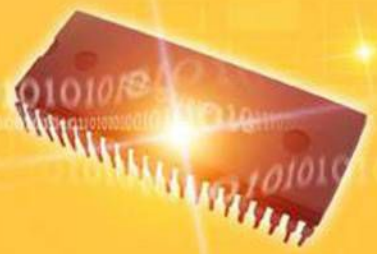
嵌入式系统工程师



Linux块设备驱动设计

大纲

- 概述
- 块设备组件构成
- 块设备的数据结构
- 块设备的注册
- 块设备的数据传输
- 使用块设备

大纲

- 概述
- 块设备组件构成
- 块设备的数据结构
- 块设备的注册
- 块设备的数据传输
- 使用块设备

概述

- 块设备
  - 数据传输以块为单位，而字符以字节为单位
  - 块设备对数据请求有缓冲区，因此可以调整响应请求的顺序
  - 牵涉到内核组件，主要与内核文件系统打交道，有些晦涩，但我们不必了解这些，把重心放在具体的块设备驱动上
- 常见的块设备
  - 硬盘、光盘、SD卡、U盘。。。

概述

- 块设备基本概念
  - 扇区(Sectors)：

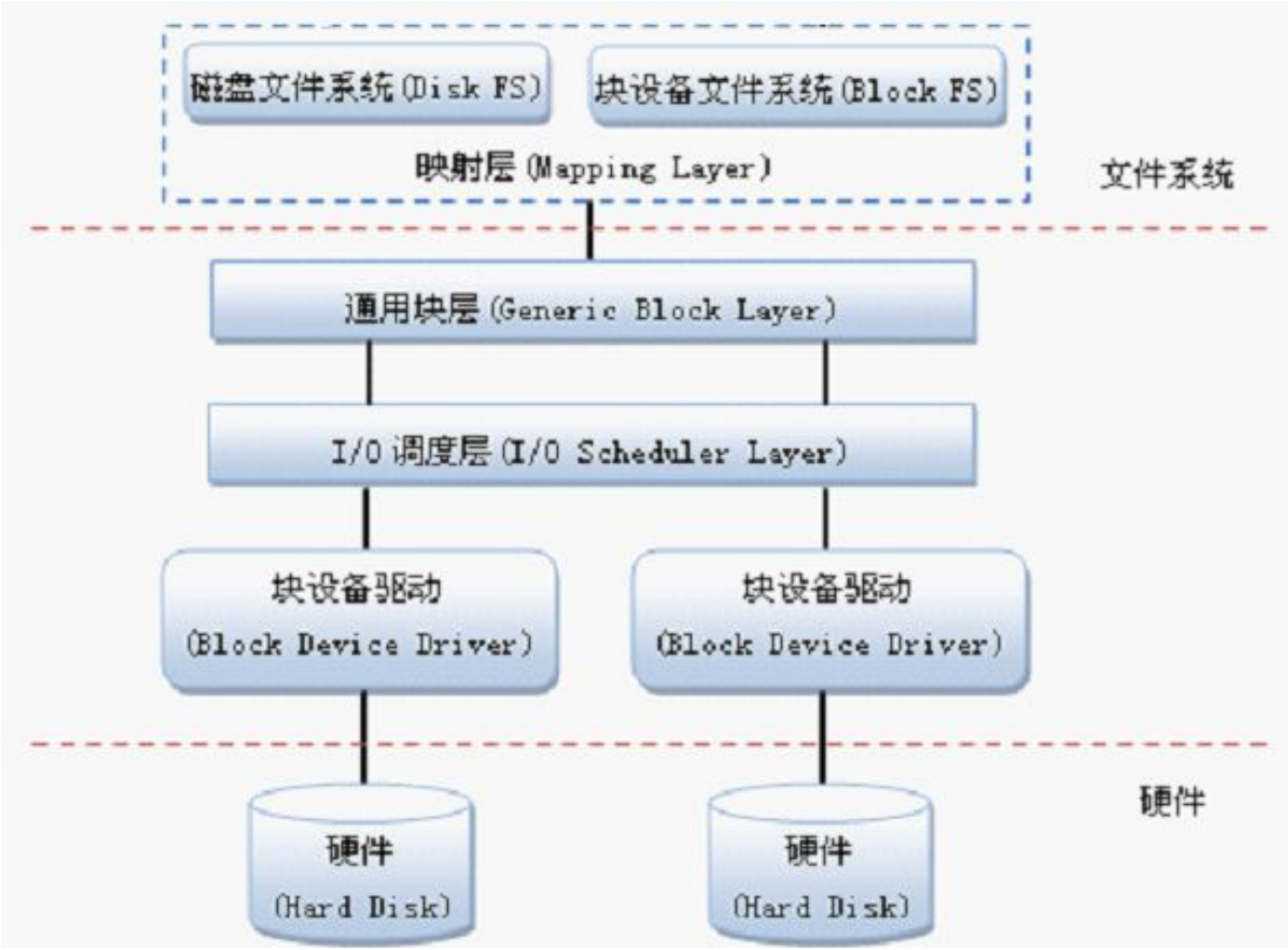
    块设备硬件对数据管理的基本单位。通常，1个扇区的大小为512byte。
  - 块(Blocks)：

