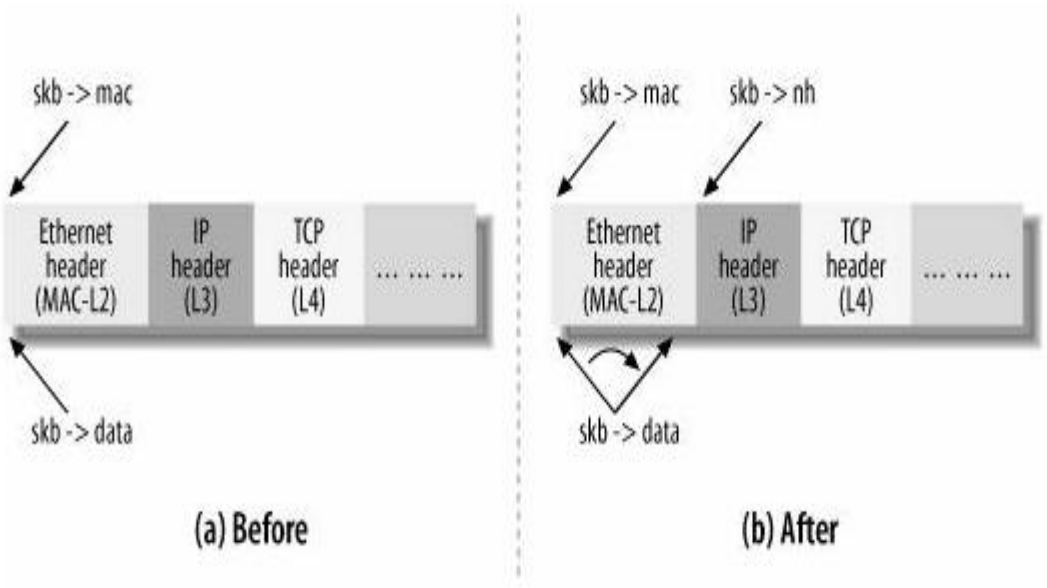
  – union { /* ... */ } h;

  – union { /* . . . */ } nh;

  – union { /*... */} mac;

  指向数据包中各个层的数据包头。h指向传输层包头，nh指向网络层包头，mac指向链路层包头

Figure 2-3. Header's pointer initializations while moving from layer two to layer three



sk_buff 结构（2）

- unsigned char *head;
- unsigned char *data;
- unsigned char *tail;
- unsigned char *end;

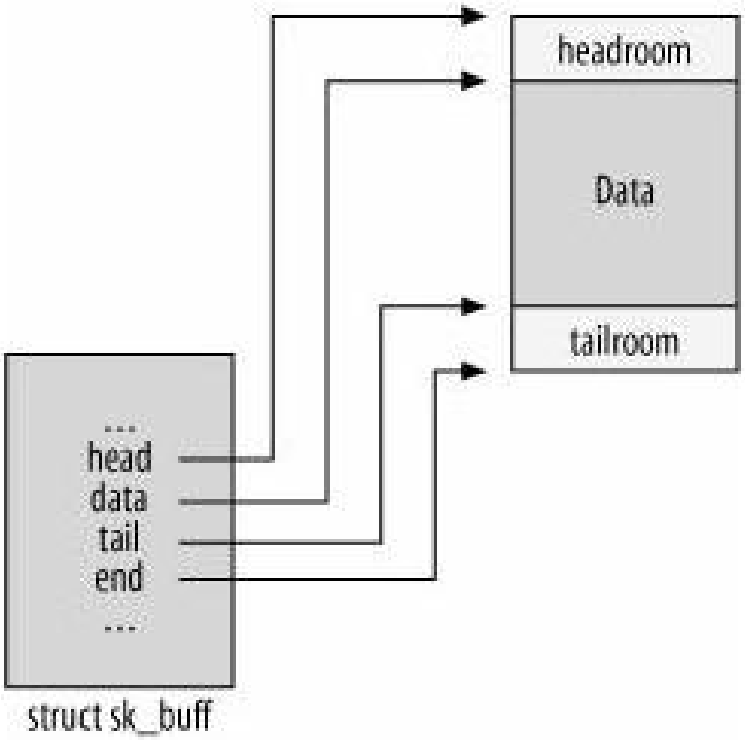
  用来寻址数据包中的数据指针。head指向已分配空间开头，data指向有效的octet开头，tail指向有效的octet结尾，而end指向tail可以到达的最大地址
- unsigned long len;

