浙江大学
ZHEJIANG UNIVERSITY

嵌入式系统

第11课 嵌入式驱动程序设计（2）

王总辉
zhwang@zju.edu.cn

http://course.zju.edu.cn

提纲

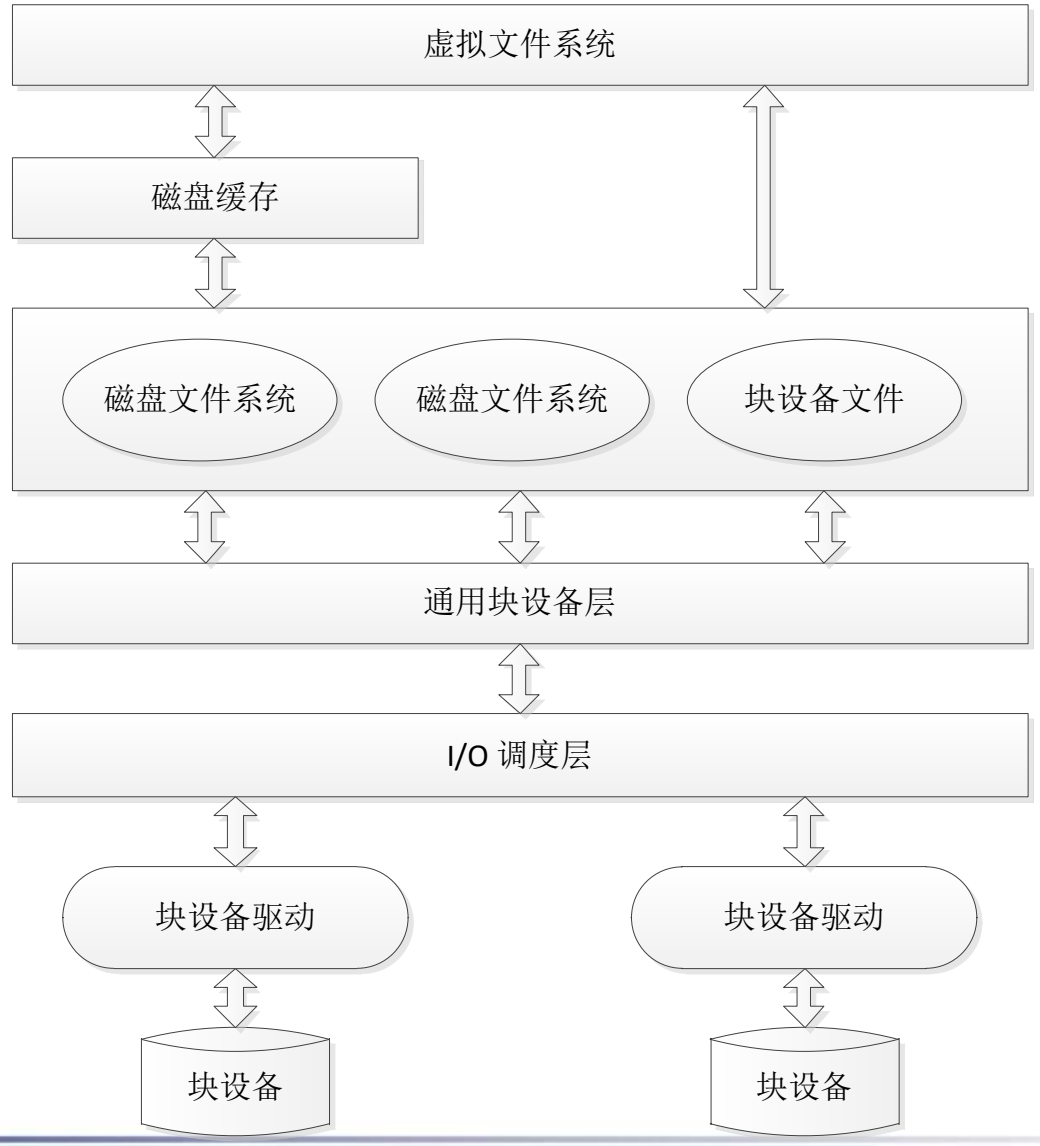
- 块设备驱动程序设计概要
- 块设备驱动程序设计框架
- MMC/SD卡驱动简介

