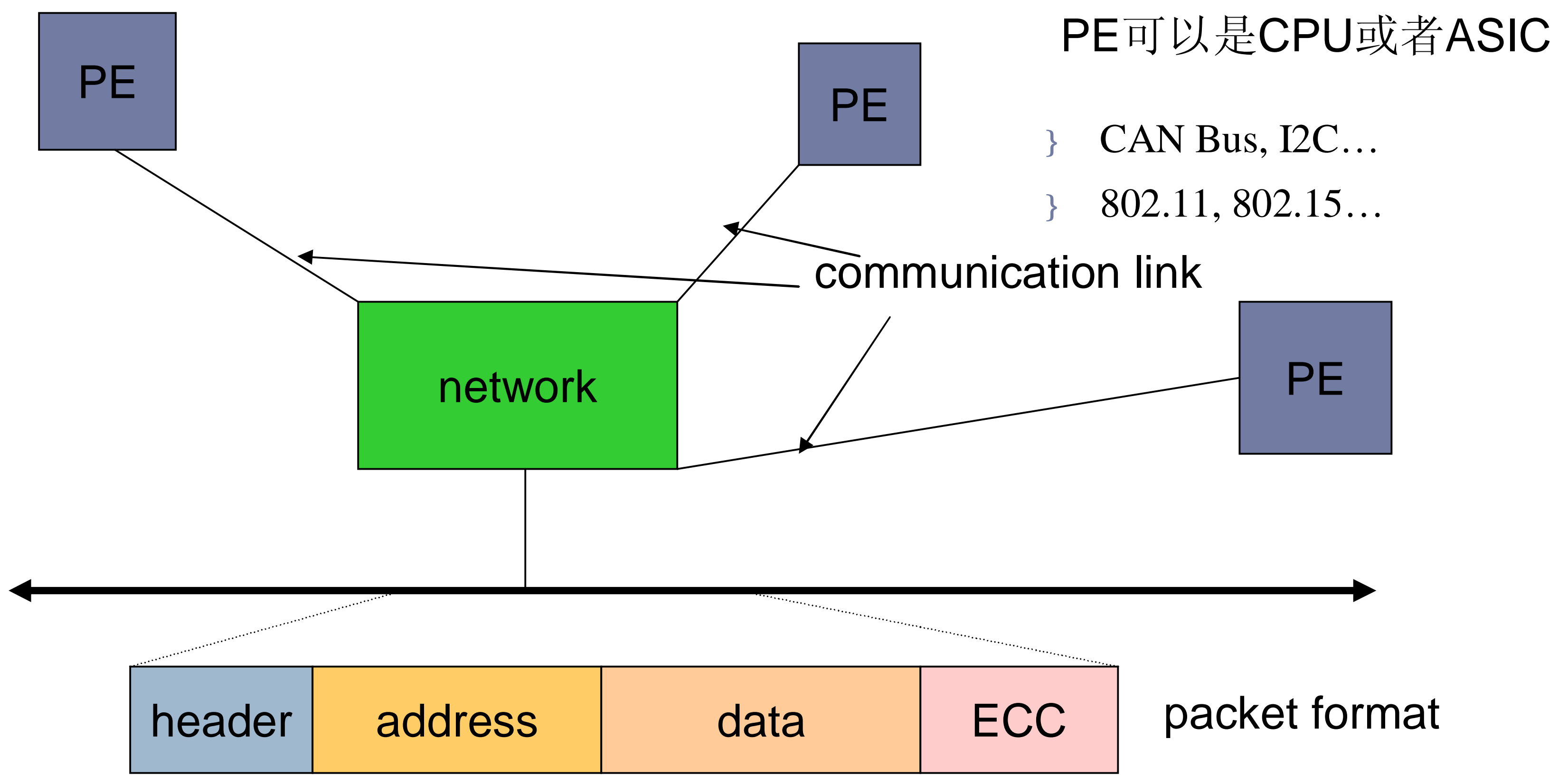
嵌入式LINUX网络驱动开发

2008.10

嵌入式Linux应用

网络化机器人

分布式计算平台



PE可以是CPU或者ASIC

- CAN Bus, I2C…
- 802.11, 802.15…

内容提纲

- Linux网络层级结构
- Socket缓存与网络设备驱动设计
- CGL技术介绍

TCP/IP和OSI模型

TCP/IP	OSI	
应用层	application	end-use interface
	presentation	data format
	session	application dialog control
传输层	transport	connections
网络层	network	end-to-end service
网络接口层	data link	reliable data transport
物理层	physical	mechanical, electrical