    由Linux制定对内核或文件系统等数据处理的基本单位。是内核与块设备驱动数据交互的单位。
- 扇区、块与驱动
  - 块驱动是基于扇区(sector)来访问底层物理磁盘，基于块(block)与上层文件系统交互。
  - 块驱动由多个组件构成，具体构成由下节介绍。。。

大纲

- 概述
- 块设备组件构成
- 块设备的数据结构
- 块设备的注册
- 块设备的数据传输
- 使用块设备

块设备驱动程序框架



块设备驱动程序框架

- 通用块层
  - 主要完成块设备的核心功能，完成与文件系统的数据交互
  - 在Linux中，内核文件子系统通过统一接口与该层进行数据传输，在通用块层该数据用struct bio结构体表示
  - 通用块层提供了一个结构体struct request，该结构体包含了多个bio请求块，以链表的方式进行管理
  - 此外该层维护了一个结构体request_queue（请求队列），请求队列作为一个容器，用来缓存多个request数据请求，及request_queue管理request数据

块设备驱动程序框架

- I/O调度层
  - 主要对块设备请求队列中的请求进行调度，最大程度优化硬件操作的性能
  - 该层的I/O调度器（电梯）操作bio数据并对请求进行合并、调整顺序等操作，减少磁头移动的距离
  - 该层包含了多个I/O调度器，如No-op、Anticipatory、Deadline等，内核block目录中的noop-iosched.c、as-iosched.c等实现了上述算法
  - I/O调度层的结果就是生成了优化后的reques数据，该数据被添加到request_queue之中