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，包括IDE硬盘、SCSI硬盘、CD-ROM等设备
- 块设备数据存取的单位是块，块的大小通常为512字节到32K字节不等；块设备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，由于其支持随机访问，所以文件系统一般采用块设备作为载体

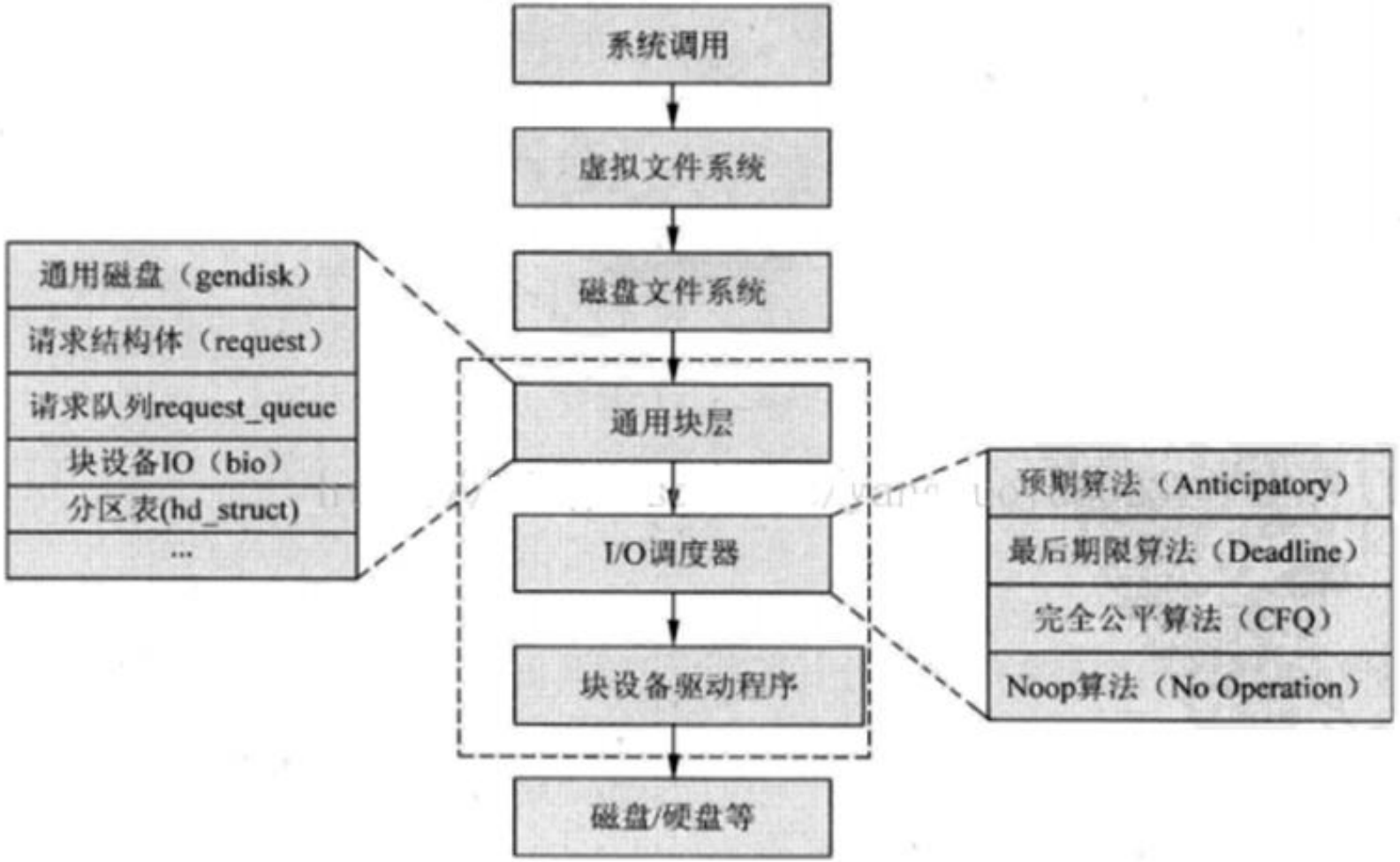
块设备驱动程序设计概要（2）

虚拟文件系统结构



块设备驱动程序设计概要（2）

- 块设备驱动程序和文件系统



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，合理的组织请求的顺序（如电梯算法），尽量进行顺序访问，可得到更好的性能
  - 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，但比较庞大与复杂的，在数据吞吐量非常大的数据库系统中它会变的比较缓慢
  - 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，提供一个公平的工作环境，比较适合桌面环境