Linux支持网络协议

FTP	TFTP	BOOTP/DHCP	CIP	Application
RTP/RTCP	NFS	SMB	More...	
TCP	UDP	SPX	GRE	Transport
RIP	OSPF	DVMRP	802.1Q VLAN	
ARP	RARP	ProxyARP	More...	
IPv4/IPv6	DECnet	AppleTalk	IPX	Network
ICMP	More...			
Ethernet	ATM	FDDI	X.25	Physical
Bluetooth	IRDA	802.11	PPP/CSLIP	
USB	Firewire	More...		

TCP/IP层级结构

ftp
telnet
mail
ping
应用层

Stream Socket
Stream Socket
Stream Socket
Raw Socket
用户模式
内核模式

TCP
UDP
传输层

IP (+ICMP)
网络层

Device Drivers
物理层

TCP/IP层级结构

用户空间

Application Layer (INET)

PF_INET　　PF_INET

内核空间

BSD Socket Layer

Protocol Layer

Network Device Driver Interface/ Queuing Discipline

Physical Device Driver

Physical Device and Media

- 网络接口socket层：为用户提供网络服务编程接口 (linux/net/socket.c)
- 网络协议层：协议实现TCP/IP、IPX、X.25、AppleTalk等协议（linux/net/ipv4）
- 网络接口层：屏蔽具体物理介质，为网络协议提供统一接口，负责下层网络包和协议栈之间交互接送（linux/net/core/dev.c）
- 网络设备接口：负责从物理介质接收和发送数据（linux/driver/net）

网络接口socket层

- socket(AF_INET,SOCK_STREAM,0) 系统调用

  => sys_socket ( )

  => socket_creat ( )

=> inet_creat ( ) {

```
    Answer              = inetsw [SOCK_MAX];   //选择对应的协议
    sock->ops   = answer->ops;   //把协议的操作集赋给socket
    sock->prot  = answer->prot;
}
```

- 特定服务映射到协议族下具体协议，如将SOCK_STREAM映射到TCP协议，将SOCK_DGRAM映射到了UDP协议，SOCK_RAW映射到了IP协议
- accept(),send(),connect()等操作函数通过ops和prot来映射

网络协议层实现

```
struct file_operations socket_file_ops = {
        llseek:     sock_lseek,
        read:       sock_read,
        write:      sock_write,
        poll:       sock_poll,
        ioctl:      sock_ioctl,
        mmap:       sock_mmap,
        open:       sock_no_open,
        release:    sock_close,
        fasync:     sock_fasync,
        readv:      sock_readv,
        writev:     sock_writev
};
```

```
struct proto_ops inet_dgram_ops = {
        family: PF_INET,
        release: inet_release,
        bind: inet_bind,
        connect: inet_dgram_connect,
        accept: sock_no_accept,
        listen: sock_no_listen,
        shutdown: inet_shutdown,
        setsockopt: inet_setsockopt,
        sendmsg: inet_sendmsg,
        recvmsg: inet_recvmsg,
};
```

网络接口层

- 网络接口核心层之上是具体网络协议，之下是网络设备驱动接口
- 网络接口核心层通过dev_queue_xmit()(net/core/dev.c)向上层(IP/ARP协议)提供统一发送接口
- dev_queue_xmit()在dev_base设备列表中找到设备后调用到dev->hard_start_xmit() 调用实际驱动程序来完成发送任务

网络接口核心层和网络协议层接收关系（1/2）

- 网络接口核心层通过函数netif_rx()(net/core/dev.c)接收下层（驱动）发送的数据，然后把数据往上层（协议层）推送

- struct packet_ptype_base[](net/core/dev.c)包含了需要接收数据包的协议及接收函数入口。如packet_type类型包含IP协议ip_rcv()数据接收函数

- 设备驱动中断服务下半部中通过cpu_raise_softirq()标识NET_RX_SOFTIRQ软中断，其处理函数net_rx_action()(net/core/dev.c)根据数据包的协议类型在数组ptype_base[ ] 找到相应协议和接收函数，将数据包交给处理函数，完成数据上推处理。
- 例如数据需要交给IP协议，则ptype_base[ETH_P_IP]->func()(ip_rcv())