  描述数据长度
- unsigned char ip_summed;

  描述该数据包的校验和策略
- unsigned char pkt_type;

  描述该数据包的类型，可以是PACKET_HOST，PACKET_BROADCAST， PACKET_MULTICAST，PACKET_OTHERHOST

Figure 2-2. head/end versus data/tail pointers



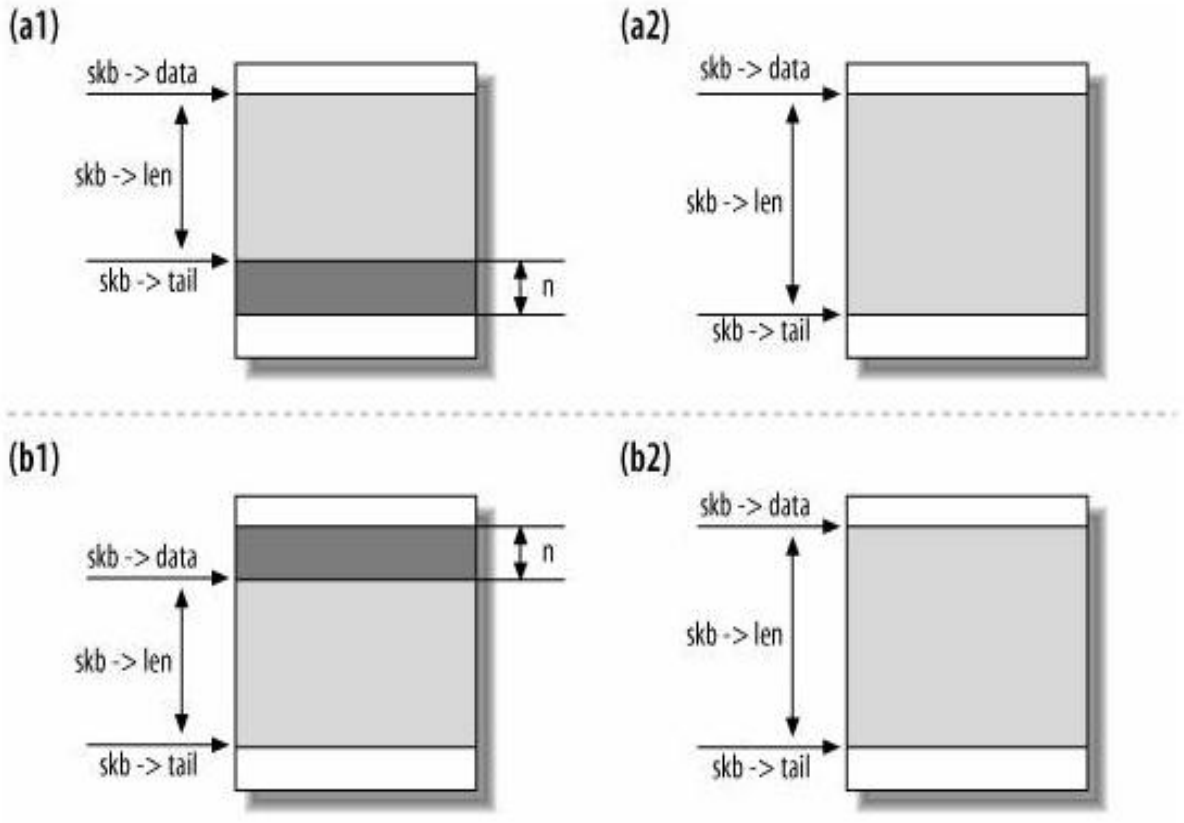
用来操作sk_buff的函数

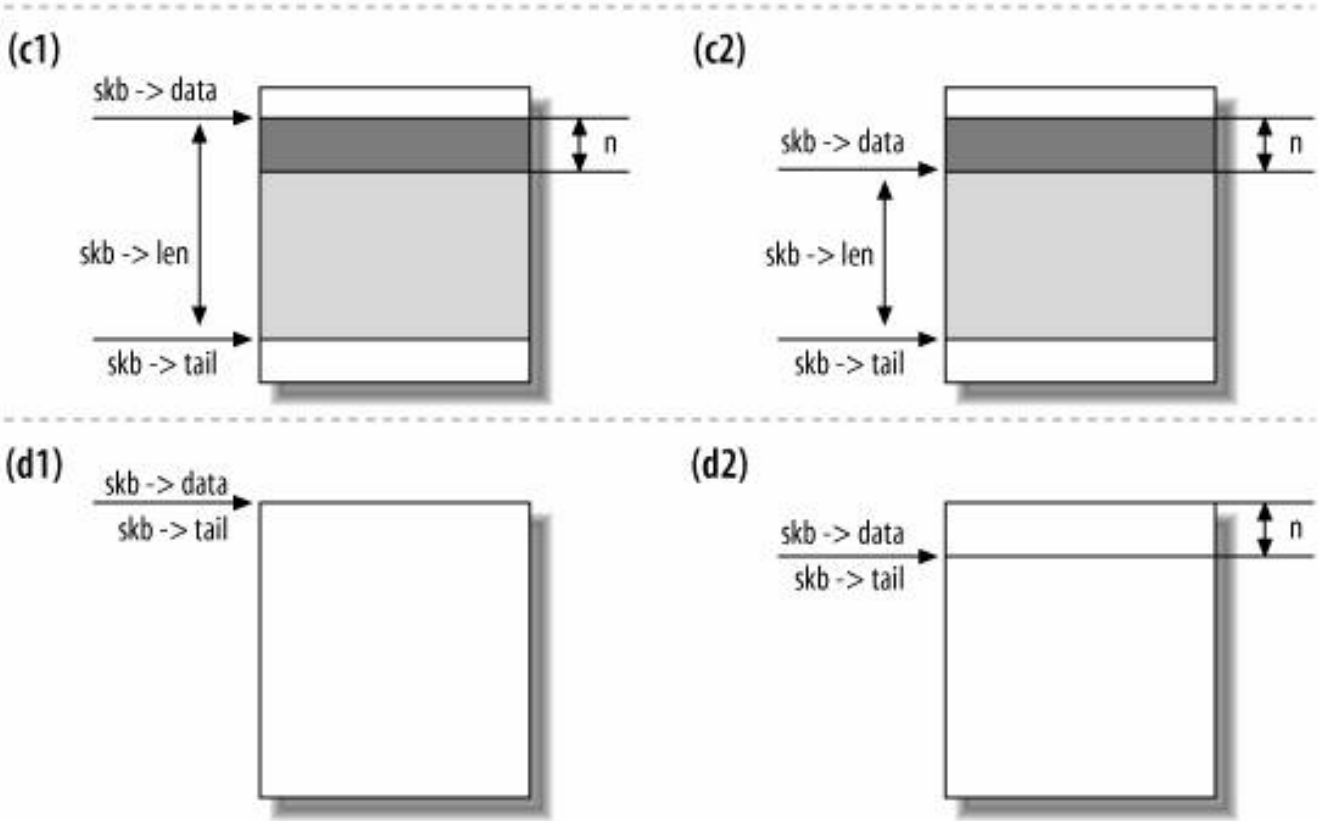
- struct sk_buff *alloc_skb(unsigned int len, int priority);
- struct sk_buff *dev_alloc_skb(unsigned int len);
  用来分配套接字缓冲区
- void kfree_skb(struct sk_buff *skb);
- void dev_kfree_skb(struct sk_buff *skb);
  用来释放套接字缓冲区
- unsigned char *skb_put(struct sk_buff *skb, int len);
- unsigned char *skb_push(struct sk_buff *skb, int len);
  这两个函数用来在缓冲区末尾添加数据，前一个函数会进行检查

用来操作sk_buff的函数（2）

- unsigned char *skb_push(struct sk_buff *skb, int len);
- unsigned char *_ _skb_push(struct sk_buff *skb, int len);
  这两个函数用来在缓冲区头部添加数据
- int skb_tailroom(struct sk_buff *skb);
  该函数返回缓冲区后部可用空间总量
- int skb_headroom(struct sk_buff *skb);
  该函数返回缓冲区前面部分可用空间总量
- void skb_reserve(struct sk_buff *skb, int len);
  该函数在可填充缓冲区之前保留包头空间
- unsigned char *skb_pull(struct sk_buff *skb, int len);
  该函数从数据包头拿出数据，通常用来剥离数据包头

Figure 2-4. Before and after: (a)skb_put, (b)skb_push, (c)skb_pull, and (d)skb_reserve




