## Linux下编写一个ramdisk块设备驱动及建立一个ext4文件系统

2019-01-11

版权声明：本文为博主原创文章，遵循[CC 4.0 BY-SA](http://creativecommons.org/licenses/by-sa/4.0/)版权协议，转载请附上原文出处链接和本声明。

本文链接：<https://blog.csdn.net/huang_165/article/details/86302757>

[版权声明：本文为博主原创文章，转载请注明出处：](https://blog.csdn.net/huang_165/article/details/85340002)<https://blog.csdn.net/huang_165/article/details/86302757>

本博文参考ldd3实现一个内存虚拟磁盘，它的通过块设备驱动申请一些内存用作“磁盘容量”并将该磁盘格式化为ext4文件系统，实现和正常磁盘无异的文件创建、存放。

源码地址:<https://github.com/Mr-jinfa/rk3399-project/tree/master/drv/ramdisk>

     块设备是与字符设备并列的概念，这两类设备在Linux 中的驱动结构有较大差异，总体而言，块设备驱动比字符设备驱动要复杂得多，在I/O 操作上也表现出极大的不同。缓冲、I/O 调度、请求队列等都是与块设备驱动相关的概念。

### 块设备的I/O 操作特点：

1）块设备只能以数据块为单位接收输入和返回输出，而字符设备则以字节为单位。  
2）块设备对于I/O 请求有对应的缓冲区，因此它们可以选择以什么顺序进行响应。对于存储设备而言，调整读写的顺序作用巨大，因为在读写连续的扇区的存储速度比分离的扇区更快。  
3）字符设备只能被顺序读写，而块设备可以随机访问。顺序地组织块设备的访问可以提高性能，对扇区1、10、3、2 的请求被调整为对扇区1、2、3、10 的请求将大大提升数据读取速率。