数据发送

- sock_write调用了sock_sendmsg
- sock->ops->sendmsg(sock, msg, size, &scm);
- inet_sendmsg()
- sk->prot->sendmsg(sk, msg, size);
- udp_sendmsg
- ip_route_output
- ip_build_xmit生成skb,并在数据包加入IP头
- skb->dst->output(skb);
- ip_output , ip_finish_output
- dev_queue_xmit
- dev->hard_start_xmit(skb, dev);

数据接收

- 当有数据到达网卡时产生一个硬件中断，网卡驱动程序中断服务程序处理（el3_interrupt）
- 通过IO操作将数据数据帧成功读取后，执行接收程序进一步处理（el3_rx(dev) ）
- 数据会被封装成skb转到netif_rx放到等候队列中，由软中断机制net_rx_action通知上层来获取
- ip_packet_type，会在数组ptype_base[16]找到相应，ETH_P_IP，所以在net_rx_action中，会选择IP层的接收处理函数，而从ip_packet_type不难看出，这个函数便是ip_recv( )
- ip_recv调用ip_rcv_finish函数，按接收到的包处理：ip_local_deliver 发往本地进程（推给上层协议）或ip_forward转发（用作网关）

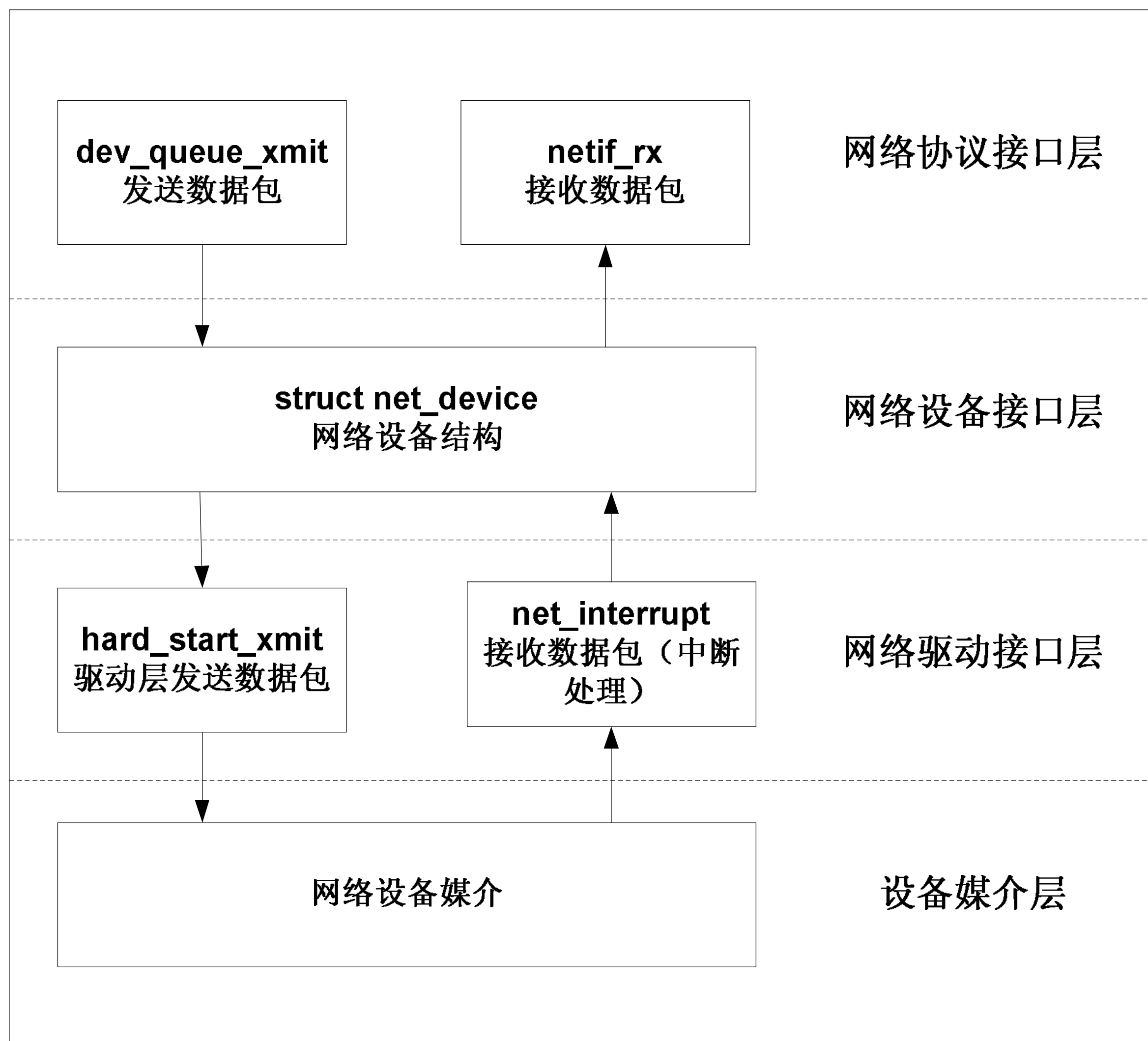
内容提纲

- Linux网络层级结构
- Socket缓存与网络设备驱动设计
- CGL技术介绍

网络设备接口

- 网络设备接口负责物理介质控制，接收及发送数据，并对物理介质进行各种设置，例如以太网卡驱动
  - 发送：el_start_xmit( )函数通过对以太网卡寄存器操作发送数据
  - 接收：接收数据包或数据包发送完成时，都产生中断，进入中断服务程序处理。例如收数据时，为数据包分配skb缓存，通过netif_rx( )把数据包交给上层

网络设备驱动结构



套接字缓冲(sk_buff)结构组成

- 套接字缓冲结构sk_buff是Linux内核网络系统核心内容
  - hard_start_transmit将一个套接字缓冲（sk_buff）中数据放入外发队列
  - 每个数据包属于更高网路层某个套接字，所有套接字输入和输出缓冲区都是sk_buff结构形成链表

- 套接字缓冲结构中重要组成：
