版权

- 华清远见嵌入式培训中心版权所有；
- 未经华清远见明确许可，不能为任何目的以任何形式复制或传播此文档的任何部分；
- 本文档包含的信息如有更改，恕不另行通知；
- 保留所有权利。

内容提纲

- 1、网络设备特点
- 2、工作原理
- 3、网络设备对象类net_device
  - 3.1、接口名字
  - 3.2、硬件信息
  - 3.3、接口信息
  - 3.4、接口标志
  - 3.5、操作方法集
  - 3.6、私有数据
- 4、内核函数
- 5、套接字缓冲区

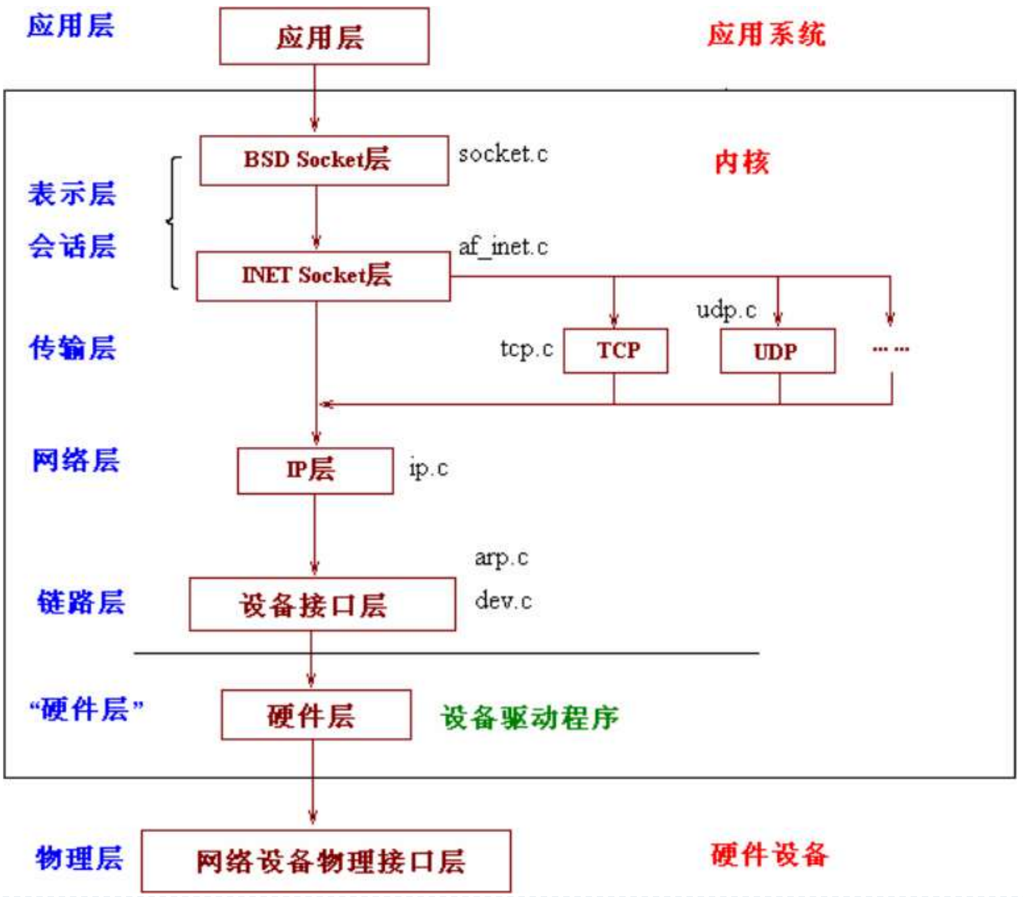
网络设备特点

- 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。在linux下一切皆是文件，是标准化接口的一种抽象手段。由于历史原因，网络编程的套接字接口标准早于linux内核出现，linux内核不得不沿用套接字的概念，从而三种接口驱动里面，仅网络设备没有设备文件。

