Veeamsnap

1. Linux 块设备IO流程

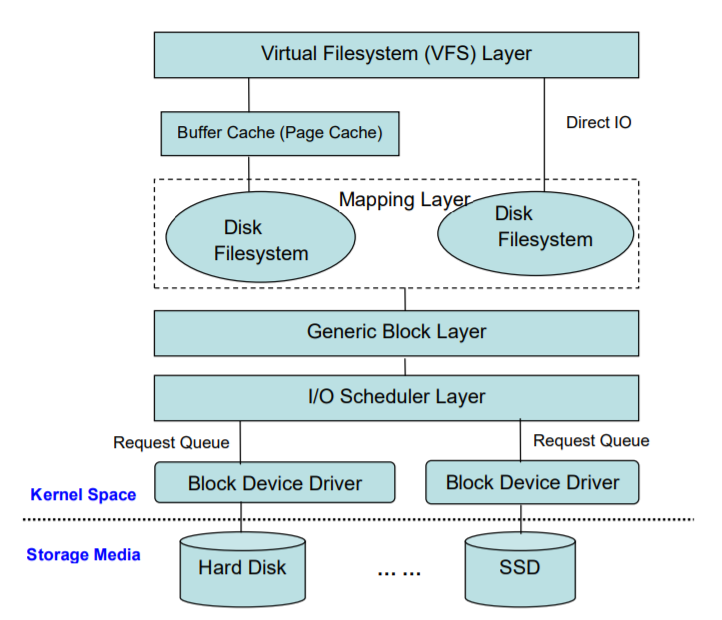


图1 Linux内核块设备I/O流程

假设一个进程使用系统调用read（）读取磁盘上的文件。下面步骤是内核响应进程读请求的步骤；

（1）系统调用read（）会触发相应的VFS（Virtual Filesystem Switch）函数，传递的参数有文件描述符和文件偏移量。

（2）VFS确定请求的数据是否已经在内存缓冲区中；若数据不在内存中，确定如何执行读操作。

（3）假设内核必须从块设备上读取数据，这样内核就必须确定数据在物理设备上的位置。这由映射层（Mapping Layer）来完成。

（4）此时内核通过通用块设备层（Generic Block Layer）在块设备上执行读操作，启动I/O操作，传输请求的数据。

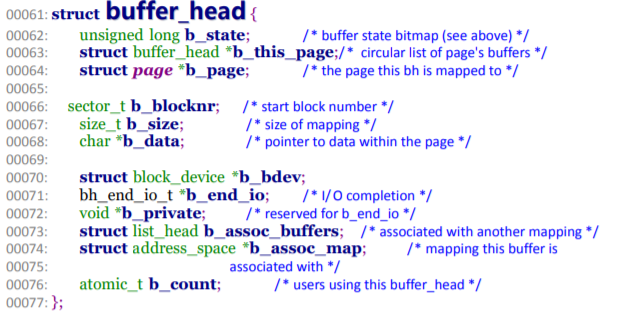
（5）在通用块设备层之下是I/O调度层（I/O Scheduler Layer），根据内核的调度策略，对等待的I/O等待队列排序。

（6）最后，块设备驱动（Block Device Driver）通过向磁盘控制器发送相应的命令，执行真正的数据传输。

1. 通用块设备层
2. 缓冲区和缓冲区头

当一个块被调入内存时（也就是，在读入或等待写出时），它要存储在一个缓冲区中。每个缓冲区都有一个块对应，它相当于是磁盘块在内存中的表示。前面提到过，块包含一个或多个扇区，但大小不能超过一个页面，所以一个页面可以容纳一个或多个内存中的块。由于内核在处理数据时，需要一些相关的控制信息（比如块属于哪一个块设备，块对应于哪一个缓冲区等），所以每个缓冲区都有一个对应的描述符。

每个块都要有自己的块缓冲区（block buffer），内核使用内存上的这块缓冲区来保存块数据。当内核从块设备上读取一个块时，就用从硬件上读取的数据填充块缓冲区；同理，当内核向块设备写数据时，就用块缓冲区中的数据写到块设备上。每个缓冲区都有一个缓冲区头（buffer head），描述类型为buffer\_head，这个数据结构里包含了内核处理缓冲区所需要的信息；于是，在每个缓冲区上操作之前，内核都要先检查它的缓冲区头。buffer\_head的定义在文件include/linux/buffer\_head.h中。