  - struct net_device *rx_dev, *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

套接字缓冲(sk_buff)结构组成

- unsigned char *head, *data, *tail, *end;
  - 用来寻址数据包中数据指针。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

以太网帧结构

- 以太网(IEEE 802.3)协议其驱动相关的帧结构：

前导位PR	帧起始位SD	目的IP地址SA	源IP地址DA	类型TYPE/长度LEN	数据域DATA	填充PAD	校验FCS
62bit	2bit	48bit	48bit	16bit	<=1500字节	可选	32bit

- 网卡驱动也不必关心所有内容，如其中前导序列、帧起始位、CRC 校验由硬件自动添加/删除，与驱动软件无关

net_device结构与网络设备初始化

- net_device结构复杂，主要字段：
  - char name[IFNAMSIZ]；
  - unsigned long rmem_end;
  - unsigned long rmem_start;
  - unsigned long mem_end;
  - 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; //描述了设备的I/O基地址
  - unsigned char irq; //描述了设备中断号
  - unsigned char if_port; //描述了多端口设备的活动端口
  - unsigned char dma; //描述了设备的DMA通道

- 网络设备初始化
  - 在init中调用probe函数，进行设备探测与初始化。检测设备、配置和初始化硬件，向系统申请资源， ether_setup方法来填充设备net_device结构，进行全局网络设备链表注册

网络设备调用方法实现

- 网络设备要实现下列函数调用方法
  - int (*open)(struct net_device *dev); 打开接口， 当ifconfig激活网络设备时接口被打开。进行资源分配，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，完成与open方法相反的操作
  - int (*hard_start_xmit) (struct sk_buff *skb, struct net_device *dev); 该方法初始化数据包传输，将完整数据包放入一个套接字缓冲区sk_buff结构