网络设备特点

- 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

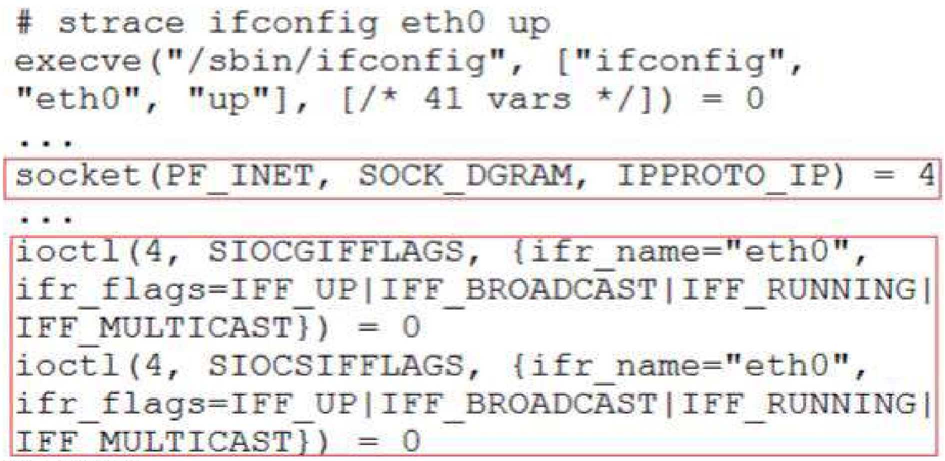
工作原理



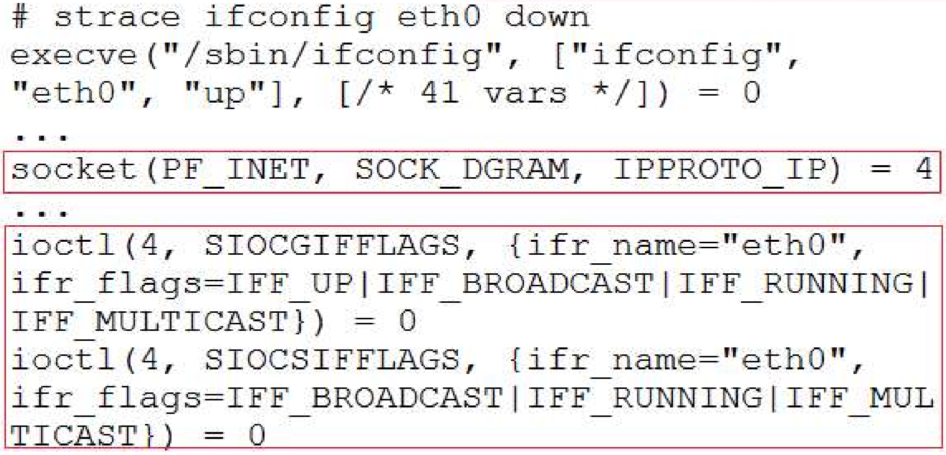
应用层
应用层
应用系统
BSD Socket层
socket.c
内核
表示层
会话层
INET Socket层
af_inet.c
udp.c
传输层
tcp.c
TCP
UDP
… …
网络层
IP层
ip.c
arp.c
链路层
设备接口层
dev.c
"硬件层"
硬件层
设备驱动程序
物理层
网络设备物理接口层
硬件设备

工作原理

- 使能网卡
- #ifconfig eth0 up
- 关闭网卡
- #ifconfig eth0 down
- 查看网卡信息
- #ifconfig



```
# strace ifconfig eth0 up
execve("/sbin/ifconfig", ["ifconfig",
"eth0", "up"], [/* 41 vars */]) = 0
...
socket(PF_INET, SOCK_DGRAM, IPPROTO_IP) = 4
...
ioctl(4, SIOCGIFFLAGS, {ifr_name="eth0",
ifr_flags=IFF_UP|IFF_BROADCAST|IFF_RUNNING|
IFF_MULTICAST}) = 0
ioctl(4, SIOCSIFFLAGS, {ifr_name="eth0",
ifr_flags=IFF_UP|IFF_BROADCAST|IFF_RUNNING|
IFF_MULTICAST}) = 0
```



```
# strace ifconfig eth0 down
execve("/sbin/ifconfig", ["ifconfig",
"eth0", "up"], [/* 41 vars */]) = 0
...
socket(PF_INET, SOCK_DGRAM, IPPROTO_IP) = 4
...
ioctl(4, SIOCGIFFLAGS, {ifr_name="eth0",
ifr_flags=IFF_UP|IFF_BROADCAST|IFF_RUNNING|
IFF_MULTICAST}) = 0
ioctl(4, SIOCSIFFLAGS, {ifr_name="eth0",
ifr_flags=IFF_BROADCAST|IFF_RUNNING|IFF_MUL
TICAST}) = 0
```

工作原理

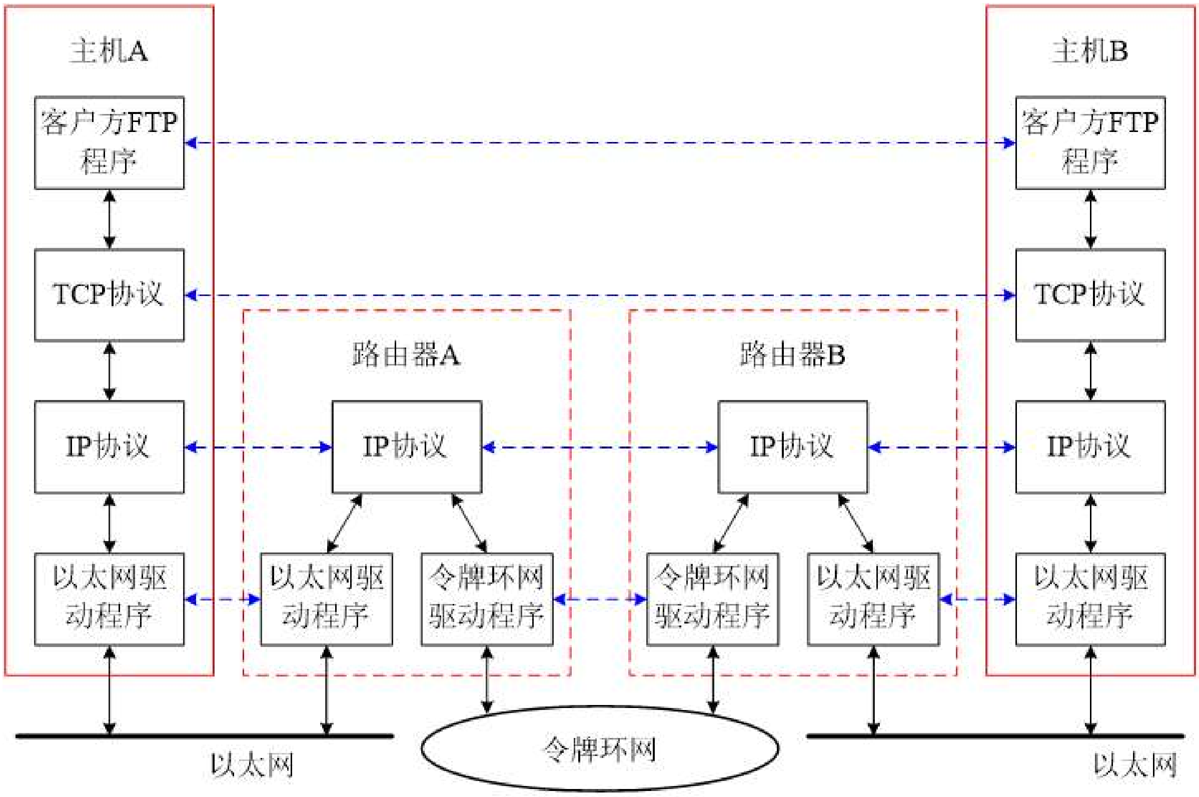
- 接口请求结构体(linux/if.h)

```
struct ifreq {
#define IFHWADDRLEN       6
        union{
                    char        ifrn_name[IFNAMSIZ]; /* if name, e.g. "en0" */
        } ifr_ifrn;
        union {
                    struct      sockaddr ifru_addr;
                    struct      sockaddr ifru_dstaddr;
                    struct      sockaddr ifru_broadaddr;
                    struct      sockaddr ifru_netmask;
                    struct  sockaddr ifru_hwaddr;
                    short       ifru_flags;
                    int         ifru_mtu;
                    ……
        } ifr_ifru;
};
```

