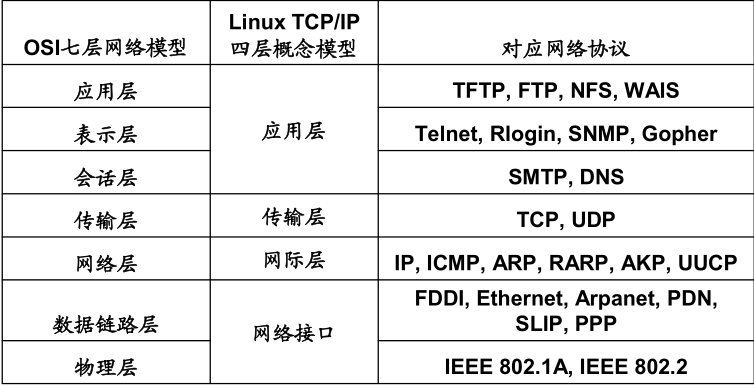
# Linux网卡驱动设计

Linux网络体系结构

Linux的优点之一在于它丰富而稳定的网络协议栈。其范围从协议无关层到各种具体的网络协议实现。

协议层次对比图



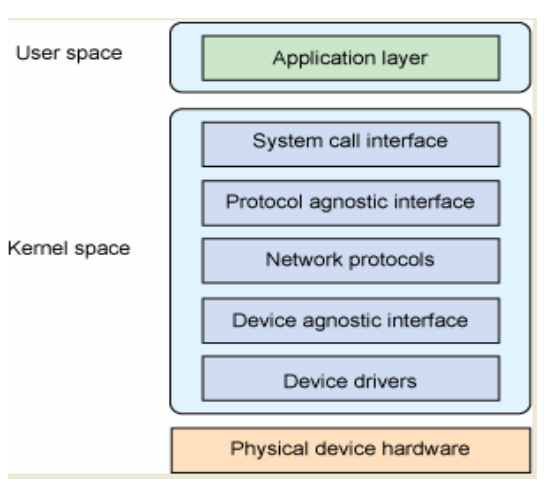
网络接口层提供访问物理设备的驱动程序，对应的网络协议主要是以太网协议。

网络层协议管理离散计算机间的数据传输，如IP协议为用户和远程计算机提供了信息包的传输方法。确保了信息包能正确的到达目的机器。

传输层的功能包括：格式化信息流、提供可靠传输。

应用层位于协议栈的顶端，它的主要任务是服务于应用。

Linux网络子系统



其顶端时系统调用接口层。位于其下面的是一个协议无关层，它提供了一个通用方法来使用传输层协议。然后是具体协议的实现，然后是设备无关层，它提供了协议与设备驱动通信的通用接口，最下面是设备驱动程序。

系统调用接口：Socket系统调用。

协议无关接口：实现一组通用函数来访问各种不同的协议:通过socket实现。这个结构包含了特定socket所需要的所有状态信息，还概括了socket所使用的特定协议和socket上可以执行的一些操作。

设备无关接口将协议与各种网络设备驱动连接在一起。这一层提供一组通用函数供底层网络设备驱动程序使用，让它们可以对高层协议栈进行操作。首先，设备驱动程序可能会通过调用用register\_netdevice或unregister\_netdevice在内核中进行注册或注销。调用者首先填写首先填写net\_device结构结构，然后传递这个结构进行注册。内核调用它的的init函数（如果定义了这种函数），然后执行一组健全性检查，并将新设备添加到设备列表中（内核中的活动设备链表）。

要从协议层向设备发送数据，需要使用dev\_queue\_xmit函函数，这个函数对数据进行排队，并交由底层设备驱动程序进行最终传输报文的接收通常是使用使用netif\_rx执行执行的。当底层设备驱动程序接收到一个报文（包含在所分配的的sk\_buff中）时，就会通过调用用netif\_rx将将数据上传至设备无关层，然后，这个函数通过过netif\_rx\_schedule将sk\_buff在上在上层协议队列中进行排队，供以后进行处理。

驱动设计

设备描述

每个网络接口都由一个net\_device结构来描述，该结构可使用如下内核函数动态分配：

1、struct net\_device \*alloc\_netdev(int sizeof\_priv, const char\*mask, void (\*setup)(struct net\_device \*))

sizeof\_priv 私有数据区大小；mask：设备名；setup 初始化函数

2、struct net\_device \*alloc\_etherdev(int sizeof\_priv)

alloc\_etherdev是alloc\_netdev针对以太网的快捷函数

struct net\_device{

char name[IFNAMSIZ];

unsigned long state;

unsigned long base\_addr;

unsigned int irq;

int (\*init)(struct net\_device \*dev);//该函数在register\_netdev时被调用来完成对net\_device结构的初始化

}

基本方法

int (\*open)(struct net\_device \*dev);打开接口。ifconfig激活时，接口将被打开。

int (\*stop)(struct net\_device \*dev);停止接口。网卡关闭时。

int (\*hard\_start\_xmit) (struct sk\_buff \*skb, struct net\_device \*dev) ;数据发送函数

设备注册

int register\_netdev(struct net\_device \*dev）

sk\_buff

Linux内核中的每个网络数据包都由一个套接字缓冲区结构结构structsk\_buff描述，即一个sk\_buff结构结构就是一个包,指向sk\_buff的的指针通常被称做skb。

该结构包含如下重要成员:

structdevice\*dev;//处处理该包的设备

\_\_u32saddr;//IP源源地址

\_\_u32daddr;//IP目目的地址

\_\_u32raddr;//IP路路由器地址

unsignedchar\*head;//分分配空间的开始

unsignedchar\*data;//有有效数据的开始

unsignedchar\*tail;//有有效数据的结束

unsignedchar\*end;//分分配空间的结束

unsignedlonglen；；//有有效数据的

skb操作函数

struct sk\_buff \*alloc\_skb(unsigned int len, int priority)

分配一个sk\_buff结构，供协议栈代码使用

struct sk\_buff \*dev\_alloc\_skb(unsigned int len)

分配一个sk\_buff结构，供驱动代码使用

unsigned char \*skb\_push(struct sk\_buff \*skb, int len)

向后移动skb的tail指针，并返回tail移动之前的值。

unsigned char \*skb\_put(struct sk\_buff \*skb, int len)

向前移动skb的head指针，并返回head移动之后的值。

kfree\_skb(struct sk\_buff \*skb)

释放一个sk\_buff结构，供协议栈代码使用。

dev\_kfree\_skb(struct sk\_buff \*skb)

释放一个sk\_buff结构，供驱动代码使用

设备打开

1. 注册中断.MDA等
2. 设置寄存器，启动设备
3. 启动发送队列

代码示例

int net\_open(struct net\_device \*dev)

{

/\*申请中断\*/

request\_irq(dev->irq, &net\_interrupt, SA\_SHIRQ,“dm9000”, dev)； ；

/\*设置寄存器,启动设备 \*/

...... ...... ...... ......

/\*启动发送队列\*/

netif\_start\_queue(dev)； ；

}

数据接收

当核心需要发送一个数据包时，他调用hard\_start\_transmit，该函数最终将用到net\_device结构中的hard\_start\_xmit函数指针。

网络接口驱动可以实现两种方式的报文接收：

中断和查询，Linux中驱动多采用中断方式。

接收流程

1. 分配skb

skb = dev\_alloc\_skb(pkt->datalen + 2)

1. 从硬件中读取数据到skb
2. 调用netif\_rx将数据交到协议栈netif\_rx(skb)