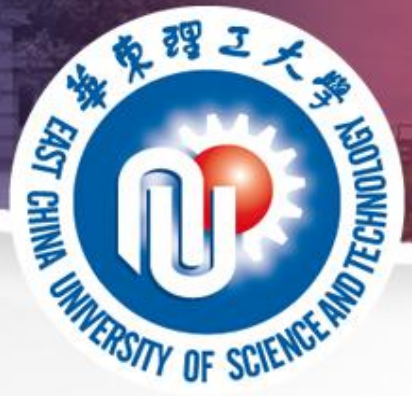


第11-12章 嵌入式驱动程序设计(2)

华东理工大学计算机系

罗 飞

Content

- 块设备驱动程序设计概要
- 块设备驱动程序设计方法
- MMC/SD卡驱动
- 网卡驱动

Content

- 块设备驱动程序设计概要
- 块设备驱动程序设计方法
- MMC/SD卡驱动
- 网卡驱动

块设备驱动程序设计概要（1）

- 块设备是Linux系统中的一大类设备
  - IDE硬盘、SCSI硬盘、CD-ROM等设备
- 块设备数据存取的单位是块
  - 每次能传输一个或多个块，支持随机访问，并采用了缓存技术
- 块设备驱动主要针对磁盘等慢速设备
  - 文件系统一般采用块设备作为载体

块设备驱动程序设计概要（2）

虚拟文件系统结构

块设备驱动程序设计概要（3）

- 块设备的数据交换方式
  - 字符设备以字节为单位进行读写，块设备则以块为单位
  - 块设备还支持随机访问，而字符设备只能顺序访问
- 块设备的I/O请求都有对应的缓冲区，并使用了请求队列对请求进行管理

块设备驱动程序设计概要（4）

- 对块设备的读写是通过请求实现
  - No-op I/O scheduler 实现了一个简单FIFO队列
  - Anticipatory I/O scheduler 当前内核中默认的I/O调度器
  - Deadline I/O scheduler 改善了AS的缺点
  - CFQ I/O schedule 系统内所有任务分配相同的带宽

Content

- 块设备驱动程序设计概要
- 块设备驱动程序设计方法
- MMC/SD卡驱动
- 网卡驱动

块设备驱动程序设计方法

- 相关重要数据结构与函数
- 块设备的注册与注销
- 块设备初始化与卸载
- 块设备操作
- 请求处理

相关重要数据结构与函数（1）

gendisk结构体

表示是一个独立磁盘设备或者一个分区

```
struct gendisk {
/* 只有major, first_minor 和minors是输入变量，不能直接使用，应当使用
  disk_devt() 和 disk_max_parts(). */
      int major;                    /* 主设备号 */
      int first_minor;
      int minors;          /* 次设备号的最大值, 若为1 则该盘不能被分区*/
      char disk_name[DISK_NAME_LEN]; /* 主驱动名称 */
      char *(*nodename)(struct gendisk *gd);
      /* 磁盘分区的指针数组，使用partno进行索引. */
      struct disk_part_tbl *part_tbl;/*分区表*/
      struct hd_struct part0;
      struct block_device_operations *fops;
      struct request_queue *queue;/*请求队列*/
```

相关重要数据结构与函数（2）

```
        void *private_data;
        int flags;
        struct device *driverfs_dev;
        struct kobject *slave_dir;
        struct timer_rand_state *random;
        atomic_t sync_io;       /* RAID */
        struct work_struct async_notify;
        int node_id;
    };
```

分配gendisk: struct gendisk *alloc_disk(int minors);
增加gendisk: void add_disk(struct gendisk *disk);
释放gendisk: void del_gendisk(struct gendisk *gp);
引用计数: get_disk和put_disk
设置和查看磁盘容量: void set_capacity(struct gendisk *disk, sector_t size); sector_t get_capacity(struct gendisk *disk);

request队列作用

- 请求队列跟踪等候的块I/O请求
  - 它存储用于描述这个设备能够支持的请求
  - 如果请求队列被配置正确了，它不会交给该设备一个不能处理的请求
- 还实现一个插入接口
  - 允许使用多个I/O调度器，I/O调度器（也称电梯）的工作是以最优性能的方式向驱动提交I/O请求