(c1)
skb -> data
n
skb -> len
skb -> tail
(c2)
skb -> data
n
skb -> len
skb -> tail
(d1)
skb -> data
skb -> tail
(d2)
skb -> data
skb -> tail
n

net_device结构分析

华清远见
FARSIGHT
嵌入式培训专家

net_device结构分析

- 该结构是网络设备驱动的核心，它包含了许多成员。
- 可以参考<include/linux/netdevice.h>文件，阅读它的完整定义。
- net_device结构可分为全局成员、硬件相关成员、接口相关成员、设备方法成员和公用成员等五个部分。
- char name[IFNAMSIZ];
- unsigned long mem_end;

net_device结构分析（2）

- unsigned long mem_start;
  这些字段描述了设备共享内存的起止地址。
- unsigned long base_addr;
  描述设备的I/O基地址
- unsigned char irq;
  描述设备中断号
- unsigned char if_port;
  描述多端口设备的活动端口
- unsigned char dma;
  描述设备的DMA通道

net_device结构分析（3）

- 网络设备要实现一系列函数作为调用方法的实现
- 基本方法包括
  - int (*open)(struct net_device *dev);
    打开接口。当ifconfig激活网络设备时，接口被打开。通常我们在open方法里面完成资源的分配，包括I/O映射、中断注册、DMA注册等。同时激活硬件，并增加使用计数
  - int (*stop)(struct net_device *dev);
    停止接口。在这个方法里面，我们完成与open方法相反的，注销操作
  - int (*hard_start_xmit) (struct sk_buff *skb, struct net_device *dev);
    该方法初始化数据包的传输。是网络设备驱动中非常重要的一个方法。我们将完整的数据包放入一个套接字缓冲区sk_buff结构里

net_device结构分析（4）

- int (*hard_header) (struct sk_buff *skb, struct net_device *dev, unsigned short type, void *daddr, 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
- int (*rebuild_header)(struct sk_buff *skb);
  该方法用来在传输数据包之前重建硬件头
- 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包

net_device结构分析（4-1）

- 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
- 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

net_device结构分析（5）

- 可选方法在以太网设备中通常都可以使用系统提供的方法，也可以自己实现
- int (*do_ioctl)(struct net_device *dev, struct ifreq *ifr, int cmd);
  用来实现设备自定义的ioctl命令。如果不需要自定义命令，可以为NULL
- 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
- 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

net_device结构分析（6）

- int (*change_mtu)(struct net_device *dev, int new_mtu);
  当接口的MTU改变时，该方法将被调用，负责做出相应的特定处理
- int (*header_cache) (struct neighbour *neigh, struct hh_cache *hh);
  根据ARP查询的结果填充hh_cache结构
- int (*header_cache_update) (struct hh_cache *hh, struct net_device *dev, unsigned char *haddr);
  在发生变化时，该方法更新hh_cache结构中的目的地址
- int (*hard_header_parse) (struct sk_buff *skb, unsigned char *haddr);
  从skb中包含的数据包中获得源地址，并将其复制到位于haddr的缓冲区中

打开和关闭网络设备

- 系统响应ifconfig命令时，打开和关闭一个接口
- ifconfig开始会调用ioctl(SIOCSIFADDR)来将地址赋予接口。响应SIOCSIFADDR由内核来完成，与设备无关
- 接着，ifconfig会调用ioctl(SIOCSIFFLAGS)，设置dev->flag的IFF_UP位来打开设备，这个调用会使得设备的open方法得到调用
- 当ifconfig调用ioctl(SIOCSIFFLAGS)，清除dev->flag的IFF_UP位时，设备的stop方法将被调用

数据包接收/发送

华清远见
FARSIGHT
嵌入式培训专家

数据包传送与接收

- 网络接口最重要任务就是发送和接收数据
- 当内核要发送一个数据包时，它会调用*hard_start_transmit*方法来讲数据放入发送队列。
- 内核处理过的每个数据包位于一个套接字缓冲区（struct sk_buff）里面

控制并发传输

- hard_start_xmit函数通过net_device结构中的一个自旋锁（xmit_lock）获得并发调用时的保护
- 实际硬件设备中的缓冲区很有限，驱动程序需要告诉网络子系统在硬件能够接受新数据之前，不能启动其他的数据发送
- 调用netif_stop_queue可以完成这种通知
- 前一个数据传送完成，当设备缓冲区再次可用时，应该调用netif_wake_queue来通知网络子系统重新启动传送队列