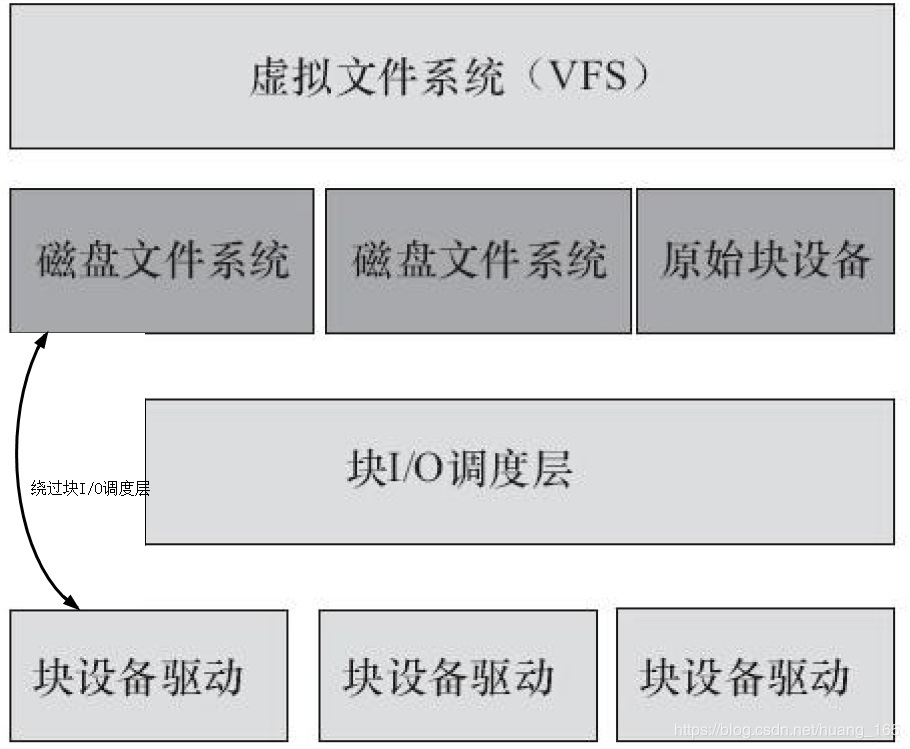
总的来说，字符设备适应IO流不大的情况，块设备适应IO流连续、繁忙的情况。

      对于块设备驱动Linux有专门api来辅助我们实现，我们只需在这些api基础上实现块设备驱动达到管理“存储硬件”就好了。

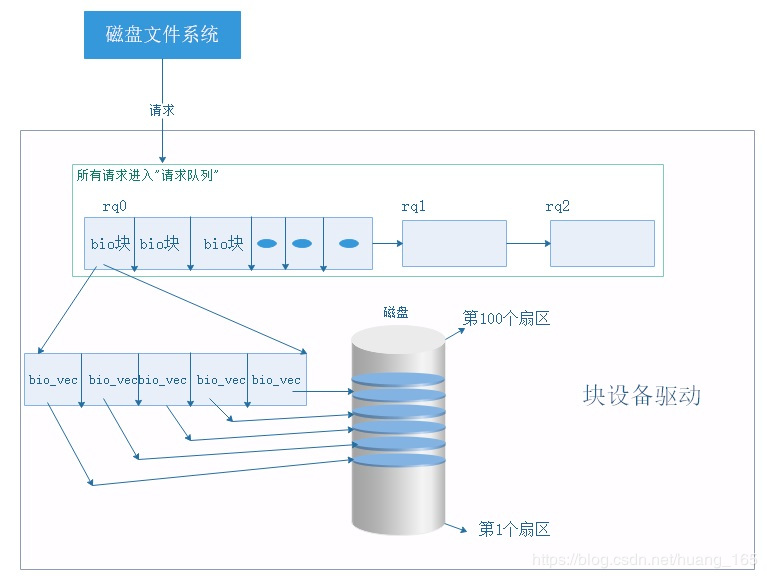
设备驱动的本质就是管理硬件、提升硬件性能。

      对于ramdisk 这种完全随机访问的非机械设备，并不需要进行复杂的I/O 调度。所以，我们可以绕开I/O调度层。

### Linux文件系统和块设备--ramdisk关系：



### 文件系统到ramdisk磁盘的数据流--块设备数据操作流程图:



块设备满足文件系统的读写操作的"请求处理函数"流程  
1：使用blk\_fetch\_request循环地从请求队列中提取一个个请求并处理。  
2：使用\_\_rq\_for\_each\_bio循环地从请求中提取一个个bio并处理。  
3：使用bio\_for\_each\_segment循环地从bio中提取一个个segment(扇区)并处理。  
3.1：使用\_\_bio\_kmap\_atomic获得一个扇区的缓冲区。  
3.2：用上面的缓冲区作为中介处理扇区中需要操作的数据。  
3.3：释放该缓冲区  
4：一个请求完成文件系统报\_\_blk\_end\_request\_all(req, 0);

### 使用块设备&调试信息：

1. insmod ramblock\_drv.ko

2. # ls -l /dev/ramblock

brw-rw---- 1 root disk 252, 0 Jan 18 08:53 /dev/ramblock

3. cat /proc/devices

Block devices:

179 mmc

252 sbull //找到我们的驱动

253 nvme

4. mkfs.ext4 /dev/ramblock

mke2fs 1.43.9 (8-Feb-2018)

[ 437.834369] do\_ramblock\_request read 1

...

Filesystem too small for a journal

[ 437.844949] do\_ramblock\_request read 17

Creating filesystem with 512 1k blocks and 64 inodes

Allocating group tables: done

Writing inode tables: done

[ 437.847527] do\_ramblock\_request read 18

...

[ 437.849576] do\_ramblock\_request read 22

Writing superblocks and filesystem accounting information: [ 437.850436] do\_ramblock\_request read 23

[ 437.851221] do\_ramblock\_request write 1

...

[ 437.859029] do\_ramblock\_request write 23

[ 437.859552] do\_ramblock\_request write 24

done

5. mkdir tmp1 && mount /dev/ramblock tmp1/

[ 550.294217] do\_ramblock\_request read 24

[ 550.294745] EXT4-fs (ramblock): couldn't mount as ext3 due to feature incompatibilities

[ 550.295719] do\_ramblock\_request read 25

[ 550.296140] EXT4-fs (ramblock): couldn't mount as ext2 due to feature incompatibilities

[ 550.297055] do\_ramblock\_request read 26

...

[ 550.299880] do\_ramblock\_request write 25

[ 550.301078] EXT4-fs (ramblock): mounted filesystem without journal. Opts: (null)

6. mount

/dev/ramblock on /run/tmp1 type ext4 (rw,relatime,block\_validity,delalloc,barrier,user\_xattr,acl)

7. cd tmp1/lost\+found/ && echo 123 >> hello.txt

[ 641.555232] do\_ramblock\_request read 31

....

[ 641.562480] do\_ramblock\_request read 43

...

[ 645.422579] do\_ramblock\_request write 35

...

8. 退出到根目录 && umount tmp1/

[ 777.374609] do\_ramblock\_request write 38

9. 再mount

#mount /dev/ramblock tmp1/

10. 查看hello.txt是否还在

#cat tmp1/lost\+found/hello.txt

123

具体程序请看提供的源码，源码的注释还是很清楚的。