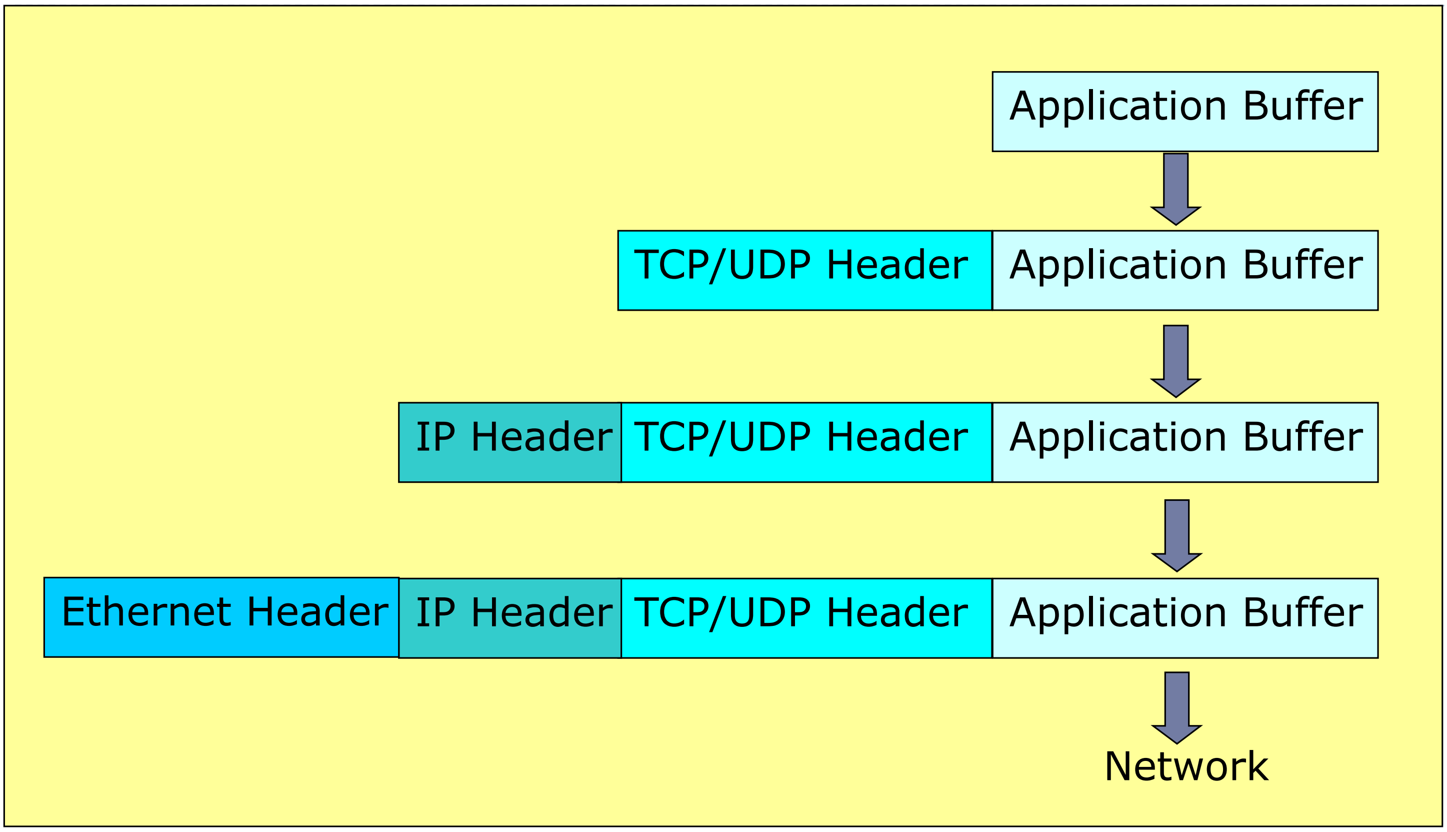
工作原理

- 结构域宏
  - #define ifr_name ifr_ifrn.ifrn_name /* interface name */
  - #define ifr_hwaddr ifr_ifru.ifru_hwaddr /* MAC address */
  - #define ifr_addr ifr_ifru.ifru_addr /* address */
  - #define ifr_dstaddr ifr_ifru.ifru_dstaddr /* other end of p-p lnk*/
  - #define ifr_broadaddr ifr_ifru.ifru_broadaddr /* broadcast address */
  - #define ifr_netmask ifr_ifru.ifru_netmask /* interface net mask */
  - #define ifr_flags ifr_ifru.ifru_flags /* flags */
  - …
- 命令码
  - #define SIOCGIFNAME 0x8910 /* get iface name */
  - #define SIOCSIFLINK 0x8911 /* set iface channel */
  - #define SIOCGIFCONF 0x8912 /* get iface list */
  - #define SIOCGIFFLAGS 0x8913 /* get flags */
  - #define SIOCSIFFLAGS 0x8914 /* set flags */
  - #define SIOCGIFADDR 0x8915 /* get PA address */
  - …