相关重要数据结构与函数（3）

request_queue队列

- 一个块请求队列是一个块I/O请求的队列

```
struct request_queue
{
        struct list_head        queue_head;
        struct request          *last_merge;
        struct elevator_queue*elevator;
        ...
        unsigned long           queue_flags;
        /*自旋锁，不能直接应用，应使用->queue_lock访问*/
        spinlock_t              __queue_lock;
        spinlock_t              *queue_lock;
        struct kobject kobj;
        /* 队列设置 */
        unsigned long           nr_requests; /* 最大请求数 */
        unsigned int        nr_congestion_on;
        ...
}
```

相关重要数据结构与函数（4）

- 请求队列的初始化和清除

  request_queue *blk_init_queue_node(request_fn_proc *rfn,spinlock_t *lock, int node_id);

  void blk_cleanup_queue(struct request_queue *q);

- 提取和删除请求

  struct request *elv_next_request(struct request_queue *q);

  void blkdev_dequeue_request(struct request *req);

  void elv_requeue_request(struct request_queue *q, struct request *rq);

- 队列的参数设置

  void blk_stop_queue(struct request_queue *q);

  void blk_start_queue(struct request_queue *q);

- 内核通告

  void blk_queue_segment_boundary(struct request_queue *q, unsigned long mask);

相关重要数据结构与函数（5）

request结构

使用request结构体来描述的I/O请求

```
struct request {
        struct list_head queuelist;
        struct call_single_data csd;
        int cpu;
        struct request_queue *q;
        unsigned int cmd_flags;
        enum rq_cmd_type_bits cmd_type;
        unsigned long atomic_flags;
        sector_t sector;                /* 下一个传输的扇区 */
        sector_t hard_sector;           /* 下一个完成的扇区 */
        ~unsigned long nr_sectors;      /* 未提交的扇区数 */
        unsigned long hard_nr_sectors;      /* 未完成的扇区数*/
        /* 当前段中未提交的扇区数 */
        unsigned int current_nr_sectors;
```

相关重要数据结构与函数（6）

```
/* 当期段中未完成的扇区数 */
unsigned int hard_cur_sectors;
struct bio *bio;
struct bio *biotail;
struct hlist_node hash;      /* 混合hash */
void *elevator_private;
void *elevator_private2;
struct gendisk *rq_disk;
unsigned long start_time;
unsigned short nr_phys_segments;
unsigned short ioprio;
void *special;
char *buffer;
int tag;
int errors;
int ref_count;
...
};
```

相关重要数据结构与函数（7）

- request实质上是一个bio结构的链表实现。bio是底层对部分块设备的I/O请求描述，其包含了驱动程序执行请求所需的全部信息。通常一个I/O请求对应一个bio。I/O调度器可将关联的bio合并成一个请求。

```
struct bio {
        sector_t        bi_sector;
        struct bio              *bi_next;    /*请求队列指针 */
        struct block_device     *bi_bdev;
        unsigned long           bi_flags;     /* 状态，命令等*/
        unsigned long           bi_rw;        /*最后一位为读写标志位,
                                 * 前面的为优先级*/
        unsigned short          bi_vcnt;      /* bio_vec数*/
        unsigned short          bi_idx;       /* 当前bio_vec中的索引 */
```

