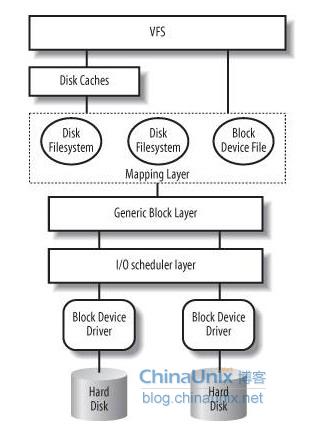
# Linux驱动\_块设备

## 1.基本概念

块设备（blockdevice）是一种具有一定结构的随机存取设备，对这种设备的读写是按块进行的，他使用缓冲区来存放暂时的数据，待条件成熟后，从缓存一次性写入设备或者从设备一次性读到缓冲区。

[](http://blog.chinaunix.net/attachment/201209/16/27664726_1347785002hS7r.jpg)

(1)当一个进程被Read时，内核会通过VFS层去读取要读的文件块有没有被cache了，这个cache由一个buffer\_head结构读取。如果要读取的文件块还没有被cache，则就要从文件系统中去读取，通过一个address\_space结构来引用，如果调用文件系统读取函数去读取一个扇区的数据。当它从磁盘读出数据时，将数据页连入到cache中，当下一次在读取时，就不需要从磁盘去读取了。Read完后将请求初始化成一个bio结构，并提交给通用块层。

(2)它通过submit\_bio()去完成，通用层在调用相应的设备IO调度器，这个调度器的调度算法，将这个bio合并到已经存在的request中，或者创建一个新的request，并将创建的插入到请求队列中，最后就剩下块设备驱动层来完成后面的所有工作。

## 2.简单的块设备驱动—ramblock

### 2.1 ramblock\_init

定义gendisk结构体与request\_queue请求队列结构体。

static struct gendisk **\***ramblock\_disk**;**

static request\_queue\_t **\***ramblock\_queue**;**

static int major**;**

static DEFINE\_SPINLOCK**(**ramblock\_lock**);**

#define RAMBLOCK\_SIZE (1024\*1024)

static unsigned char **\***ramblock\_buf**;**

gendisk结构体，主要是用于定义与内核，硬件有关的一些重要信息，还有就是，告诉内核定义请求队列的结构体以及操作函数的结构体。

请求队列：主要是提供读写能力，实现读写请求的存储，然后自己调用do\_ramblock\_request函数来实现读写操作。

static int ramblock\_init**(**void**)**

**{**

/\* 1. 分配一个gendisk结构体 \*/

ramblock\_disk **=** alloc\_disk**(**16**);** /\* 次设备号个数: 分区个数+1 \*/

/\* 2. 设置 \*/

/\* 2.1 分配/设置队列: 提供读写能力 \*/

ramblock\_queue **=** blk\_init\_queue**(**do\_ramblock\_request**,** **&**ramblock\_lock**);**

ramblock\_disk**->**queue **=** ramblock\_queue**;**

/\* 2.2 设置其他属性: 比如容量 \*/

major **=** register\_blkdev**(**0**,** "ramblock"**);** /\* cat /proc/devices \*/

ramblock\_disk**->**major **=** major**;**

ramblock\_disk**->**first\_minor **=** 0**;**

sprintf**(**ramblock\_disk**->**disk\_name**,** "ramblock"**);**

ramblock\_disk**->**fops **=** **&**ramblock\_fops**;**

set\_capacity**(**ramblock\_disk**,** RAMBLOCK\_SIZE **/** 512**);**

/\* 3. 硬件相关操作 \*/

ramblock\_buf **=** kzalloc**(**RAMBLOCK\_SIZE**,** GFP\_KERNEL**);**

/\* 4. 注册 \*/

add\_disk**(**ramblock\_disk**);**

**return** 0**;**

**}**

init函数主要完成了如下功能：

(1)分般gendisk结构体，并且设备此设备个数为16个

(2)分配、初始化队列，并且指定队列读写函数do\_ramblock\_request函数。

(3)设置以下虚拟块设备的一些属性，包括主设备号，次设备号，名字，操作函数，队列，设备容量等。

(4)申请内存空间。

(6)最后就是注册gendisk结构体。

### 2.2 do\_ramblock\_request

一些都准备就绪之后，当用户对虚拟块设备发出请求时，系统会调用读写函数do\_ramblock\_request来实现读写功能。

static void do\_ramblock\_request**(**request\_queue\_t **\*** q**)**

**{**

static int r\_cnt **=** 0**;** //读次数

static int w\_cnt **=** 0**;** //写次数

struct request **\***req**;**

//获取I/O请求队列中下一个请求

**while** **((**req **=** elv\_next\_request**(**q**))** **!=** **NULL)** **{**

/\* 数据传输三要素: 源,目的,长度 \*/

/\* 源/目的: \*/

unsigned long offset **=** req**->**sector **\*** 512**;**

/\* 目的/源: \*/

// req->buffer

/\* 长度: \*/

unsigned long len **=** req**->**current\_nr\_sectors **\*** 512**;**

//读数据

**if** **(**rq\_data\_dir**(**req**)** **==** READ**)**

**{**

printk**(**"do\_ramblock\_request read %d\n"**,** **++**r\_cnt**);**

//读ramblock\_buf中对应的数据

memcpy**(**req**->**buffer**,** ramblock\_buf**+**offset**,** len**);**

**}**

**else**

**{**

printk**(**"do\_ramblock\_request write %d\n"**,** **++**w\_cnt**);**

//把数据写到ramblock\_buf对应的地方

memcpy**(**ramblock\_buf**+**offset**,** req**->**buffer**,** len**);**

**}**

end\_request**(**req**,** 1**);**

**}**

**}**

### 2.3 ramblock\_getgeo

如果要使用fdisk分区，需要提供getgeo操作函数。

static int ramblock\_getgeo**(**struct block\_device **\***bdev**,** struct hd\_geometry **\***geo**)**