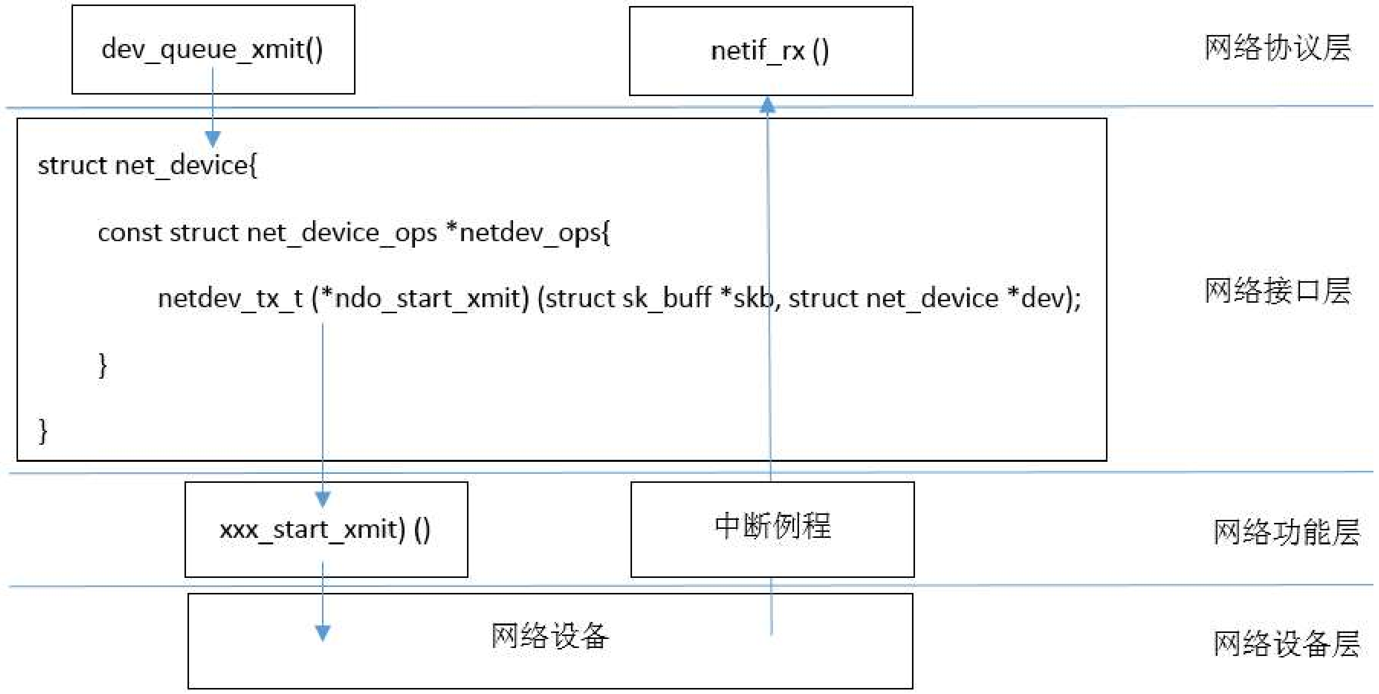
工作原理


主机A
客户方FTP程序
TCP协议
IP协议
以太网驱动程序
路由器A
IP协议
以太网驱动程序
令牌环网驱动程序
路由器B
IP协议
令牌环网驱动程序
以太网驱动程序
主机B
客户方FTP程序
TCP协议
IP协议
以太网驱动程序
以太网
令牌环网
以太网

工作原理


Application Buffer
TCP/UDP Header
Application Buffer
IP Header
TCP/UDP Header
Application Buffer
Ethernet Header
IP Header
TCP/UDP Header
Application Buffer
Network

工作原理



dev_queue_xmit()

netif_rx ()

网络协议层

```
struct net_device{
        const struct net_device_ops *netdev_ops{
                netdev_tx_t (*ndo_start_xmit) (struct sk_buff *skb, struct net_device *dev);
        }
}
```

网络接口层

xxx_start_xmit) ()