块设备驱动程序设计框架

- 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);

相关重要数据结构与函数（3）

request结构

```
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（4）

```
/* 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（5）

request_queue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（6）

- 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（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;
};
```

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
  - 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

MMC/SD卡驱动

- 驱动简介
- 硬件原理图和工作模式
- 驱动请求处理流程
- 驱动结构

MMC/SD驱动简介（1）

- 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驱动简介（2）

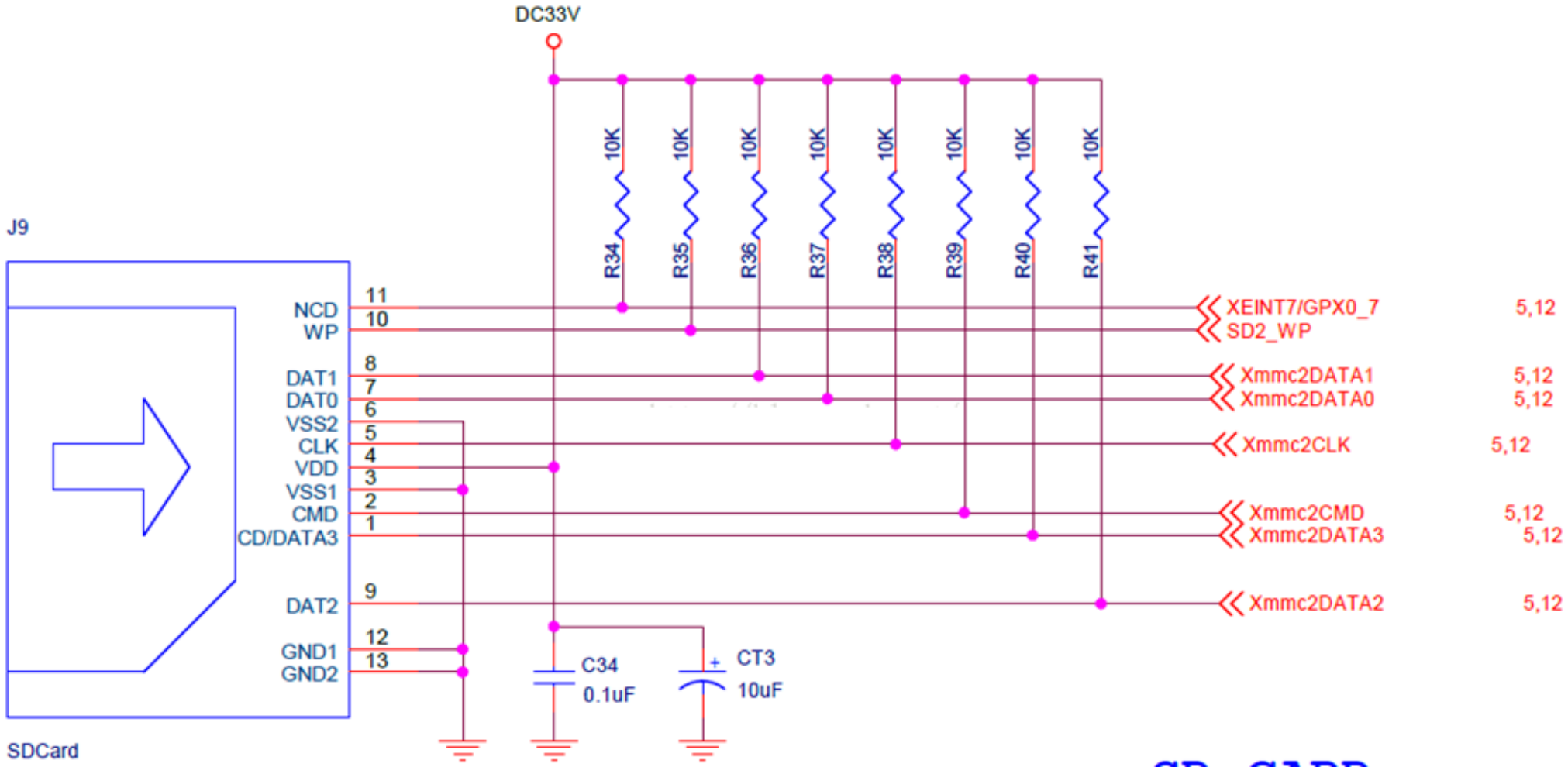
- SDIO是在SD标准上定义了一种外设接口，它和SD卡规范上增加了低速标准。在SDIO卡只需要SPI和1位SD传输模式。低速卡的目标应用是以最小的硬件开销支持低速IO能力。
- 手机或是手持装置都支持 SDIO 的功能，许多SDIO外设也都被开发出来
  - Wi-Fi card （无线网络卡）、CMOS sensor card
  - GPS card、GSM/GPRS modem card、Bluetooth card
- SDIO 的应用将是未来嵌入式系统最重要的接口技术之一，并且也会取代目前GPIO式的SPI接口。

MMC/SD驱动简介（3）

卡类型 / 卡属性	MMC卡	SD卡	SDIO卡
引脚个数	7	9	9
宽度	24mm	24mm	24mm
长度	32mm	32mm	32mm+