相关重要数据结构与函数（8）

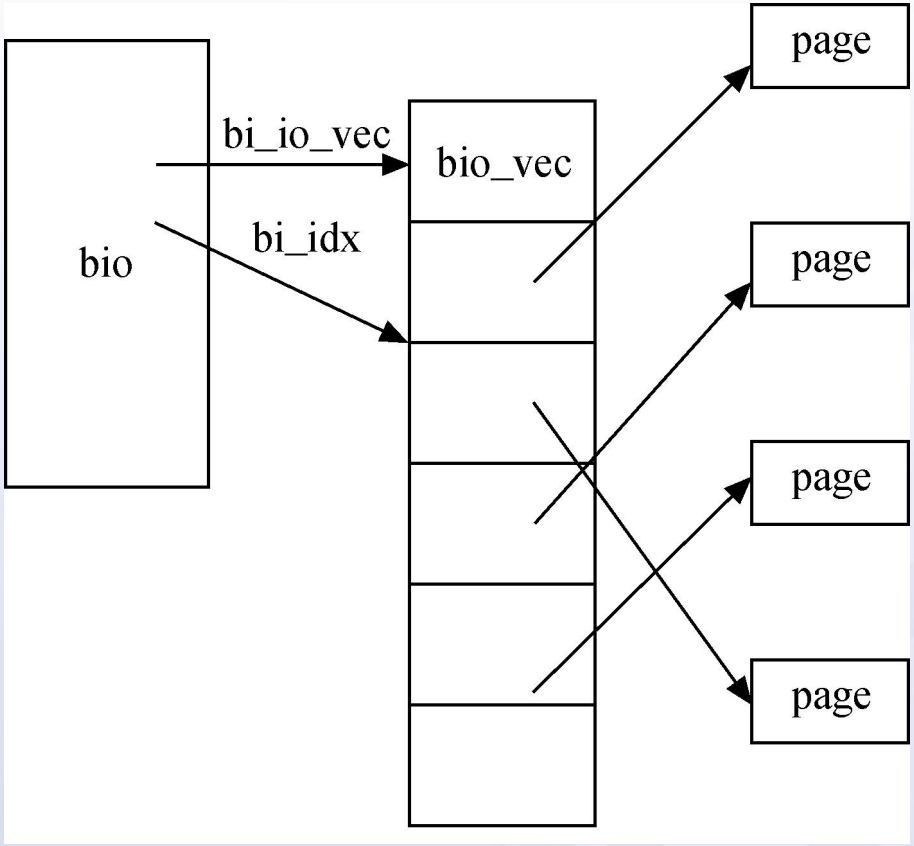
```
        /* 该bio的分段信息（设置了物理地址聚合有效） */
        unsigned int        bi_phys_segments;
        unsigned int        bi_size;
        unsigned int        bi_seg_front_size;
        unsigned int        bi_seg_back_size;
        unsigned int        bi_max_vecs;  /* 最大bvl_vecs数*/
        unsigned int        bi_comp_cpu;
        atomic_t            bi_cnt;             /* 针脚数 */
        struct bio_vec          *bi_io_vec;    /*真正的vec列表 */
        ...
};
struct bio_vec {
        struct page     *bv_page;
        unsigned int    bv_len;
        unsigned int    bv_offset;
};
```