中断例程

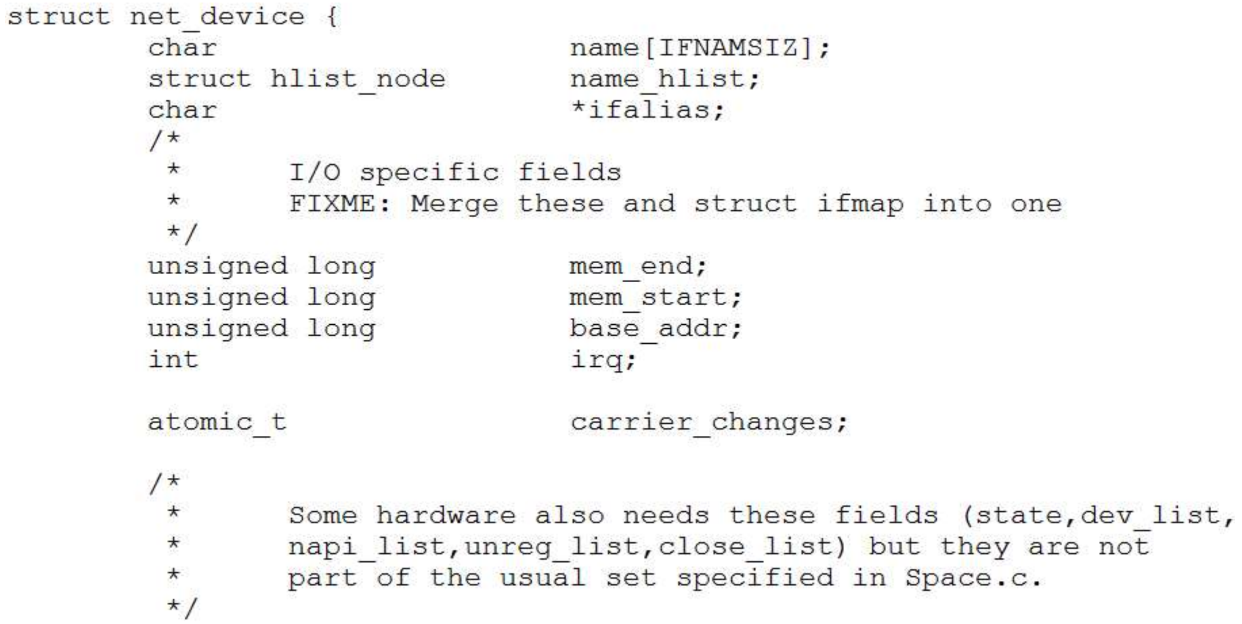
网络功能层

网络设备

网络设备层

网络设备对象类

- 头文件:linux/netdevice.h
- net_device结构体在内核中抽象一个网络设备，网络设备驱动程序只需通过填充net_device的具体成员并注册net_device即可实现硬件操作函数与内核的挂接。
- net_device本身是一个巨型结构体，包含网络设备的属性描述和操作接口。当我们编写网络设备驱动程序时，只需要了解其中的一部分。



```
struct net_device {
        char                    name[IFNAMSIZ];
        struct hlist_node       name_hlist;
        char                    *ifalias;
        /*
         *      I/O specific fields
         *      FIXME: Merge these and struct ifmap into one
         */
        unsigned long           mem_end;
        unsigned long           mem_start;
        unsigned long           base_addr;
        int                     irq;

        atomic_t                carrier_changes;

        /*
         *      Some hardware also needs these fields (state,dev_list,
         *      napi_list,unreg_list,close_list) but they are not
         *      part of the usual set specified in Space.c.
         */
```

网络设备对象类

- 物理接口名
- struct net_device {
- 	char			name[IFNAMSIZ];
- 	struct hlist_node	name_hlist;
- 	char			*ifalias;
- 	…
- };
- name域：
  - 设备接口名，该名字包含一个 %d 格式串，在驱动对象注册时候 用一个数替换它以形成一个唯一的名子，分配的编号从 0 开始。
- name_hlist域：
  - 设备接口名的HASH表，内核建议不要用
- ifalias域：
  - SNMP别名

网络设备对象类

- 硬件信息
- struct net_device {
-     ...
-     unsigned long   mem_end;
-     unsigned long   mem_start;
-     unsigned long   base_addr;
-     int   irq;
-     ...
- };
- mem_end/mem_start域：
  - 共享内存起始首地址
- base_addr域：
  - 基地址
- irq域：
  - 中断号

网络设备对象类

- 接口信息

```
struct net_device {
        ...
        netdev_features_t     features;              //当前活动的设备特征
        netdev_features_t     hw_features;           //用户可修改的设备特征
        struct net_device_stats stats;               //统计信息
        unsigned char          if_port;              //端口选择
        unsigned char          dma;                  //DMA通道
        unsigned int           mtu;                  //mtu值
        unsigned short         type;                 //硬件类型
        unsigned short         hard_header_len;      //硬件头长
        struct in_device __rcu*ip_ptr;               //本地ip（IPv4）
        struct inet6_dev __rcu*ip6_ptr;              //本地ip（IPv6）
        unsigned char          *dev_addr;            //mac地址
        unsigned char          broadcast[MAX_ADDR_LEN]; //广播地址
        ...
};
```

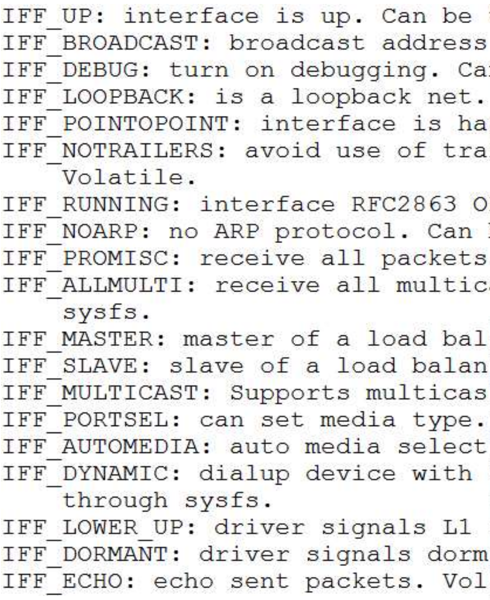
网络设备对象类

```
IFF_802_1Q_VLAN: 802.1Q VLAN device
IFF_EBRIDGE: Ethernet bridging device
IFF_SLAVE_INACTIVE: bonding slave not the curr. active
IFF_MASTER_8023AD: bonding master, 802.3ad
IFF_MASTER_ALB: bonding master, balance-alb
IFF_BONDING: bonding master or slave
IFF_SLAVE_NEEDARP: need ARPs for validation
IFF_ISATAP: ISATAP interface (RFC4214)
IFF_MASTER_ARPMON: bonding master, ARP mon in use
IFF_WAN_HDLC: WAN HDLC device
IFF_XMIT_DST_RELEASE: dev_hard_start_xmit() is allowed to
    release skb->dst
IFF_DONT_BRIDGE: disallow bridging this ether dev
IFF_DISABLE_NETPOLL: disable netpoll at run-time
IFF_MACVLAN_PORT: device used as macvlan port
IFF_BRIDGE_PORT: device used as bridge port
IFF_OVS_DATAPATH: device used as Open vSwitch datapath port
IFF_TX_SKB_SHARING: The interface supports sharing skbs on transmit
IFF_UNICAST_FLT: Supports unicast filtering
IFF_TEAM_PORT: device used as team port
IFF_SUPP_NOFCS: device supports sending custom FCS
IFF_LIVE_ADDR_CHANGE: device supports hardware address
    change when it's running
IFF_MACVLAN: Macvlan device
```