**{**

/\* 容量=heads\*cylinders\*sectors\*512 \*/

geo**->**heads **=** 2**;**

geo**->**cylinders **=** 32**;**

geo**->**sectors **=** RAMBLOCK\_SIZE**/**2**/**32**/**512**;**

**return** 0**;**

**}**

static struct block\_device\_operations ramblock\_fops **=** **{**

**.**owner **=** THIS\_MODULE**,**

**.**getgeo **=** ramblock\_getgeo**,**

**};**

如果没有getgeo操作函数，无法进行分区示语为unknown valus for cylinders不知道扇区的值，所以，为了能够实现用fdisk进行分区，我们就必须在程序中模拟伪装扇区，利用block\_device\_opreations结构体中的getgeo函数来人工的造一些扇区信息给系统看。

### 2.4 测试结果

insmod后可以查看到设备已经存在了。

# cat /proc/devices

Block devices**:**

1 ramdisk

7 loop

8 sd

31 mtdblock

254 ramblock

# ls -l /dev/ramblock

brw**-**rw**----** 1 0 0 254**,** 0 Jan 1 00**:**01 **/**dev**/**ramblock

下面测试下挂载操作。

//格式化

# mkdosfs /dev/ramblock

mkdosfs 2.11 **(**12 Mar 2005**)**

//挂载

# mount /dev/ramblock /tmp/ram

//ram下创建文件

# touch 1.txt

//卸载

# umount ram

//检查ram下的文件，不存在了

# ls ram/

//重新挂载

# mount /dev/ramblock /tmp/ram

//重新查看

# ls ram/

1.txt

测试下fdisk操作。

# fdisk /dev/ram

Command **(**m **for** help**):** n

Command action

e extended

p primary partition **(**1**-**4**)**

p

Partition number **(**1**-**4**):** 1

First cylinder **(**1**-**32**,** **default** 1**):** 1

Last cylinder **or** **+**size **or** **+**sizeM **or** **+**sizeK **(**1**-**32**,** **default** 32**):** 16

Command **(**m **for** help**):** n

Command action

e extended

p primary partition **(**1**-**4**)**

p

Partition number **(**1**-**4**):** 2

First cylinder **(**17**-**32**,** **default** 17**):** 17

Last cylinder **or** **+**size **or** **+**sizeM **or** **+**sizeK **(**17**-**32**,** **default** 32**):** Using **default** value 32

Command **(**m **for** help**):** w

The partition table has been altered**!**

ramblock**:** ramblock1 ramblock2

Calling ioctl**()** to re**-**read partition table

# ls -l /dev/ramblock\*

brw**-**rw**----** 1 0 0 254**,** 0 Jan 1 00**:**25 **/**dev**/**ramblock

brw**-**rw**----** 1 0 0 254**,** 1 Jan 1 00**:**25 **/**dev**/**ramblock1

brw**-**rw**----** 1 0 0 254**,** 2 Jan 1 00**:**25 **/**dev**/**ramblock2

## 3.Linux中的内存文件系统

在[Linux](http://lib.csdn.net/base/linux)中可以将一部分内存mount为分区来使用，通常称之为RamDisk，分为：Ramdisk, ramfs, tmpfs。

**(1)Ramdisk**

在编译内核时须将Device Drivers -->> Block devices -->> Ramdisk support 支持选上，它下面还有两个选项：第一个设定Ramdisk个数，默认16个；第二个是设定Ramdisk的大小，默认是4096k。

首先查看一下可用的RamDisk，使用ls /dev/ram\*

然后对/dev/ram0创建文件系统，运行mke2fs /dev/ram0

最后挂载/dev/ram0，运行mount /dev/ram /mnt/test

**(2)ramfs**

Ramfs顾名思义是内存文件系统，它处于虚拟文件系统（VFS）层，而不像ramdisk那样基于虚拟在内存中的其他文件系统(ex2fs)。

因而，它无需格式化，可以创建多个，只要内存足够，在创建时可以指定其最大能使用的内存大小。

在编译内核时须将File systems -->> pseudo filesystems -->>  Virtual memory file system support支持选上。

# mount -t ramfs none /testRAM

缺省情况下，Ramfs被限制最多可使用内存大小的一半。可以通过maxsize（以kbyte为单位）选项来改变。

# mount -t ramfs none /testRAM -o maxsize=2000 (创建了一个限定最大使用内存为2M的ramdisk)

**(3)Tmpfs**

是一个虚拟内存文件系统，它不同于传统的用块设备形式来实现的Ramdisk，也不同于针对物理内存的Ramfs。

Tmpfs可以使用物理内存，也可以使用交换分区。在Linux内核中，虚拟内存资源由物理内存（RAM）和交换分区组成，这些资源是由内核中的虚拟内存子系统来负责分配和管理。

Tmpfs向虚拟内存子系统请求页来存储文件，它同Linux的其它请求页的部分一样，不知道分配给自己的页是在内存中还是在交换分区中。同Ramfs一样，其大小也不是固定的，而是随着所需要的空间而动态的增减。

在编译内核时须将File systems -->> pseudo filesystems -->>  Virtual memory file system support支持选上。

# mkdir -p /mnt/tmpfs

# mount tmpfs /mnt/tmpfs -t tmpfs

同样可以在加载时指定tmpfs文件系统大小的最大限制:

# mount tmpfs /mnt/tmpfs -t tmpfs -o size=32m