# 背景

当内核开始一个读操作（比如进程发起一个read()系统调用），它首先会检查需要的数据是否在页高速缓存中。如果在，则放弃访问磁盘，而直接从内存中读取。这个行为称作缓存命中。如果数据没有在缓存中，称为缓存未命中，那么内核必须调度块I/O操作从磁盘去读取数据。然后内核将读来的数据放入页缓存中，于是任何后续相同的数据读取都可以命中缓存了。

注：这里说明了什么时候使用块I/O调度，即页高速缓存未命中的情况下，执行磁盘的块I/O操作。

# 概述

系统中能够随机（不需要按顺序）访问固定大小数据片（chunks）的硬件设备称作块设备，这些固定大小的数据片就称为块。最常见的块设备是硬盘，此外还有软盘驱动器、光驱和闪存等块设备。

另一种基本的设备类型是字符设备。字符设备按照字符流的方式被有序访问，像串口和键盘都属于字符设备。

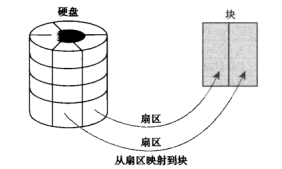
内核管理块设备要比管理字符设备细致得多，需要考虑的问题和完成的工作相对于字符设备来说要复杂很多。这是因为字符设备仅仅需要控制一个位置——当前位置，而块设备访问的位置必须能够在介质的不同区间前后移动。所以事实上内核不必提供一个专门的子系统来管理字符设备，但是对块设备的管理却必须要有一个专门的提供服务的子系统。

# 块设备

块设备中最小的可寻址单元是扇区。扇区大小一般是2的整数倍，而最常见的是512字节。扇区的大小是块设备的物理属性，扇区是所有块设备的基本单元——块设备无法对比它还小的单元进行寻址和操作，尽管许多块设备能够一次对多个扇区进行操作。

因为各种软件的用途不同，所以他们都会用到自己的最小逻辑可寻址单元——块。块是文件系统的一种抽象——只能基于块来访问文件系统。虽然物理磁盘寻址是按照扇区级进行的，但是内核执行的所有磁盘操作都是按照块进行的。由于扇区是设备的最小可寻址单元，所以块不能比扇区还小，只能整倍于扇区大小。另外，内核（对有扇区的硬件设备）还要求块大小是2的整数倍，而且不能超多一个页的长度。所以，对块大小的最终要求是，必须是扇区大小的2的整数倍，并且要小于页面大小。所以通常块大小是512字节、1KB或4KB。

扇区——设备的最小寻址单元，有时会称作“硬扇区”或“设备块”；同样，块——文件系统的最小寻址单元，有时会称作“文件块”或“I/O块”。



# 缓冲区/缓冲区头

当一个块被调入内存时（也就是说，在读入后或等待写出时），它要存储在一个缓冲区中。每个缓冲区与一个块对应，它相当于是磁盘块在内存中的表示。块包含一个或多个扇区，但是大小不能超多一个页面，所以一个页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，块对应于哪个缓冲区等），所以每一个缓冲区都有一个对应的描述符。该描述符用buffer\_head结构体表示，称作缓冲区头。

缓冲区头的目的在于描述磁盘块和物理内存缓冲区（在特定页面上的字节序列）之间的映射关系。这个结构体在内核中只扮演一个描述符的角色，说明从缓冲区到块的映射关系。

在2.6内核版本之前，缓冲区头作用比现在更重要。因为缓冲区头作为内核中的I/O操作单元，不仅仅描述了从磁盘块到物理内存的映射，而且还是所有块I/O操作的容器。可是，将缓冲区头作为I/O操作单元有两个弊端：

首先，缓冲区头是一个很大且不易控制的数据结构体（现在是缩减过的了），而且缓冲区头对数据的操作既不方便也不清晰。对于内核来说，它更倾向于操作页面结构，因为页面操作起来更为简便，同时效率也高。使用一个巨大的缓冲区头表示每一个独立的缓冲区（可能比页面小）效率低，所以在2.6版本中，许多I/O操作都是通过内核直接对页面或者地址空间操作来实现的，不再使用缓冲区头了。