bio_vec结构体

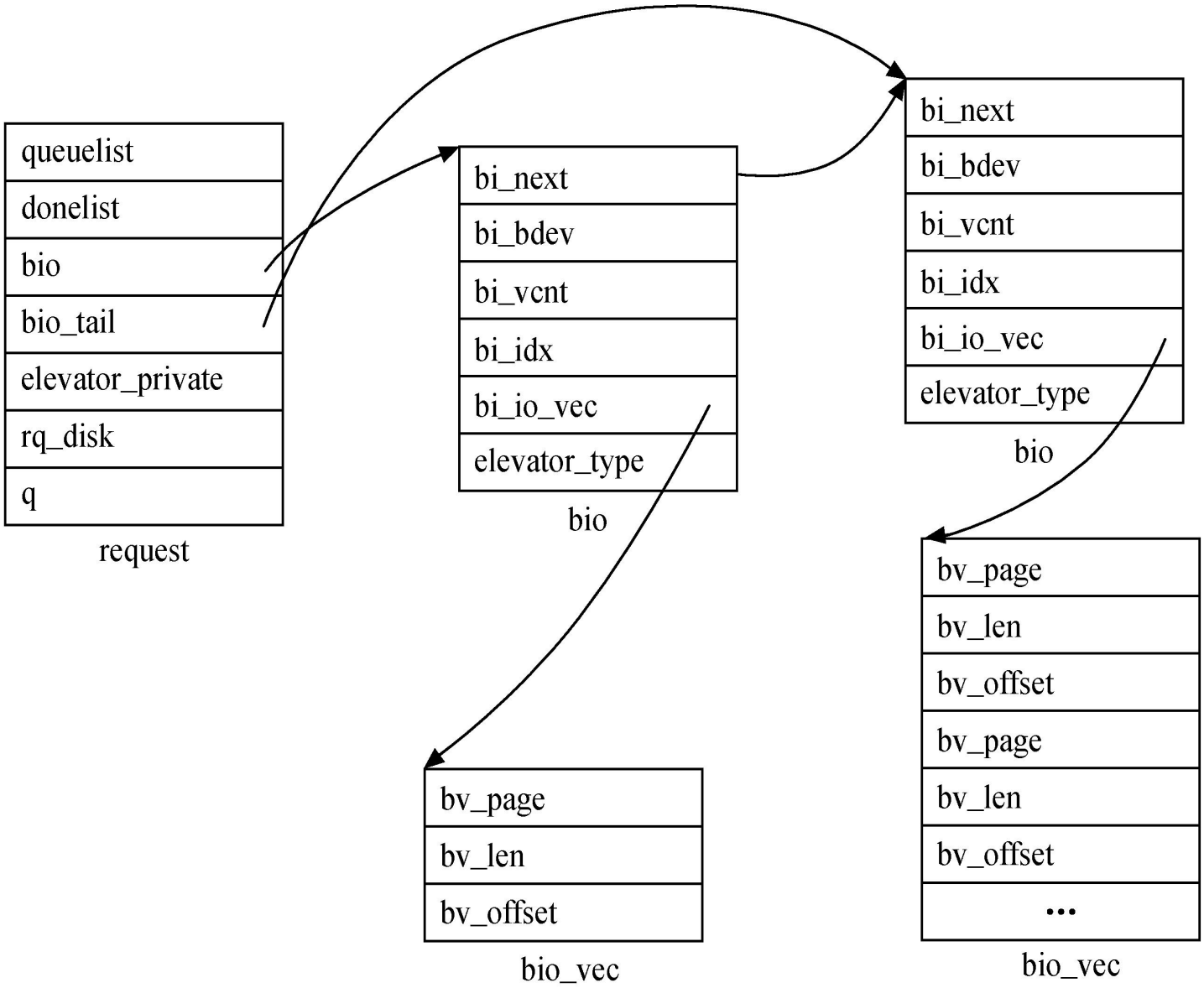
- bio的核心是一个称为bi_io_vec的数组

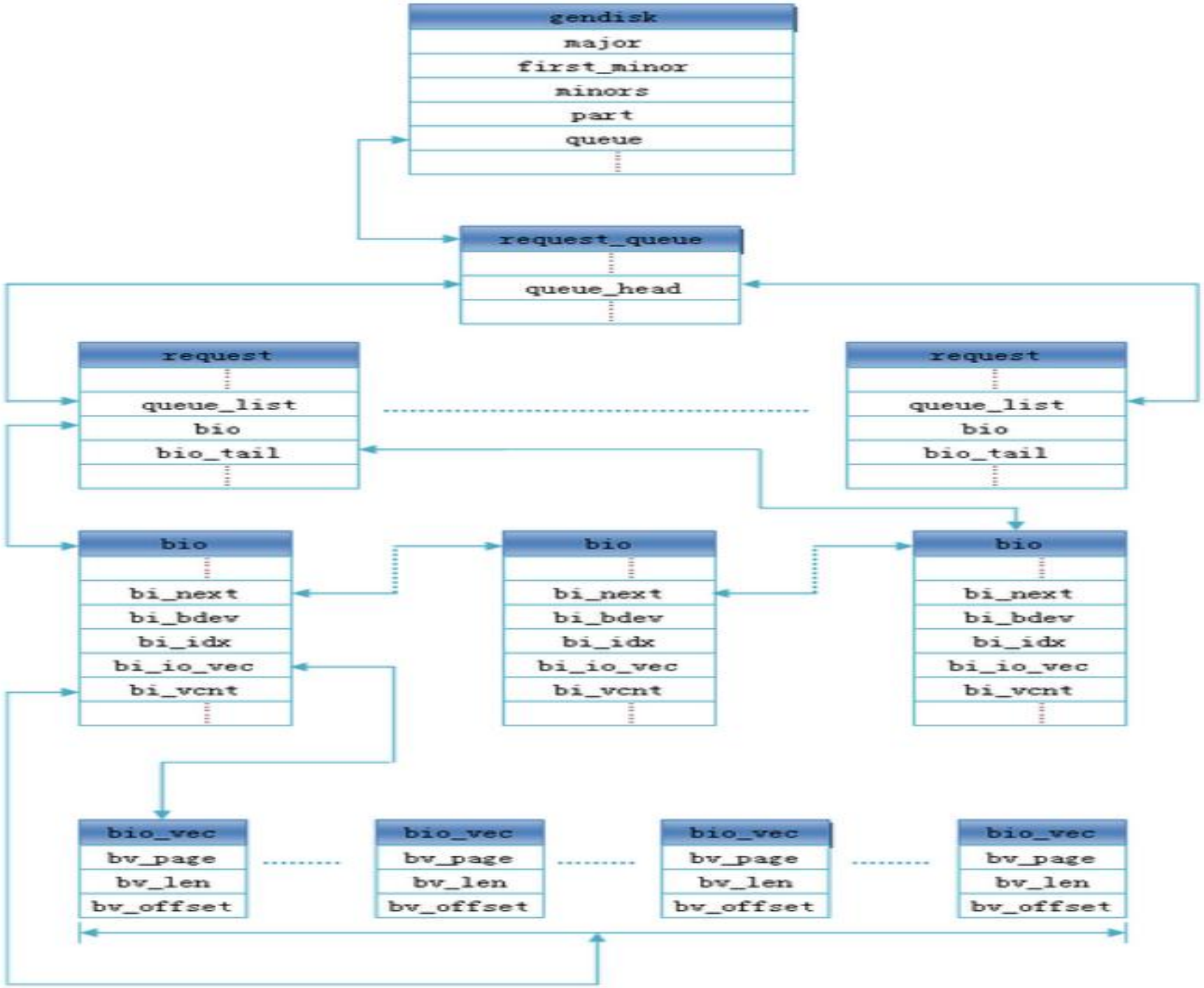
```
struct bio_vec
{
    struct page *bv_page;  /* 页指针 */
   unsigned int bv_len;  /* 传输的字节数 */
   unsigned int bv_offset; /* 偏移位置 */
};
```