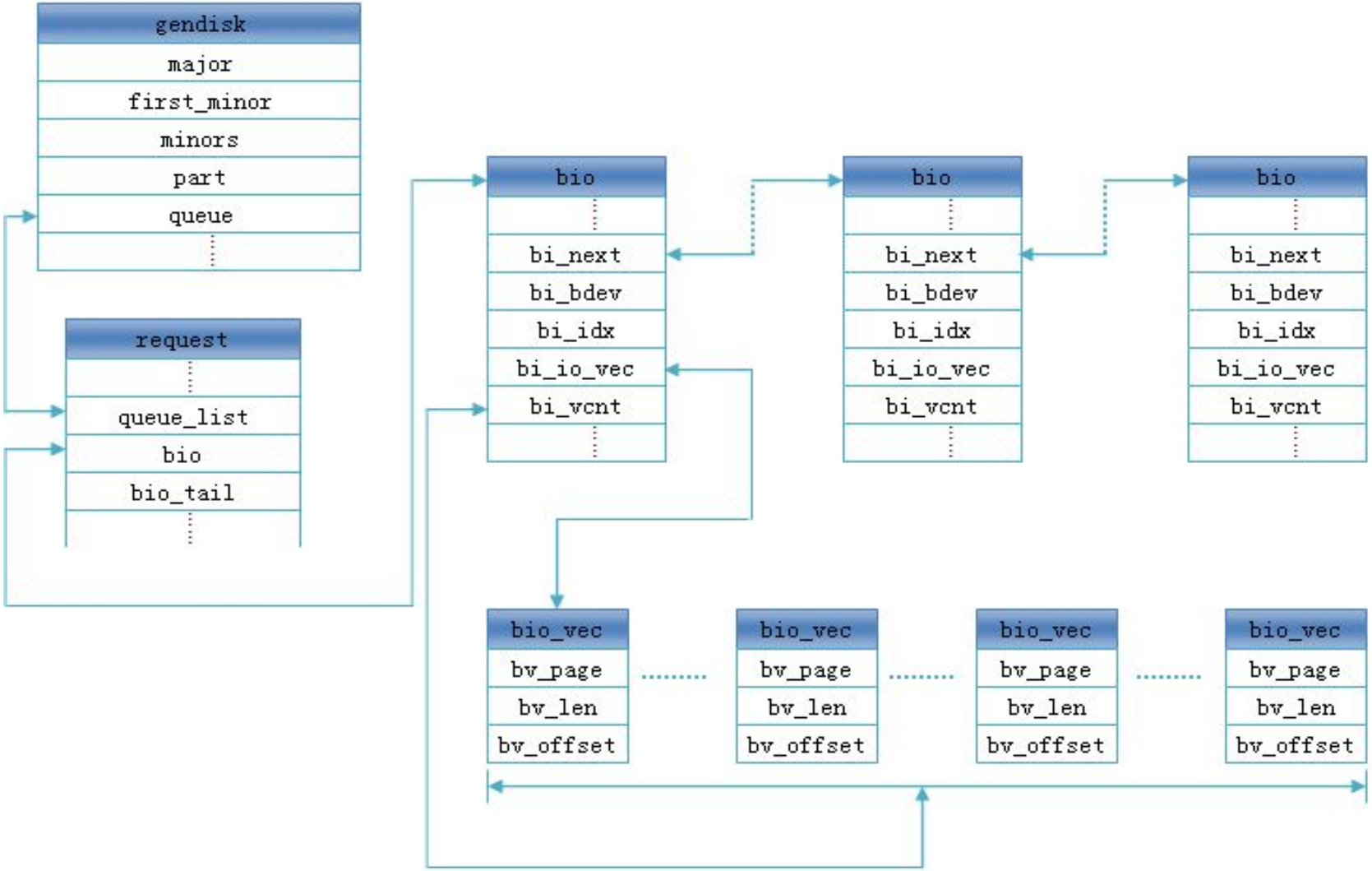
块设备驱动程序框架

- 块设备驱动层
  - 该层完成具体的硬件操作，实现对request数据的处理
  - 块设备驱动层是我们学习的重点，我们写块驱动的任务量也主要集中在这一层
  - 块设备驱动层可以用一个gendisk结构体表示，该结构体包涵了整个块设备的信息，如设备名、主从设备号、扇区大小扇区数等等，并提供了标准的操作接口函数（类似字符中的file_operations），前面提到的request_queue结构体也定义在该结构体中
  - gendisk结构体代表了一个具体的磁盘或者磁盘上的一个具体的分区
  - 实现块设备驱动的本质就是构造并注册gendisk结构体

大纲

- 概述
- 块设备组件构成
- 块设备的数据结构
- 块设备的注册
- 块设备的数据传输
- 使用块设备

块设备的数据结构



块设备的数据结构

- gendisk结构体
  - 内核使用gendisk结构来表示一个独立的磁盘设备与分区
  - 该结构体记录块设备的信息，定义在#include<linux/genhd.h>头文件里。
  - 在gendisk结构中的许多成员必须由内核接口进行初始化。
  - 结构体信息如下：

块设备的数据结构

```
struct gendisk {
    int major;              //主分区号
    int first_minor;        //起始从设备号
    int minors;             //支持的从设备号总数
    char disk_name[DISK_NAME_LEN];    //dev下设备名称
    char *(*devnode)(struct gendisk *gd, mode_t
    *mode);
    struct disk_part_tbl *part_tbl;
    struct hd_struct part0;
    //块设备操作函数接口
    const struct block_device_operations *fops;
    struct request_queue *queue;      //数据请求队列
```

块设备的数据结构

```
    void *private_data;          //可保存自定义的私有数据
    int flags;
    struct device *driverfs_dev;
    struct kobject *slave_dir;
    struct timer_rand_state *random;
    atomic_t sync_io;
    struct work_struct async_notify;
    #ifdef  CONFIG_BLK_DEV_INTEGRITY
    struct blk_integrity *integrity;
    #endif
    int node_id;
};
```

块设备的数据结构

➢block_device_operations结构体

```
struct block_device_operations {
        int (*open) (struct block_device *, fmode_t);
        int (*release) (struct gendisk *, fmode_t);
        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 *);
        int (*media_changed) (struct gendisk *);
        void (*unlock_native_capacity) (struct gendisk *);
        int (*revalidate_disk) (struct gendisk *);
        int (*getgeo)(struct block_device *, struct hd_geometry *);
        /* this callback is with swap_lock and sometimes page table lock held */
        void (*swap_slot_free_notify) (struct block_device *, unsigned long);
        struct module *owner;
};
```

块设备的数据结构

- open()
  - 完成对设备的打开操作，例如光驱、软驱等仓门的打开
- release()
  - 完成对设备的关闭操作，同上
- ioctl()
  - 提供对设备的控制，高层块设备代码处理了绝多数控制，具体驱动中我们只需实现设备相关的控制
- getgeo()
  - 获取块设备的物理参数，通过hd_geometry结构体返回磁盘的磁头、柱面、每磁道的扇区等物理参数
  - 结构体格式如下：