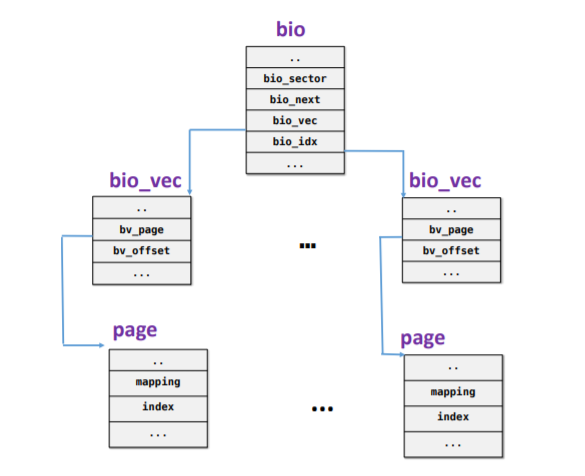


1. bio结构体

I/O操作的基本容器由bio结构体表示，它定义在文件include/linux/bio.h中。该结构体代表了正在活动的以段（segment）链表形式组织的块I/O操作。一个段是一小块连续的内存缓冲区。

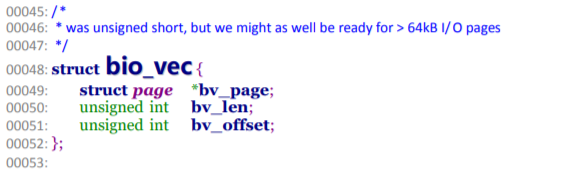
bio结构体中几个成员变量的含义：

|  |  |
| --- | --- |
| bi\_sector | 磁盘上相关的扇区 |
| bi\_next | 请求链表 |
| bi\_bdev | 相关的块设备 |
| bi\_rw | 读还是写 |
| bi\_vcnt | bio\_vecs偏移量 |
| bi\_idx | bi\_io\_vec的当前索引 |
| bi\_io\_vec | bio\_vec链表 |
| bi\_end\_io | I/O完成方法 |
| bi\_cnt | 使用计数 |



bio结构体、bio\_vec结构体和page之间的关系

bi\_io\_vecs指向一个bio\_vec结构体数组，该结构体链表包含了一个特定I/O操作所需要使用到的所有段（segment）。每个bio\_vec结构都是一个形式为<page, offset, len>的向量，它描述的是一个特定的段：段所在的物理页、块在物理页中的偏移量、从给定偏移量开始的块长度。整个bio\_io\_vec结构体数组表示了一个完整的缓冲区。



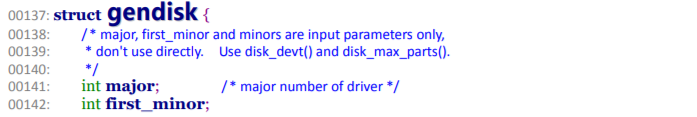
在每个给定的块I/O操作中，bi\_vcnt域用来描述bi\_io\_vec所指向的bio\_vec数组中的向量数目。当块I/O操作执行完毕后，bi\_idx指向数组的当前索引。

总之，每个块I/O请求都通过一个bio结构体表示。每个请求包含一个或多个块，这些块存储在bio\_vec结构体数组中。这些结构体描述了每个片段在物理页中的实际位置，并且像向量一样被组织在一起。I/O操作的第一个段（segment）由bi\_io\_vec结构体所指向，其他的段在其后依次放置，共有bi\_vcnt个段。当块I/O层开始执行请求，需要使用各个片段时，bi\_idx就会不断更新，从而总指向当前片段。bi\_idx指向数组的当前bio\_vec段，块I/O层通过它可以跟踪块I/O操作的完成进度。

1. 通用块设备层对请求的处理
2. 磁盘和磁盘分区的表示

磁盘（disk）是一个逻辑块设备，物理设备在内核中的表示，它由通用块设备层来处理。通常一个磁盘对应一个硬件块设备，如硬盘、软盘和光盘等。然而，一个磁盘也可以是多个物理磁盘分区组成的虚拟设备，也可以是内存上某些页组成。通过通用块设备层，内核中的上层部分可以以相同方式访问物理磁盘设备。

内核使用gendisk结构，定义在include/linux/genhd.h中，来表示一个独立的磁盘设备。实际上内核还使用gendisk表示分区，但是驱动程序不需要了解这些。在gendisk结构中的许多成员必须由驱动程序进行初始化。