bio与bio_vec



request、bio、bio_vec




gendisk
major
first_minor
minors
part
queue
request_queue
queue_head
request
queue_list
bio
bio_tail
request
queue_list
bio
bio_tail
bio
bi_next
bi_bdev
bi_idx
bi_io_vec
bi_vcnt
bio
bi_next
bi_bdev
bi_idx
bi_io_vec
bi_vcnt
bio
bi_next
bi_bdev
bi_idx
bi_io_vec
bi_vcnt
bio_vec
bv_page
bv_len
bv_offset
bio_vec
bv_page
bv_len
bv_offset
bio_vec
bv_page
bv_len
bv_offset
bio_vec
bv_page
bv_len
bv_offset

块设备的注册与注销

- 注册

  int register_blkdev(unsigned int major, const char *name);

- 注销

  void unregister_blkdev(unsigned int major, const char *name);

块设备初始化与卸载

初始化

- 注册块设备及块设备驱动程序
- 分配、初始化、绑定请求队列（如果使用请求队列的话）
- 分配、初始化gendisk，为相应的成员赋值并添加gendisk。
- 其它初始化工作，如申请缓存区，设置硬件尺寸（不同设备，有不同处理）

卸载

- 删除请求队列
- 撤销对gendisk的引用并删除gendisk
- 释放缓冲区，撤销对块设备的应用，注销块设备驱动

块设备操作

```
struct block_device_operations {
    /* 打开与释放*/
    int (*open) (struct block_device *, fmode_t);
    int (*release) (struct gendisk *, fmode_t);
    /* I/O操作 */
    int (*locked_ioctl) (struct block_device *, fmode_t, unsigned, unsigned long);
    int (*ioctl) (struct block_device *, fmode_t, unsigned, unsigned long);
    int (*compat_ioctl) (struct block_device *, fmode_t, unsigned, unsigned long);
    int (*direct_access) (struct block_device *, sector_t,void **, unsigned long *);
    /*介质改变*/
    int (*media_changed) (struct gendisk *);
    unsigned long long (*set_capacity) (struct gendisk *,unsigned long long);
    /* 使介质有效 */
    int (*revalidate_disk) (struct gendisk *);
    /*获取驱动器信息 */
    int (*getgeo)(struct block_device *, struct hd_geometry *);
    struct module *owner;
};
```

请求处理

- 块设备不像字符设备操作，它并没有保护read和write。对块设备的读写是通过请求函数完成的，因此请求函数是块设备驱动的核心。
  - 使用请求队列: 使用请求队列对于提高机械磁盘的读写性能具有重要意义，I/O调度程序按照一定算法（如电梯算法）通过优化组织请求顺序，帮助系统获得较好的性能。
  - 不使用请求队列: 但是对于一些本身就支持随机寻址的设备，如SD卡、RAM盘、软件RAID组件、虚拟磁盘等设备，请求队列对其没有意义。针对这些设备的特点，块设备层提供了“无队列”的操作模式