华清远见 FARSIGHT 嵌入式培训专家

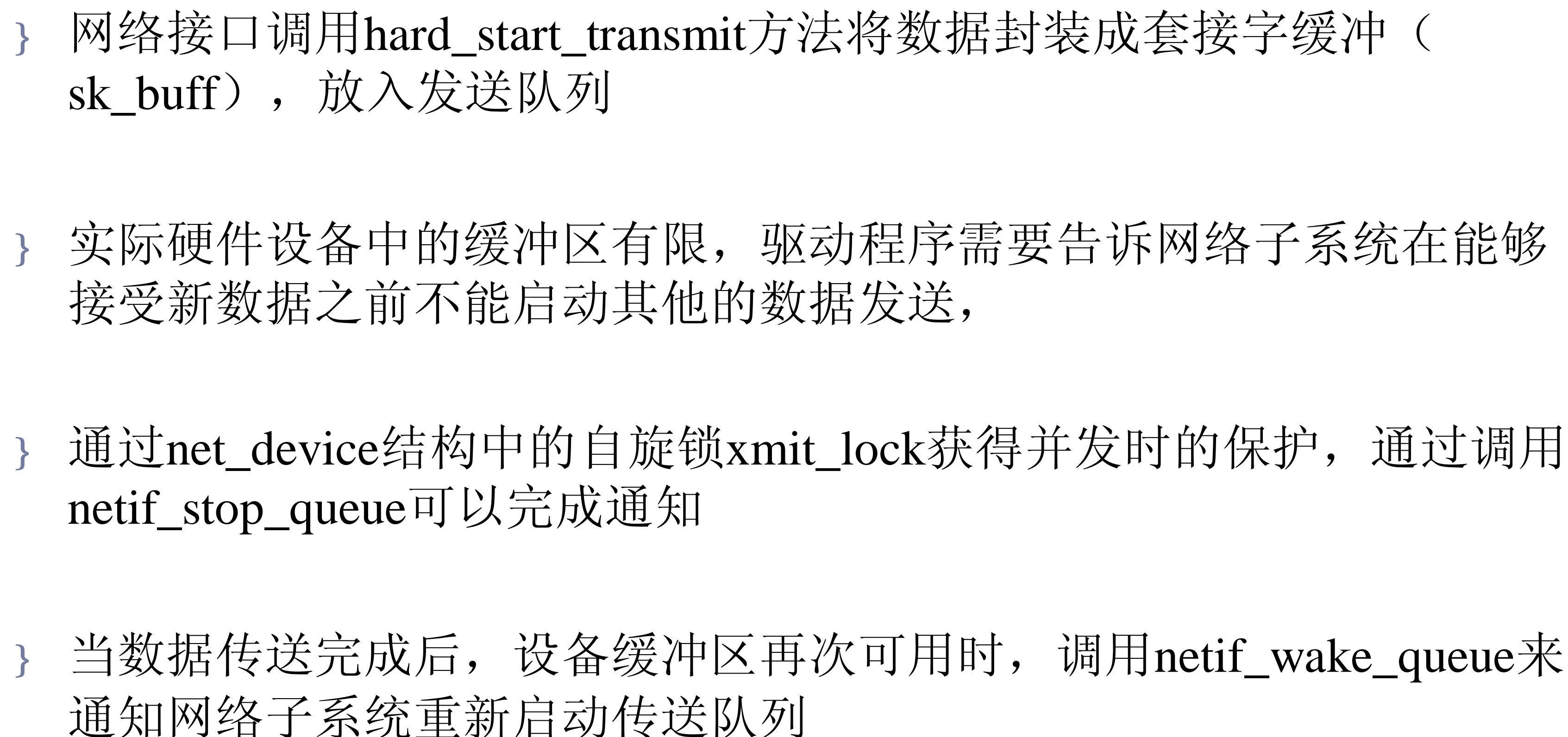
网络设备调用方法实现

- 网络设备要实现下列函数调用方法
  - int (*hard_header) (struct sk_buff *skb, struct net_device *dev, unsigned short type, void *daddr, void *saddr, unsigned len); //根据源和目的硬件地址建立硬件头
  - int (*rebuild_header)(struct sk_buff *skb); //用来在传输数据包前重建硬件头
  - void (*tx_timeout)(struct net_device *dev); //如数据包在超时时间内发送失败，该方法被调用，用于解决传送失败问题并重传
  - struct net_device_stats *(*get_stats)(struct net_device *dev); // 当应用程序需要获得接口统计信息时，该方法被调用
  - int (*set_config)(struct net_device *dev, struct ifmap *map); //改变接口的配置，例如改变I/O端口和中断号等

网络设备调用方法（可选）

- 通常都可使用系统提供的方法或自己实现
- 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，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
- int (*change_mtu)(struct net_device *dev, int new_mtu); //当接口MTU改变时，该方法被调用，负责做出相应处理
- int (*header_cache) (struct neighbour *neigh, struct hh_cache *hh); //根据ARP查询结果填充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中包含数据包中获得源地址，并将其复制到位于haddr的缓冲区中

数据包传送与接收



- 网络接口调用hard_start_transmit方法将数据封装成套接字缓冲（sk_buff），放入发送队列
- 实际硬件设备中的缓冲区有限，驱动程序需要告诉网络子系统在能够接受新数据之前不能启动其他的数据发送，
- 通过net_device结构中的自旋锁xmit_lock获得并发时的保护，通过调用netif_stop_queue可以完成通知
- 当数据传送完成后，设备缓冲区再次可用时，调用netif_wake_queue来通知网络子系统重新启动传送队列