块设备的数据结构

```
struct hd_geometry {
      unsigned char heads;
      unsigned char sectors;
      unsigned short cylinders;
      unsigned long start;
};
```

- check_media_change()、revalidate()
  - 用于支持可移动存储设备(如floppy disk、CD-ROM)
  - check_media_change用于检查自从上次检查以来设备是否发生变化
  - revalidate在检查块设备被替换之后重新初始化驱动状态

大纲

- 概述
- 块设备组件构成
- 块设备的数据结构
- 块设备的注册
- 块设备的数据传输
- 使用块设备

块设备驱动程序框架

- 块设备驱动注册步骤（模块初始化函数）
  1. 注册块设备(获取主设备号)
  2. 定义gendisk结构体（采用内核接口函数）
  3. 初始化gendisk结构体（主设备号、从设备号、请求队列、设备名、设备操作结构体、数据处理函数等等）
  4. 注册gendisk结构体

块设备驱动程序框架

- 块设备驱动卸载步骤（模块卸载函数）
  1. 注销gendisk结构体
  2. 删除gendisk结构体
  3. 释放占用的系统资源（如中断、kmalloc空间信号量、自旋锁等等）
  4. 注销块设备

块设备驱动程序框架

- 驱动程序注册与清除函数
  - int register_blkdev(unsigned int major, const char *name);
  - int unregister_blkdev(unsigned int major, const char *name);

块设备驱动程序框架

- gendisk结构体分配和删除
  - struct gendisk *alloc_disk(int minors)
  - void put_disk(struct gendisk *disk)
- gendisk结构体初始化
  - 该部分主要任务是设置主从设备号、设备名、扇区信息以及request_queue的配置（有关request_queue的内容下一节进行讲解）

块设备驱动程序框架

- 块设备注册与注销
  - 注册：void add_disk(struct gendisk *disk)
  - 注销：void del_gendisk(struct gendisk *gp)

大纲

- 概述
- 块设备驱动程序框架
- 块设备的数据结构
- 块设备的注册
- 块设备的数据传输
- 使用块设备

块设备的数据传输

```
static int __init xxx_module_init(void)
{
    MAJOR_NR = register_blkdev(MAJOR_NR, DRIVER_NAME);
    if(MAJOR_NR < 0){
        PRINTK("register block device fail!\n");
        return MAJOR_NR;
    }

    gdisk = alloc_disk(1);          //申请gendisk结构体
    ....                            //结构体初始化
    /*初始化gendisk中的请求队列，并指定request的处理函数vrd_request,
    该处理函数由块设备驱动层实现，完成具体的数据请求到硬件的传输*/
    gdisk->queue = blk_init_queue(vrd_request,&splock);
    ....                            //结构体初始化

    add_disk(gdisk);                //注册块设备结构
}
```

块设备的数据传输

- 请求队列初始化函数：
  - struct request_queue* blk_init_queue(request_fn_proc *rfn, spinlock_t *lock)
    - 返回值：已初始化好的请求队列
    - rfn：绑定请求处理函数的指针
    - lock：自旋锁变量，给块组件使用
- 编写自己的请求处理函数
  - void vrd_request(struct request_queue *q)
    - 参数q：由内核传递
    - 在该函数中我们需要亲自完成request_queue中的每一个request数据请求

块设备的数据传输

➢ 请求处理函数处理流程

```
void vrd_request(struct request_queue *q)
{
    struct request *req;
    ....
    req = blk_fetch_request(q);              //提取一个request请求
    while(req){
        start = blk_rq_pos(req);             //获取读写操作的起始扇区
        size = blk_rq_cur_bytes(req);        //获取数据读写的大小
        ....
        switch(rq_data_dir(req)){            //计算数据请求的方向
            case READ:
                ....          //读处理函数
            break;
            case WRITE:
                ....          //写处理函数
            break;
        }
        if(!__blk_end_request_cur(req,0))    //通知内核请求处理结束
            req = blk_fetch_request(q);      //获取下一个request请求
    }
}
```

大纲

- 概述
- 块设备驱动程序框架
- 块设备的数据结构
- 块设备的注册
- 块设备的数据传输
- 使用块设备

使用块设备

- 通常，我们需要使用块设备上的文件系统对块设备进行操作
  1. 编译、插入块设备驱动，/dev下生成相应设备文件
  2. 格式化块设备#mkfs.vfat /dev/ramdiskX
  3. 挂载块设备到指定目录下
  - mount [-t vfat] 驱动程序节点 挂载路径
  - 以自己实现的ramdisk驱动为例：
    - mount -t vfat /dev/ramdiskX /mnt

凌阳教育官方微信：Sunplusedu

Tel: 400-705-9680 , BBS: www.51develop.net , QQ群：241275518