Content

MMC卡驱动

- MMC/SD芯片介绍
- MMC/SD卡驱动结构
- MMC卡块设备驱动分析
- HSMCI接口驱动设计分析

MMC/SD芯片介绍

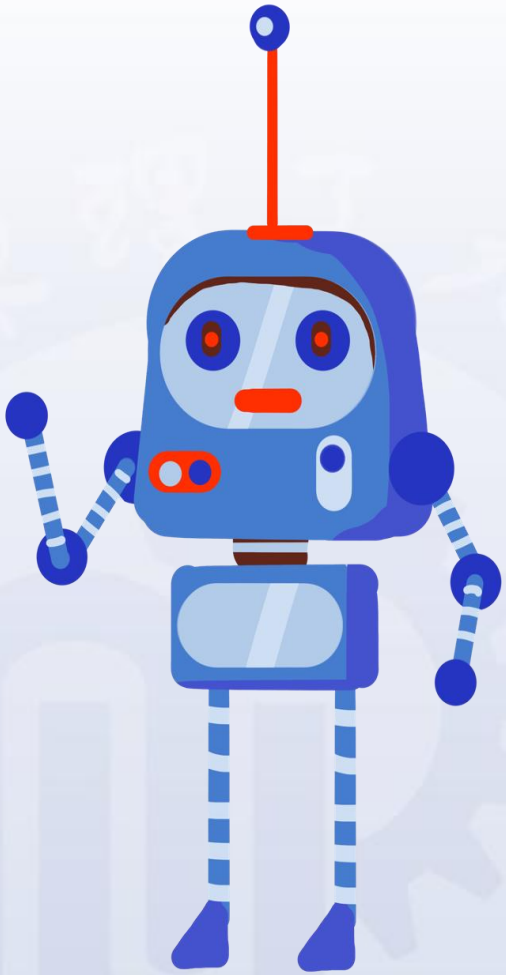
- MMC卡（Multimedia Card）是一种快闪记忆卡标准。在1997年由西门子及SanDisk共同开发，该技术基于东芝的NAND快闪记忆技术。它相对于早期基于Intel NOR Flash技术的记忆卡（例如CF卡），体积小得多
- SD卡（Secure Digital Memory Card）也是一种快闪记忆卡，同样被广泛地在便携式设备上使用，例如数字相机、PDA和多媒体播放器等。SD卡的数据传送协议和物理规范是在MMC卡的基础上发展而来。SD卡比MMC卡略厚，但SD卡有较高的数据传送速度，而且不断地更新标准

MMC/SD卡驱动结构

文件系统
块设备驱动(driver/mmc/card)
MMC/SD核心(driver/mmc/core)
MMC/SD接口(driver/mmc/host)

MMC卡块设备驱动分析

- 注册与注销
  - mmc_blk_init
  - mmc_blk_exit
- 设备加载与卸载
  - mmc_blk_probe
  - mmc_blk_remove
- 设备的打开与释放
  - mmc_blk_open
  - mmc_blk_release
- MMC驱动的请求处理函数
  - mmc_prep_request
  - mmc_blk_issue_rq
  - mmc_requset



Content

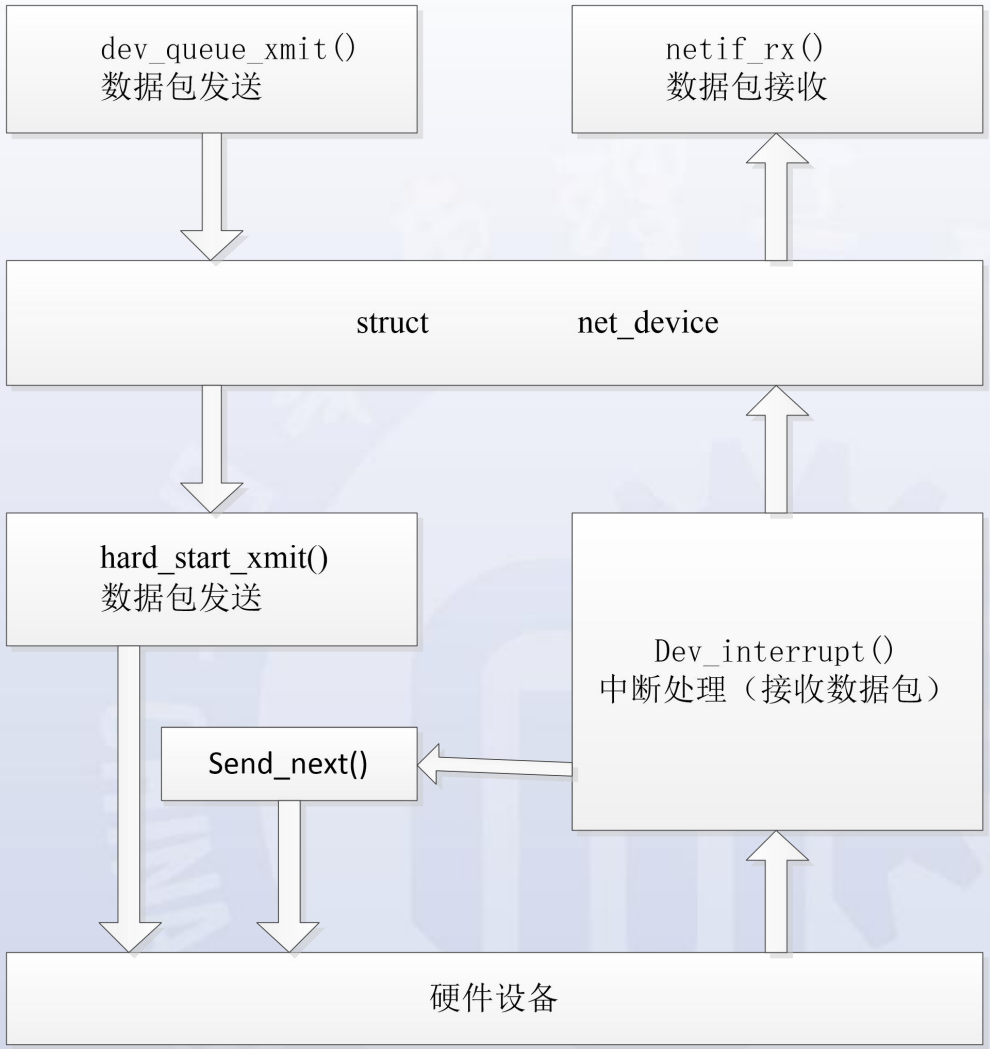
- 块设备驱动程序设计概要
- 块设备驱动程序设计方法
- MMC/SD卡驱动
- 网卡驱动