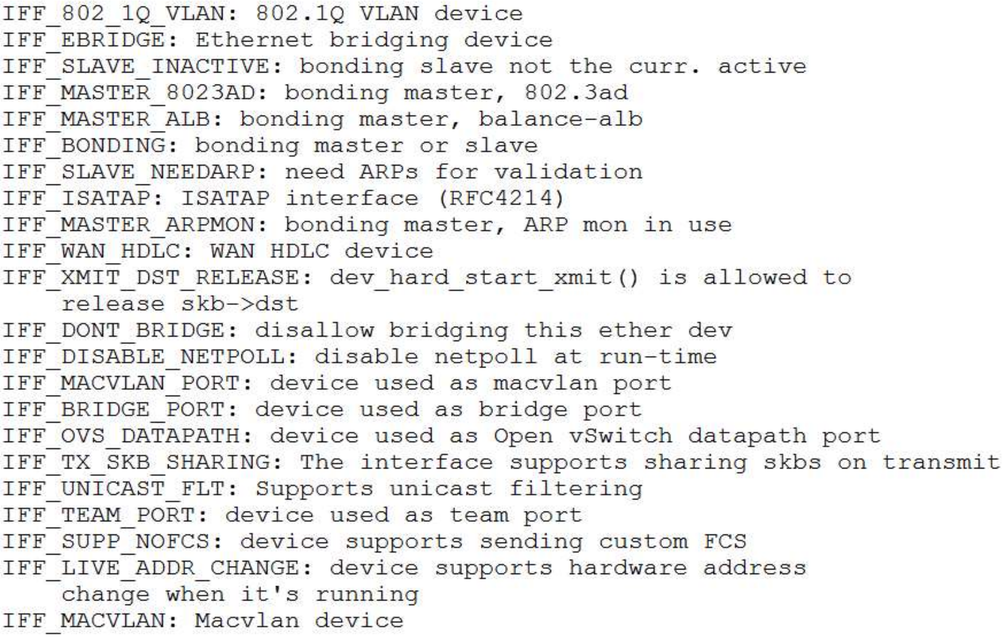
网络设备对象类

- 接口标志
- struct net_device {
-     ...
-     unsigned int flags;
-     unsigned int priv_flags;
-     ...
- };
- flags域：
  - flags标志，可以使能网卡、允许广播、允许组播、允许混杂模式，等等
- priv_flags域：
  - 跟flags一样，仅内核可见，用户空间不可见

网络设备对象类



```
IFF_UP: interface is up. Can be toggled through sysfs.
IFF_BROADCAST: broadcast address valid. Volatile.
IFF_DEBUG: turn on debugging. Can be toggled through sysfs.
IFF_LOOPBACK: is a loopback net. Volatile.
IFF_POINTOPOINT: interface is has p-p link. Volatile.
IFF_NOTRAILERS: avoid use of trailers. Can be toggled through sysfs.
    Volatile.
IFF_RUNNING: interface RFC2863 OPER_UP. Volatile.
IFF_NOARP: no ARP protocol. Can be toggled through sysfs. Volatile.
IFF_PROMISC: receive all packets. Can be toggled through sysfs.
IFF_ALLMULTI: receive all multicast packets. Can be toggled through
    sysfs.
IFF_MASTER: master of a load bal
IFF_SLAVE: slave of a load balan
IFF_MULTICAST: Supports multicas
IFF_PORTSEL: can set media type.
IFF_AUTOMEDIA: auto media select
IFF_DYNAMIC: dialup device with
    through sysfs.
IFF_LOWER_UP: driver signals L1
IFF_DORMANT: driver signals dorm
IFF_ECHO: echo sent packets. Vol
```



```
IFF_802_1Q_VLAN: 802.1Q VLAN device
IFF_EBRIDGE: Ethernet bridging device
IFF_SLAVE_INACTIVE: bonding slave not the curr. active
IFF_MASTER_8023AD: bonding master, 802.3ad
IFF_MASTER_ALB: bonding master, balance-alb
IFF_BONDING: bonding master or slave
IFF_SLAVE_NEEDARP: need ARPs for validation
IFF_ISATAP: ISATAP interface (RFC4214)
IFF_MASTER_ARPMON: bonding master, ARP mon in use
IFF_WAN_HDLC: WAN HDLC device
IFF_XMIT_DST_RELEASE: dev_hard_start_xmit() is allowed to
    release skb->dst
IFF_DONT_BRIDGE: disallow bridging this ether dev
IFF_DISABLE_NETPOLL: disable netpoll at run-time
IFF_MACVLAN_PORT: device used as macvlan port
IFF_BRIDGE_PORT: device used as bridge port
IFF_OVS_DATAPATH: device used as Open vSwitch datapath port
IFF_TX_SKB_SHARING: The interface supports sharing skbs on transmit
IFF_UNICAST_FLT: Supports unicast filtering
IFF_TEAM_PORT: device used as team port
IFF_SUPP_NOFCS: device supports sending custom FCS
IFF_LIVE_ADDR_CHANGE: device supports hardware address
    change when it's running
IFF_MACVLAN: Macvlan device
```

网络设备对象类

- 操作方法集
- struct net_device {
-     ...
-     const struct net_device_ops *netdev_ops;
-     const struct ethtool_ops *ethtool_ops;
-     const struct header_ops *header_ops;
-     ...
- };
- netdev_ops域：
  - 操作接口集
- ethtool_ops域：
  - 管理工具ethtool集
- header_ops域：
  - 数据帧首部操作集

网络设备对象类

```
struct net_device_ops {
        int                     (*ndo_open)(struct net_device *dev);         //打开网卡
        int                     (*ndo_stop)(struct net_device *dev);         //关闭网卡
        netdev_tx_t             (*ndo_start_xmit) (struct sk_buff *skb,      //发送数据
                                            struct net_device *dev);
        int                     (*ndo_set_mac_address)(struct net_device *dev,//设置MAC
                                            void *addr);
        int                     (*ndo_validate_addr)(struct net_device *dev);//检查MAC
        int                     (*ndo_do_ioctl)(struct net_device *dev, struct ifreq *ifr, int cmd);   //标准接口外的参数读写、状态读、控制，等等功能实现
        int                     (*ndo_change_mtu)(struct net_device *dev, //修改mtu值
                                            int new_mtu);
        void                    (*ndo_tx_timeout) (struct net_device *dev); //发送超时
        struct net_device_stats* (*ndo_get_stats)(struct net_device *dev);  //得到统计信息
        ......
};
```