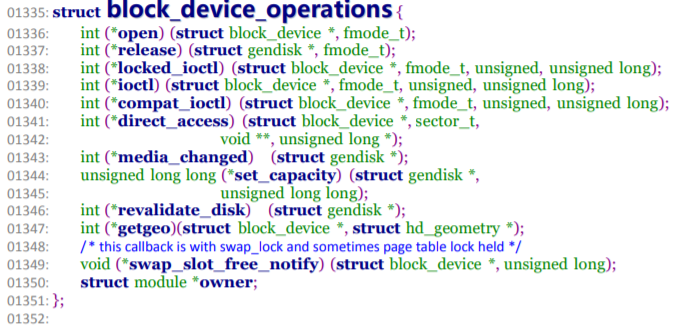


minors：磁盘使用这些成员描述设备号。一个驱动器至少使用一个设备号。如果驱动器是可被分区的（大多数情况下），用户将要为每个可能的分区都分配一个次设备号。minors的值常取16，可以包含15个分区。但是，某些磁盘驱动程序设置每个设备可使用64个次设备号；

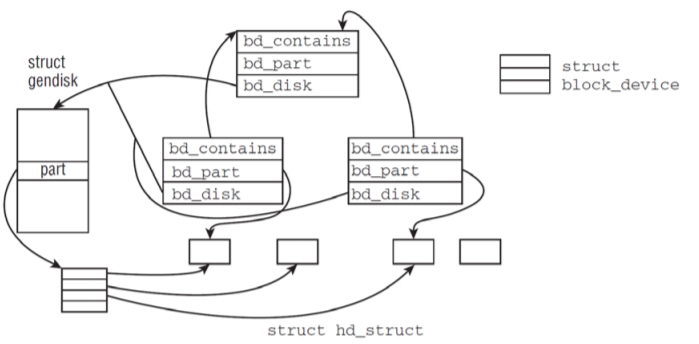
fops：设置前面描述的各种设备操作；

queue：内核使用该结构为设备管理I/O请求；

gendisk中的fops成员的数据结构类型为struct block\_device\_operations，其定义在include/linux/fs.h中，结构体中包括了对块设备操作的几个关键方法。



物理磁盘通常被分成多个逻辑分区。每个块设备文件可以表示一个整个物理磁盘或者其中的一个分区。如/dev/sda、/dev/sda1、/dev/sda2等。若一个物理磁盘有多个分区，则磁盘的布局保存在hd\_struct数据结构数组中，数组的地址由gendisk结构体中的part成员保存。hd\_struct数据结构的定义在文件include/linux/genhd.h中。

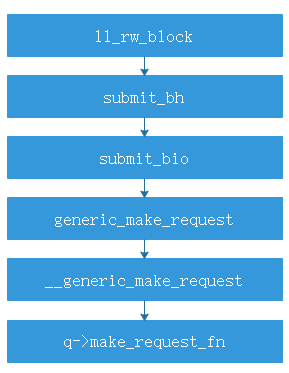


当内核在系统中发现一个新磁盘时，调用alloc\_disk（）分配相关数据结构，如gendisk，hd\_struct等，然后调用add\_disk（）将磁盘添加到系统中。注意：一旦调用了add\_disk，磁盘设备就被“激活“，表示可以使用，并随时会调用它们提供的方法。

1. 向通用块设备层发送请求

以向设备上写数据为例，即以write（）系统调用为描述对象。一个IO请求从

ll\_rw\_block（）开始，调用过程大体如下：



ll\_rw\_lock（）

ll\_rw\_block（）函数产生块设备请求；内核和设备驱动程序的很多地方都会调用这个函数。调用submit\_bh（）提交请求。

submit\_bh（）

submit\_bh函数的主要任务是为buffer head分配一个bio对象，并且对其进行初始化，然后将bio提交给对应的块设备对象。提交给块设备的行为其实就是让对应的块设备驱动程序对其进行处理。

submit\_bh（）函数假定缓冲区头已完全初始化，即b\_bdev，b\_blocknr和b\_size成员变量已正确设置，能够确定请求数据在磁盘上的具体块位置。它从缓冲区头中的内容来创建一个bio请求，然后调用submit\_bio（）

submit\_bio（）

submit\_bio（）的功能比较简单，主要就是设置bio->bi\_rw的值，即确定数据传输方向。若属于正常的读写数据，就在提交任务之前，更新I/O统计信息。然后调用generic\_make\_request（）。