厚度	1.4mm	2.1mm	2.1mm
SPI传输模式	可选	支持	支持
1位传输模式	是	是	是
4位传输模式	否	可选	可选
时钟频率	0-20MHz	0-25MHz	0-25MHz
最高传输率	20Mbit/s	100Mbit/s	100Mbit/s
最高SPI传输率	20Mbit/s	25Mbit/s	25Mbit/s

- MCI是Multimedia Card Interface的简称，即多媒体卡接口。上述的MMC,SD,SDIO卡定义的接口都属于MCI接口。MCI这个术语在驱动程序中经常使用，很多文件，函数名字都包括”mci”

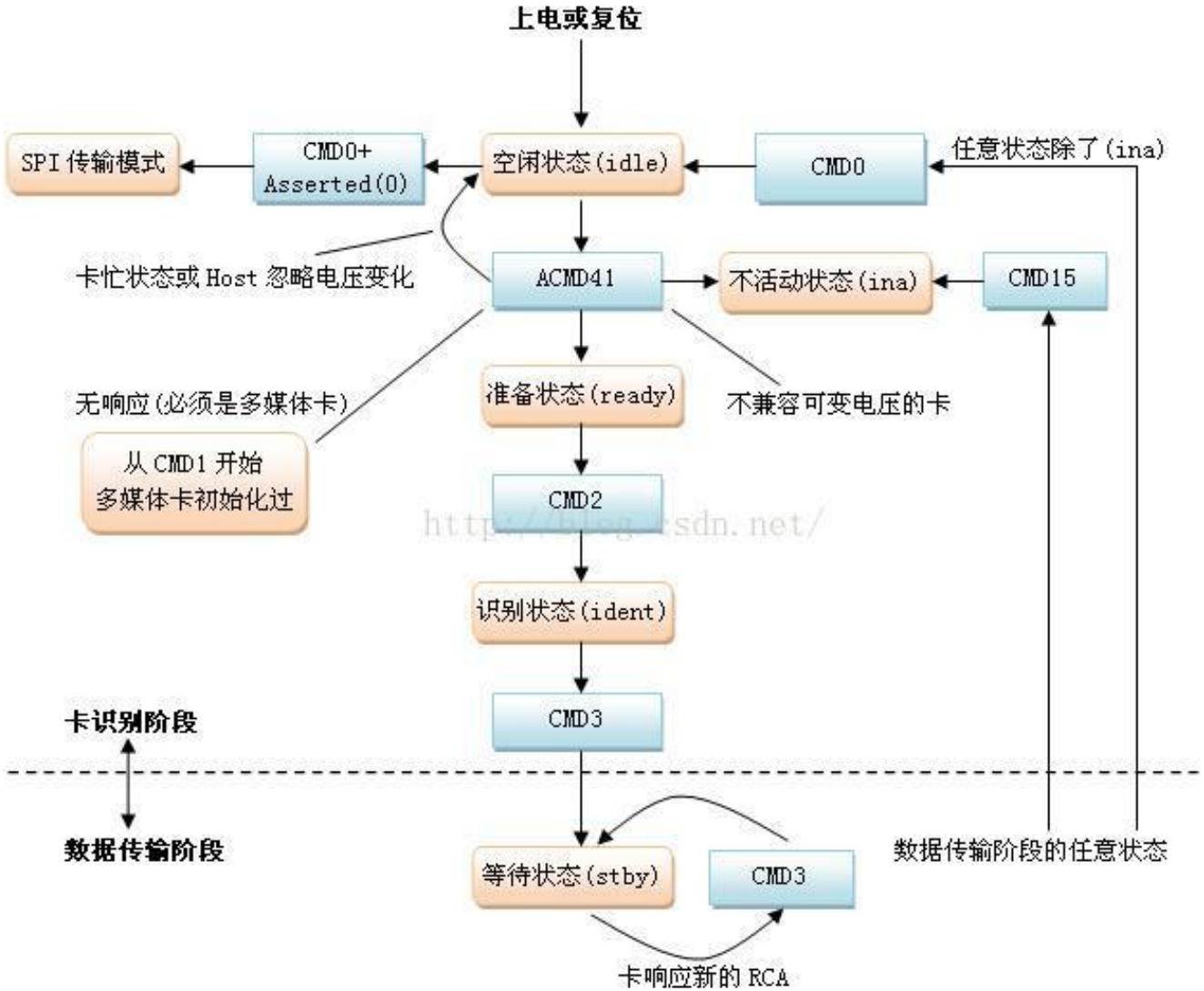
硬件原理图


DC33V
J9
NCD
WP
DAT1
DAT0
VSS2
CLK
VDD
VSS1
CMD
CD/DATA3
DAT2
GND1
GND2
11
10
8
7
6
5
4
3
2
1
9
12
13
R34
R35
R36
R37
R38
R39
R40
R41
10K
C34
0.1uF
CT3
10uF
SDCard
XEINT7/GPX0_7
SD2_WP
Xmmc2DATA1
Xmmc2DATA0