网络设备对象类

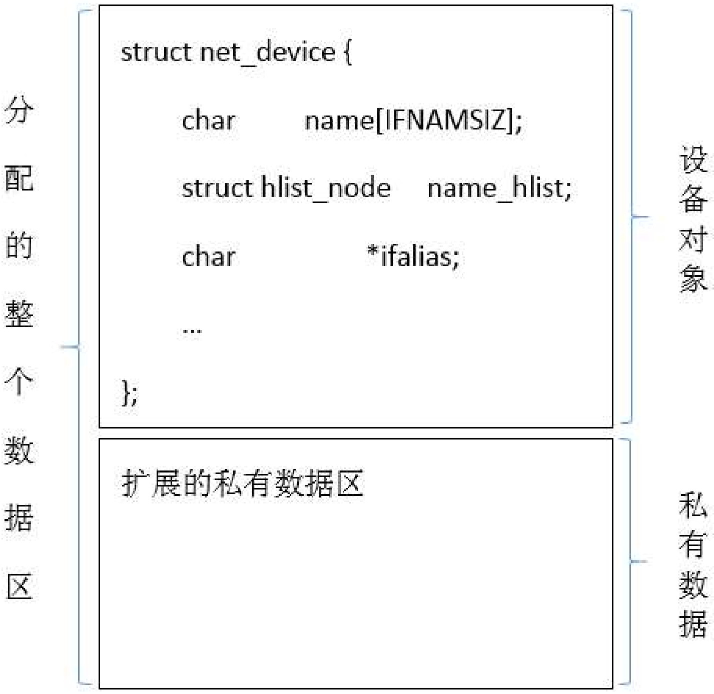
- linux/ethtool.h
- struct ethtool_ops {
    - //读写各种设备设置（ struct ethtool_cmd 描述的信息）
    - int		(*get_settings)(struct net_device *, struct ethtool_cmd *);
    - int		(*set_settings)(struct net_device *, struct ethtool_cmd *);
    - //读驱动信息（ struct ethtool_drvinfo描述的信息）
    - void		(*get_drvinfo)(struct net_device *, struct ethtool_drvinfo *);
    - //当网卡已连接的连接状态
    - u32		(*get_link)(struct net_device *);
    - //读stringset（枚举量）描述的信息
    - void		(*get_strings)(struct net_device *, u32 stringset, u8 *);
    - //读扩展的设备状态信息
    - void		(*get_ethtool_stats)(struct net_device *, struct ethtool_stats *, u64 *);
    - ……
- };

网络设备对象类

```
struct header_ops {
          //创建协议头（Ethernet：int eth_header()）
          int           (*create) (struct sk_buff *skb, struct net_device *dev,
                                   unsigned short type, const void *daddr,
                                   const void *saddr, unsigned int len);
          //复制协议头（Ethernet：int eth_header_parse ()）
          int           (*parse)(const struct sk_buff *skb, unsigned char *haddr);
          //缓存协议头（Ethernet： int eth_header_cache()）
         int (*cache)(const struct neighbour *neigh, struct hh_cache *hh, __be16 type);
          //刷新协议头缓存（Ethernet： void eth_header_cache_update()）
          void          (*cache_update)(struct hh_cache *hh,
                                        const struct net_device *dev,
                                        const unsigned char *haddr);
};
```

网络设备对象类

- 私有数据
- 得到私有数据
- void *netdev_priv(const struct net_device *dev)
- 32bytes对齐
- ALIGN(x, a)



```
struct net_device {
        char          name[IFNAMSIZ];
        struct hlist_node     name_hlist;
        char                  *ifalias;
        …
};
```

内核函数

- /* 功能：分配网络设备对象
- * 参数：
-     int sizeof_priv – 私有数据字节大小
-     const char *name – 物理接口名：范例 “Demo%d”
-     unsigned char name_assign_type – 网络设备名类型
-         NET_NAME_UNKNOWN、NET_NAME_ENUM、…
-     void (*setup)(struct net_device *)- 初始化回调函数
- * 返回值：
-     成功：struct net_device对象首地址
-     失败：NULL
- */
- struct net_device *alloc_netdev(int sizeof_priv, const char *name,
-         unsigned char name_assign_type,
-         void (*setup)(struct net_device *));

内核函数

- /* 功能：分配以太网络设备对象
- * 参数：
- int sizeof_priv – 私有数据字节大小
- * 返回值：
- 成功：struct net_device对象首地址
- 失败：NULL
- */
- struct net_device *alloc_etherdev(int sizeof_priv);
- /* 功能：释放网络设备对象
- * 参数：
- struct net_device *dev – 指向网络设备对象首地址
- */
- void free_netdev(struct net_device *dev);

内核函数

- /* 功能：注册网络设备对象
- * 参数：
-            struct net_device *dev – 指向网络设备对象首地址
- * 返回值：
-            成功：0
-            失败：负数，绝对值是错误码
- */
- int register_netdev(struct net_device *dev);
- /* 功能：注销网络设备对象
- * 参数：
-            struct net_device *dev – 指向网络设备对象首地址
- */
- void unregister_netdev(struct net_device *dev)

内核函数

- 其他常用内核函数
  - 确认网络包的协议ID
  - __be16 eth_type_trans(struct sk_buff *skb, struct net_device *dev);
  - 设置新的MAC
  - int eth_mac_addr(struct net_device *dev, void *p);
  - 修改mtu值
  - int eth_change_mtu(struct net_device *dev, int new_mtu);
  - 判断mac地址是否有效
  - int eth_validate_addr(struct net_device *dev);
  - 随机生成mac地址
  - void eth_hw_addr_random(struct net_device *dev);