传输超时

- 网络传输需要面对可能不可靠的设备、传输介质
- 硬件驱动程序必须能够处理硬件不能正确响应的问题
- 一般来说驱动程序采取超时方式处理这类异常情况，即：
  - 如果某个操作在定时器到期时还未完成，则认为出现了异常问题
  - 网络驱动程序可以在**net_device**结构的**watchdog_timeo**字段中设置超时时间，以**jiffies**为单位
  - 如果当前的系统时间超过设备的**trans_start**时间至少一个超时周期，那么网络层将最终调用驱动程序的**tx_timeout**方法
  - 这个方法完成解决超时问题的工作，并保证正在进行的任何传输能够正常结束

数据包的接收

- 接收数据包首先通过设备中断，由硬件通知驱动程序有数据包到达。
- 网络例子驱动程序实现cs8900_receive函数，该函数在收到数据包后被调用。它获得一个指向数据的指针以及数据包的长度，然后将数据以及附加信息发送到上层的网络代码。

网络驱动的中断处理

华清远见
FARSIGHT
嵌入式培训专家

网络中断处理

- 中断处理程序可以检查物理设备上的状态寄存器，用来区分是数据包到达中断还是传输完成中断
- 传输结束时，中断处理程序更新统计信息，并且调用dev_kfree_skb将不再使用的套接字缓冲区释放给系统
- 同时在传输结束时，如果之前驱动程序停止了传输队列，则调用netif_wake_queue重新启动传输队列
- 而在数据包到达中断发生时，中断处理程序只是调用数据接收函数就够了

网络设备驱动的基本实现

华清远见
FARSIGHT
嵌入式培训专家

网络设备驱动的基本实现

- 一个网络设备最基本的方法有初始化、打开、关闭、发送和接收。
- 初始化程序完成硬件的初始化、网络设备中变量的初始化和系统资源的申请。发送程序是在驱动程序的上层协议层有数据要发送时自动调用的。
- 一般驱动程序中不对发送数据进行缓存，而是直接使用硬件的发送功能把数据发送出去。接收数据一般是通过硬件中断来通知的。在中断处理程序里，把硬件帧信息填入一个sk_buff结构中，然后调用netif_rx()传递给上层处理。

初始化

- 设备探测工作在init方法中进行，一般调用一个称之为probe方法的函数
- 初始化的主要工作是检测设备，配置和初始化硬件，最后向系统申请这些资源。此外，填充该设备的dev结构，我们可以调用内核提供的ether_setup方法来设置一些以太网默认的设置
- 当我们需要卸载网络驱动程序时，我们应该释放初始化时分配的资源，最后调用unregister_netdev来讲自己从全局网络设备链表中注销

打开（open）

- open这个方法在网络设备驱动程序里是网络设备被激活的时候被调用(即设备状态由down-->up)。
- 实际上很多在初始化中的工作可以放到这里来做。比如资源的申请，硬件的激活。如果dev->open返回非0(error)，则硬件的状态还是down。
- 网络驱动程序例子中由cs8900_start 函数实现open方法。

关闭（stop）

- stop方法做和open相反的工作。
- 可以释放某些资源以减少系统负担。
- stop是在设备状态由up转为down时被调用的。
- 网络驱动程序例子中由cs8900_stop函数实现stop方法。

发送(hard_start_xmit)

- 在系统调用驱动程序的hard_start_xmit时，发送的数据放在一个sk_buff结构中。一般的驱动程序把数据传给硬件发出去。也有一些特殊的设备比如loopback把数据组成一个接收数据再回送给系统，或者dummy设备直接丢弃数据。
- 如果发送成功，hard_start_xmit方法里释放sk_buff，返回0(发送成功)。如果设备暂时无法处理，比如硬件忙，则返回1。网络驱动程序例子中由cs8900_send_start函数实现发送

接收(reception)

- 驱动程序并不存在一个接收方法。有数据收到应该是驱动程序来通知系统的。
- 一般设备收到数据后都会产生一个中断，在中断处理程序中驱动程序申请一块sk_buff(skb)，从硬件读出数据放置到申请好的缓冲区里。
- 接下来填充sk_buff中 的一些信息。网络驱动程序例子中由cs8900_receive函数实现接收功能