网卡驱动程序设计概要（1）

- 以太网对应于ISO网络分层中的数据链路层和物理层，其中数据链路层分为逻辑链路控制子层LLC（Logic Link Control）和介质访问控制子层MAC（Media Access Control)
- 以太网接口包括了介质访问控制子层（MAC）和物理层（PHY）。在以太网控制器中MAC控制器的功能是连接和控制物理接口，并实现MAC协议。而PHY负责具体的数据收发
- 在许多嵌入式处理器中都集成了MAC控制器，但是处理器通常是不集成物理层接收器（PHY）的，在这种情况下需要外接PHY芯片，如RTL8201BL、VT6103等

网卡驱动程序设计概要（2）

网络驱动框架



网卡驱动程序设计方法

- 网络设备驱动基本数据结构
- 网络设备初始化
- 打开和关闭接口
- 数据接收与发送
- 查看状态与参数设置

基本数据结构（1）

net_device数据结构

```
char        name[IFNAMSIZ];       /*以下为全局信息 */
int         (*init)(struct net_device *dev);
unsigned long          mem_end;      /* 以下为硬件信息*/
unsigned long          mem_start;
unsigned long          base_addr;
unsigned int           irq;
unsigned char          if_port;
unsigned char          dma;
struct net_device_stats    stats;
unsigned               mtu;      /* 以下为接口信息*/
unsigned short         type;
unsigned short         hard_header_len;
unsigned char          dev_addr[MAX_ADDR_LEN];
unsigned char          broadcast[MAX_ADDR_LEN];
```

基本数据结构（2）

操作函数

```
Int    (*open)(struct net_device *dev);
Int    (*stop)(struct net_device *dev);
Int    (*hard_start_xmit) (struct sk_buff *skb, struct net_device
*dev);
void (*set_multicast_list)(struct net_device *dev);
int    (*set_mac_address)(struct net_device *dev, void *addr);
int    (*do_ioctl)(struct net_device *dev, struct ifreq *ifr, int cmd);
int     (*set_config)(struct net_device *dev,struct ifmap *map);
int     (*change_mtu)(struct net_device *dev, int new_mtu);
void    (*tx_timeout) (struct net_device *dev);
struct net_device_stats* (*get_stats)(struct net_device *dev);
void    (*poll_controller)(struct net_device *dev);
```

基本数据结构（3）

sk_buffer 数据结构

```
/* 网络协议头 */
sk_buff_data_t        transport_header;
sk_buff_data_t        network_header;
sk_buff_data_t        mac_header;
/* 缓冲区指针 */
sk_buff_data_t        tail;
sk_buff_data_t        end;
unsigned char         *head,*data;
/* 操作函数 */
struct sk_buff  *alloc_skb(unsigned int size,gfp_t priority);
void kfree_skb(struct sk_buff *skb);
unsigned char *skb_put(struct sk_buff *skb, unsigned int len);
unsigned char *skb_push(struct sk_buff *skb, unsigned int len);
 unsigned char *skb_pull(struct sk_buff *skb, unsigned int len);
```