generic\_make\_request（）

通用块设备层的主要入口是generic\_make\_request（），源码也在在文件driver/block/ll\_rw\_blk.c中。

函数主要功能为维护进程bio链表和对\_\_generic\_make\_request（）的直接封装调用。

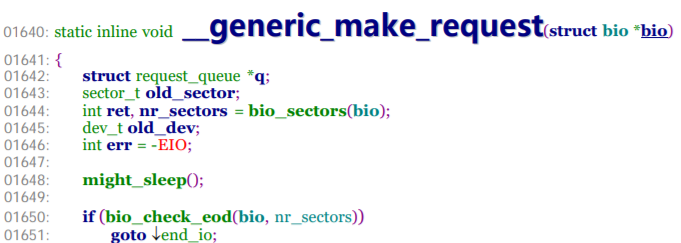


在每个进程的task\_struct中，都包含有两个变量struct bio \*bio\_list, \*\*bio\_tail，generic\_make\_request（）的主要工作就是用这两个变量，来维护当前待添加的bio链表。bio\_tail是一个二级指针，这个值最初是NULL，当有bio添加进来，bio\_tail将会指向bio->bi\_next(如果bio全都递交上去了，则bio\_tail将会指向bio\_list)，也就是说除了第一次调用外，其他每次递归调用generic\_make\_request（）函数都会出现bio\_tail不为NULL的情形，因此当bio\_tail不为NULL时，则只将bio添加到由bio\_list和bio\_tail维护的链表中，然后直接返回，加入到链表的结点generic\_make\_request会在后面处理,而不再次调用\_\_generic\_make\_request()，这样便防止了多重递归的产生。\_\_generic\_make\_request（）会执行\_\_make\_request（），直到最后\_\_make\_request函数返回0，才算是告诉generic\_make\_request无需再转发bio了，bio转发结束。

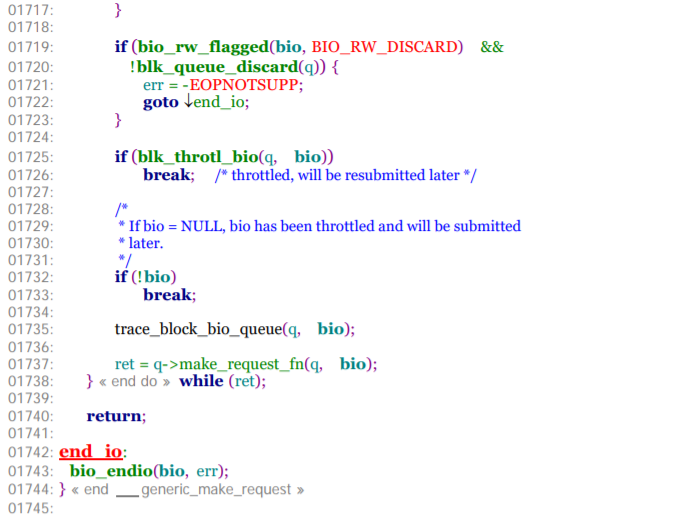
\_\_generic\_make\_request（）函数会调用块设备的make\_request\_fn方法。普通设备make\_request\_fn实现为内核\_\_make\_request函数，一些特殊设备会使用自己的make\_request\_fn，例如md和dm设备，可以统称此类设备为 virtual disk。

\_\_generic\_make\_request（）

\_\_generic\_make\_request（）功能：调用blk\_partition\_remap（）函数检查该块设备是否是一个磁盘分区，如果是个分区，则需要做一些转化。bio->bio\_sector的值是相对于分区的起始扇区号，这里需要转换成相对于整个磁盘的扇区号。然后再调用q->make\_request\_fn（）把bio请求插入到请求队列中。 q->make\_request\_fn（）把通用块层和I/O调度层连接起来，make\_request\_fn的值是在块设备驱动在加载时里调用blk\_init\_queue（）设置。







该函数主要完成以下工作：

（1）bio\_check\_eod（）检查bio->bio\_sector没有超过设备的最大扇区数。若bio->bio\_sector超过设备的最大扇区数，则设置bio->bi\_flags为BIO\_EOF，打印一条内核错误信息，然后调用bio\_endio（）（fs/bio.c）结束（1650行）。

（2）获取块设备的请求队列q（1666行），获取方法很简单，返回bdev->bd\_disk->queue即可。

（3）调用blk\_partion\_remap（）（/drivers/block/ll\_rw\_blk.c），检查该块设备是否是一个分区（即bio->bi\_dev与bio->bi\_dev->bd\_contains不相等）。若是一个分区，则获取从bio->bi\_dev中获取hd\_struct描述符（）（1692行）。

（4）调用q->make\_request\_fn方法，将bio请求插入请求队列q中。对于所有的块设备来说，q->make\_request\_fn方法就是\_\_make\_request（），

bio\_endio（）

bio\_endio（）更新bi\_size和bi\_sector的值，然后调用bio的bi\_end\_io方法。