其次，它仅能描述单个缓冲区，当作为所有I/O的容器使用时，缓冲区头会促使内核必然会造成不必要的负担和空间浪费。所以2.5开发版内核的主要目标就是为块I/O操作引入一种新型、灵活并且轻量级的容器，即bio结构体。

# bio结构体

# 请求队列

块设备将它们挂起的I/O请求保持在请求队列中，该队列由request\_queue结构体表示，包含一个双向请求链表以及相关控制信息。通过内核中向文件系统这样高层的代码将请求加入到队列中。请求队列只要不为空，队列对应的块设备驱动程序就会从队列头获取请求，然后将其送入对应的块设备上去。请求队列中的每一项都是一个单独的请求，有request结构体表示。

# I/O调度程序

如果简单地以内核产生请求的次序直接将请求发现块设备的话，性能肯定让人难以接受。磁盘寻址是整个计算机中最慢的操作之一，每一次寻址（定位硬盘磁头到特定块上的某个地址）需要花费不少时间。所以尽量缩短寻址时间无疑是提高系统性能的关键。

为了优化寻址操作，内核既不会简单地按请求接收次序，也不会立即将其提交给磁盘。相反，它会在提交前，先执行名为合并与排序的预操作，这种预操作可以极大地提高系统的整体性能。在内核中负责提交I/O请求的子系统称为I/O调度程度。

I/O调度程序将磁盘I/O资源分配给系统中所有挂起的块I/O请求。具体地说，这种资源分配是通过将请求队列中挂起的请求合并和排序来完成的。注意不要将I/O调度程序和进程调度程序混淆，进程调度程序的作用是将处理器资源分配给系统中的运行进程。进程调度程序和I/O调度程序都是将一个资源虚拟给多个对象，对进程调度程序来说，处理器被虚拟并被系统中的运行进程共享。这种虚拟提供给用户的就是多任务和分时操作系统，像Unix系统。相反，I/O调度程序虚拟块设备给多个磁盘请求，以便降低磁盘寻址时间，确保磁盘性能的最优化。

## 作用

I/O调度程序通过两种方法减少磁盘寻址时间：合并与排序。合并指将两个或多个请求结合成一个新请求。

## Linus电梯

## 最终期限I/O调度程序

最终期限（deadline）I/O调度程序是为了解决Linus电梯所带来的饥饿问题而提出的。由于减少磁盘寻址时间的考虑，对某个磁盘区域上的繁重操作，无疑会使得磁盘其他位置上的操作请求得不到运行机会。实际上，一个对磁盘同一位置操作的请求流可以造成较远位置的其他请求永远得不到运行机会，这是一种很不公平的饥饿现象。

在最后期限I/O调度程序中，每个请求都有一个超时时间。默认情况下，读请求的超时时间是500ms，写请求的超时时间是5s。最后期限I/O调度请求类似于Linus电梯，也以磁盘物理位置为次序维护请求队列，这个队列称为排序队列。

## 预测I/O调度程序

虽然最后期限I/O调度程序为降低读操作响应时间做了许多工作，但是它同时也降低了系统吞吐量。预测I/O调度程序的目标就是在保持良好的读操作的同时也能提供良好的全局吞吐量。

预测I/O调度的基础仍然是最后期限I/O调度程序，所以它们有很多相同之处。预测I/O调度程序也实现了三个队列（加上一个派发队列），并为每个请求设置了超时时间，这点与最后期限I/O调度程序一样。预测I/O调度程序主要的改进是它增加了预测启发（anticipation-heuristic）能力。

## 完全公正的排队I/O调度程序

完全公正的排队I/O调度程序（Complete Fair Queuing，CFQ）是为专有工作负荷设计的，不过，在实际中，也为多种工作负荷提供了良好的性能。

CFQ I/O调度程序把进入的I/O请求放入特定的队列中，这种队列是根据引起II/O请求的进程组织的。

CFQ I/O调度程序以时间片轮询调度队列，从每个队列中选取请求数，然后进行下一轮调度。这就在进程级提供了公平，确保每个进程接收公平的磁盘带宽片断。

## 空操作的I/O调度程序

## I/O调度程序的选择