Xmmc2CLK
Xmmc2CMD
Xmmc2DATA3
Xmmc2DATA2
5,12
SD CARD

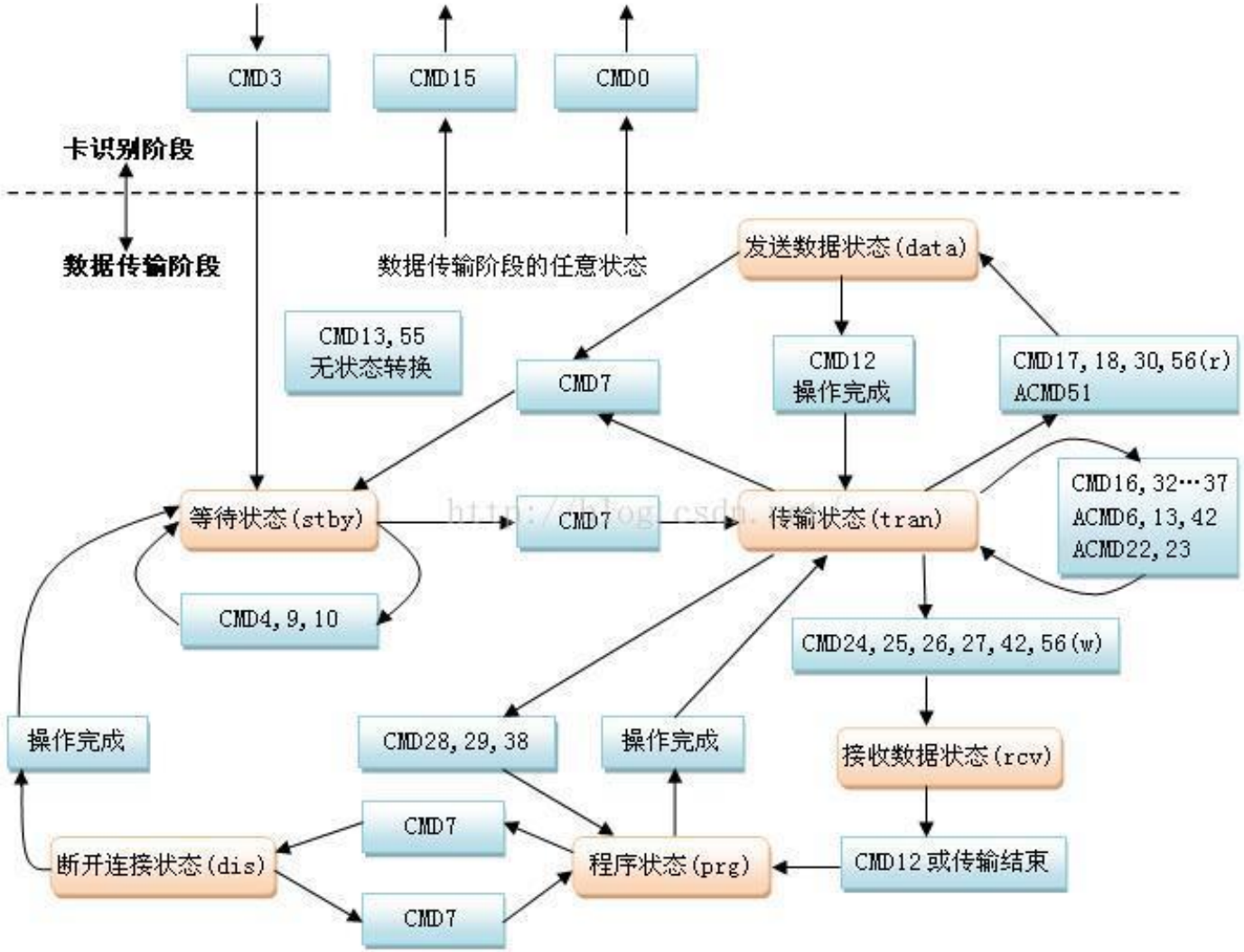
工作模式

- MMC/SD卡两种工作模式，MMC模式是默认工作模式，具有MMC的全部特性。而SPI模式则是MMC协议的一个子集，用于低速系统
  - MMC模式物理层定义：
    - D0-D3 数据传送
    - CMD 进行CMD 和Respons
    - CLK HOST时钟信号线
    - VDD VSS 电源和地
  - SPI模式物理层定义：
    - CLK HOST时钟信号线
    - DATAIN HOST SD Card数据输入信号线
    - DATAOUT HOST SD Card数据输出信号线

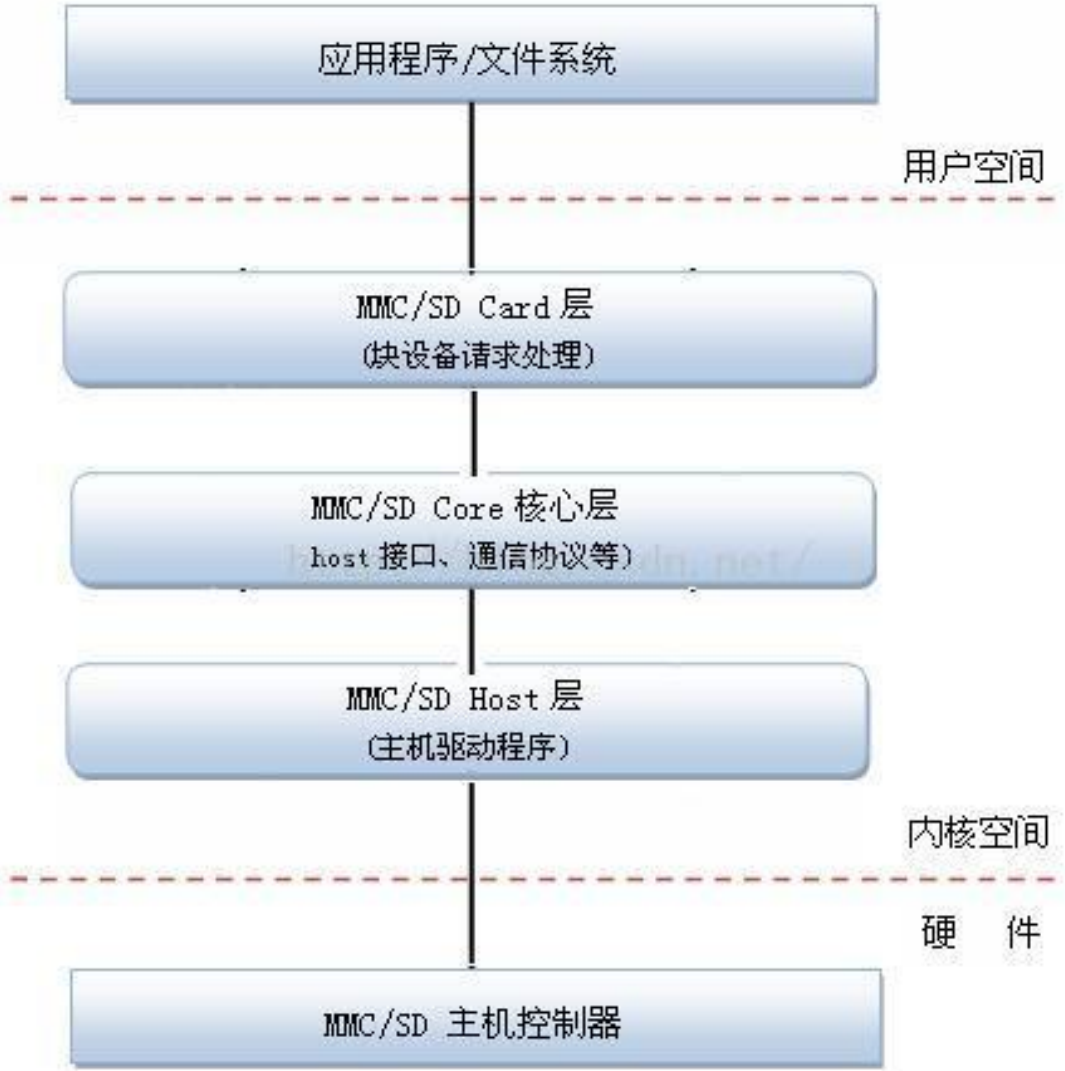
请求处理流程—卡识别


上电或复位
空闲状态(idle)
SPI 传输模式
CMD0+
Asserted(0)
CMD0
任意状态除了(ina)
卡忙状态或 Host 忽略电压变化
ACMD41
不活动状态(ina)
CMD15
无响应(必须是多媒体卡)
从 CMD1 开始
多媒体卡初始化过
准备状态(ready)
不兼容可变电压的卡
CMD2
识别状态(ident)
CMD3
卡识别阶段
数据传输阶段
等待状态(stby)
CMD3
数据传输阶段的任意状态
卡响应新的 RCA

请求处理流程—数据传输



MMC/SD卡驱动结构


应用程序/文件系统
用户空间
MMC/SD Card 层
(块设备请求处理)
MMC/SD Core 核心层
host 接口、通信协议等)
MMC/SD Host 层
(主机驱动程序)
内核空间
硬 件
MMC/SD 主机控制器

MMC/SD卡驱动结构

- 整个MMC/SD模块中最重要的部分是Core核心层，提供了一系列的接口函数，对上提供了将主机驱动注册到系统，给应用程序提供设备访问接口，对下提供了对主机控制器控制的方法及块设备请求的支持。
- 对于主机控制器的操作就是对相关寄存器进行读写，而对于MMC/SD设备的请求处理则比较复杂

MMC卡块设备驱动案例

- 注册与注销
  - mmc_blk_init
  - 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

□谢　谢！