華清遠見 FARSIGHT 嵌入式培训专家

传输超时

- 驱动程序必须能处理硬件不能及时响应的情况
  - 驱动程序采取定时器到期等超时方式处理这类异常
  - 在net_device结构中的watchdog_timeo字段中设置超时时间(jiffies)
  - 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，网络层将最终调用驱动程序tx_timeout方法完成超时问题解决

数据包接收与中断处理

- 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
- 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

- 中断处理程序检查状态寄存器，区分数据包到达和传输完成中断
- 传输完成则中断处理程序更新统计信息，调用dev_kfree_skb将套接字缓冲区释放。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
- 数据包到达时，中断处理程序只是调用数据接收函数

华清远见 FARSIGHT 嵌入式培训专家

定制ioctl命令

- 套接字命令号在<linux/sockios.h>里定义，函数sock_ioctl直接调用协议相关函数
- 协议层不能识别的ioctl命令会被传递到设备层，设备相关的ioctl命令从用户空间接受其他参数
- 除了使用标准化调用之外，每个接口还可以定义16个接口私有的ioctl命令：从SIOCDEVPRIVATE到SIOCDEVPRIVATE+15，调用相关接口驱动程序dev->do_ioctl方法

Cs8900a驱动设计

FARSIGHT 华清远见 嵌入式培训专家

- 驱动移植
  - 中断的挂接：在系统中根据实际中断线申请相应中断号
  - IO地址正确设置与访问：
    - 例如网络片选为CS2，地址空间是0X20000000~0X2FFFFFFF
    - #define S3C2410_PHYS_CS8900A CS2_PHYS_BASE /* physical */
    - #define S3C2410_VIRT_CS8900A (0xfc000000)   /* virtual */
  - 手动设置接口的属性：如果没有EEPROM则需要在代码中手动设置接口的属性，如以太芯片MAC地址等

内容提纲

- Linux网络层级结构
- Socket缓存与网络设备驱动设计
- CGL技术介绍

内核支持

- Real-time kernel
- Configurable OOM killer
- posix_fadvise(2)
- Kexec for fast reboot
- LSM
- NBD
- DRBD
- Evlog
- Rmon
- corefile naming

- IMQ
- Fumount
- NAPI support for drivers
- Open file by inode
- Panic handler enhancements
- Asynchronous events (Libevent)
- Machine Check Architecture (MCA)
- RAID disk mirroring
- RAID multihost

连接和I/O

- TCP/IPv4
- IPv6 Certification Ready
- IPSec
- IKE
- VLAN Tagging (IEEE 802.1Q)
- SCTP
- TCP Abort
- Hotplug (ATCA, uTCA, AMC)
- Serial ATA
- USB2 Host
- USB Human Interface Device (HID) Input Core
- MTD
- Ifenslave
- IPMI
- Logical Volume Management/Logical Volume Management2 (LVM/LVM2)
- Fibrechannel
- Routable ARP
- AIC

实时响应

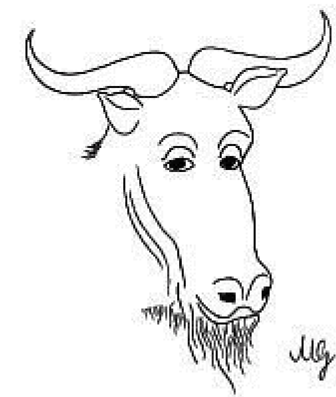
- Real time preemptible kernel technology
  (100% native Linux; no double-kernel non-Linux add-ons)
- High resolution POSIX timers
- Threaded soft and hard IRQ handlers
- Application-level priority inheritance
- Priority queuing
- Robust mutexes
- Futexes Userland support
- Preempt_RT

handhelds.org

inuxDevices.com

KernelTrap.org

- Kernel
  - http://kernel.org
- Linux Kernel Discussion
  - http://www.kerneltraffic.org
  - http://www.kerneltrap.org
- General Linux
  - http://lwn.net
  - http://slashdot.org
  - http://freshmeat.net
  - http://www.linux-tutorial.info
- Embedded Linux
  - http://www.linuxdevices.com



- GPL = General Public License
  - http://www.gnu.org/copyleft/copyleft.html
  - http://www.gnu.org/licenses/gplfaq.html
- ELC
  - http://www.embedded-linux.org
  - http://www.osdl.org
  - http://www.celf.org

谢谢！

huangxin@farsight.com.cn