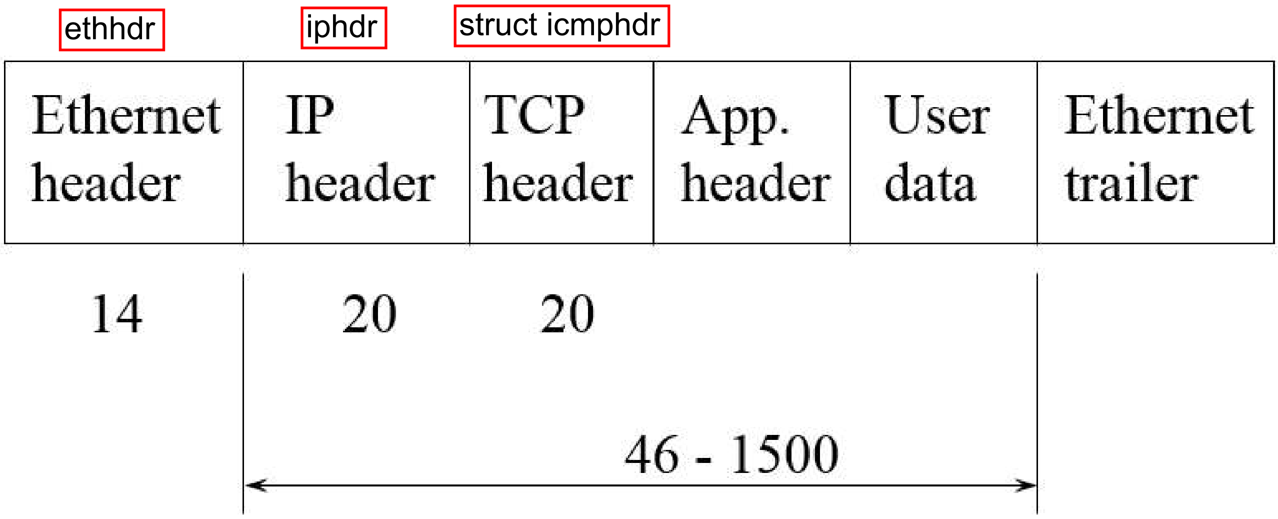
内核函数

- 其他常用内核函数
  - 设置载波（链路连接）
  - void netif_carrier_on(struct net_device *dev)
  - 清除载波（链路断开）
  - void netif_carrier_off(struct net_device *dev)
  - 开启发送队列
  - void netif_start_queue(struct net_device *dev)
  - 停止发送队列
  - void netif_stop_queue(struct net_device *dev)

套接字缓冲区

- 网络通讯中的数据流是一个包的形式，一般的：
- IP包 = 协议头+正文

ping命令走的是icmp协议



ethhdr

iphdr

struct icmphdr

Ethernet header	IP header	TCP header	App. header	User data	Ethernet trailer
14	20	20			

46 - 1500

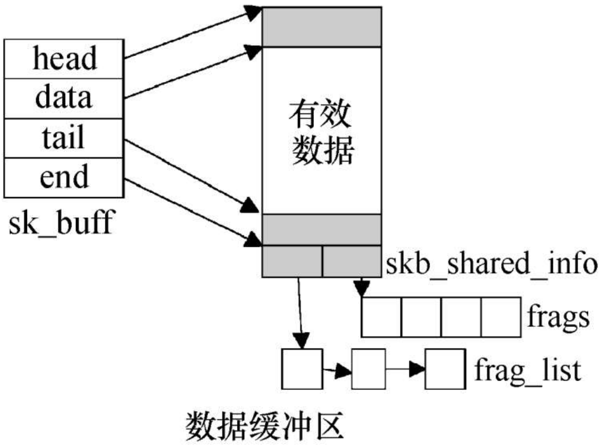
套接字缓冲区

- 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:
  - （1）各层协议头h、nh和mac。

    传输层TCP/UDP（及ICMP和IGMP）协议头h、网络层协议头nh和链路层协议头mac
  - （2）数据缓冲区指针head、data、tail和end。

    指向这片缓冲区不同位置的指针head、data、tail和end。
  - （3）长度信息len

    指数据包有效数据的长度



套接字缓冲区

- Linux套接字缓冲区支持分配、释放、指针移动等功能函数
  - （1）分配
    - struct sk_buff *dev_alloc_skb(unsigned int len);
    - 分配成功之后，因为还没有存放具体的网络数据包，所以sk_buff的data、tail指针都指向存储空间的起始地址head，而len的大小则为0。
  - （2）释放
    - void dev_kfree_skb(struct sk_buff *skb);
    - 用于释放被alloc_skb()函数分配的套接字缓冲区和数据缓冲区

套接字缓冲区

- （3）指针移动
  - Linux套接字缓冲区中的数据缓冲区指针移动操作包括put（放置）、push（推）、pull（拉）、reserve（保留）等。
  - ① put操作
  - unsigned char *skb_put(struct sk_buff *skb, unsigned int len);
  - 将tail指针下移，增加sk_buff的len值，并返回skb->tail的当前值 。
  - ② push操作
  - unsigned char *skb_push(struct sk_buff *skb, unsigned int len);
  - 将data指针上移，因此也要增加sk_buff的len值。
  - ③ pull操作
  - unsigned char * skb_pull(struct sk_buff *skb, unsigned int len);
  - 将data指针下移，并减小skb的len值 。
  - reserve操作
  - void skb_reserve(struct sk_buff *skb, unsigned int len);
  - data指针和tail指针同时下移

套接字缓冲区

```
/* 分配新的套接字缓冲区和数据缓冲区 */
    skb = dev_alloc_skb(length + 2);
    if (skb == NULL)
    {
            .../* 分配失败 */
           return ;
    }
    skb_reserve(skb, 2); /* 预留空间以使网络层协议头对齐 */
  /* 将硬件上接收到的数据复制到数据缓冲区 */
    readwords(ioaddr, RX_FRAME_PORT, skb_put(skb, length), length >> 1);
    if (length &1)
      skb->data[length - 1] = readword(ioaddr, RX_FRAME_PORT);
```