网络设备初始化

- 主要对net_device结构体进行初始化
- 由net_device的init函数指针指向的函数完成，当加载网络驱动模块时该函数就会被调用
  - 检测网络设备的硬件特征，检查物理设备是否存在。
  - 检测到设备存在，则进行资源配置。
  - 对net_device成员变量进行赋值。

打开和关闭接口

- 打开接口
  - 在数据包放送前，必须打开接口并初始化接口
  - 打开接口的工作由net_device的open函数指针指向的函数完成，该函数负责的工作包括请求系统资源，如申请I/O区域、DMA通道及中断等资源
  - 告知接口开始工作，调用netif_start_queue激活设备发送队列。
- 关闭接口
  - 该操作由net_device的stop函数指针指向的函数完成，该函数需要调用netif_stop_queue停止数据包传送

数据接收与发送（1）

- 数据发送
  - 数据在实际发送的时候会调用net_device结构的hard_start_transmit函数指针指向的函数，该函数会将要发送的数据放入外发队列，并启动数据包发送
- 并发控制
  - 发送函数在指示硬件开始传送数据后就立即返回，但数据在硬件上的传送不一定完成。硬件接口的传送方式是异步的，发送函数又是可重入的，可利用net_device结构中的xmit_lock自旋锁来保护临界区资源
- 传输超时
  - 驱动程序需要处理超时带来的问题，内核会调用net_device的tx_timeout，完成超时需做的工作，并调用netif_wake_queue函数重启设备发送队列

数据接收与发送（2）

数据接收

- 中断方式：当网络设备接收到数据后触发中断，中断处理程序判断中断类型。如果是接收中断，则接收数据，并申请sk_buffer结构和数据缓冲区，根据数据的信息填写sk_buffer结构。然后将接收到的数据复制到缓冲区，最后调用netif_rx函数将skb传递给上层协议
- 轮询方式：在轮询方式下，首个数据包到达产生中断后触发轮询过程，轮询中断处理程序首先关闭“接收中断”，在接收到一定数量的数据包并提交给上层协议后，再开中断等待下次轮询处理。使用netif_receive_skb函数向上层传递数据

查看状态与参数设置（1）

- 链路状态：驱动程序可以通过查看设备的寄存器来获得链路状态信息。当链路状态改变时，驱动程序需要通知内核
  - void netif_carrier_off(struct net_device *dev);
  - void netif_carrier_on(struct net_device *dev);
- 设备状态：驱动程序的get_stats()函数向用户返回设备的状态和统计信息，保存在一个net_device_stats结构体。
- 设置MAC地址：调用ioctl并且参数为SIOCSIFHWADDR时，就会调用set_mac_address函数指针指向的函数。
- 接口参数设置：调用ioctl并且参数为SIOCSIFMAP时，就会调用set_config函数指针指向的函数，内核会给该函数传递一个ifmap的结构体。该结构体中包含了要设置的I/O地址、中断等信息

查看状态与参数设置（2）

```
struct net_device_stats
{
    unsigned long   rx_packets;     /*收到的数据包数    */
    unsigned long   tx_packets;     /*发送的数据包数    */
    unsigned long   rx_bytes;       /*收到的字节数 */
    unsigned long   tx_bytes;       /*发送的字节数 */
    unsigned long   rx_errors;      /*收到的错误包数    */
    unsigned long   tx_errors;      /*发送的错误包数    */
    unsigned long   rx_dropped;     /* 接收包丢包数*/
    unsigned long   tx_dropped;     /* 发送包丢包数*/
    unsigned long   multicast;      /*收到的广播包数    */
    unsigned long   collisions;
```

查看状态与参数设置（3）

```
/* 详细接收错误信息: */
unsigned long      rx_length_errors; /*接收长度错误 */
unsigned long      rx_over_errors;   /*溢出错误       */
unsigned long      rx_crc_errors;    /*CRC校验错误   */
unsigned long      rx_frame_errors;  /*帧对齐错误     */
unsigned long      rx_fifo_errors;   /* 接收fifo错误  */
/* 详细发送错误信息 */
unsigned long      tx_aborted_errors; /*发送中止          */
unsigned long      tx_carrier_errors; /*载波错误       */
unsigned long      tx_fifo_errors;   /*发送fifo错误    */
unsigned long      tx_window_errors;    /*发送窗口错误 */
```

AT91SAM9G45网卡驱动

- 设备侦测：macb_probe
- 设备打开与关闭：macb_open/macb_close
- 数据的接收：macb_rx_frame
- 数据的发送：macb_tx
- 中断处理：macb_interrupt

THANKS!