

讲堂 > Linux性能优化实战 > 文章详情

24 | 基础篇：Linux 磁盘I/O是怎么工作的（上）

2019-01-14 倪朋飞



24 | 基础篇：Linux 磁盘I/O是怎么工作的（上）

朗读人：冯永吉 10'42" | 9.82M

你好，我是倪朋飞。

上一节，我们学习了 Linux 文件系统的工作原理。简单回顾一下，文件系统是对存储设备上的文件，进行组织管理的一种机制。而 Linux 在各种文件系统实现上，又抽象了一层虚拟文件系统 VFS，它定义了一组，所有文件系统都支持的，数据结构和标准接口。

这样，对应用程序来说，只需要跟 VFS 提供的统一接口交互，而不需要关注文件系统的具体实现；对具体的文件系统来说，只需要按照 VFS 的标准，就可以无缝支持各种应用程序。

VFS 内部又通过目录项、索引节点、逻辑块以及超级块等数据结构，来管理文件。

- 目录项，记录了文件的名字，以及文件与其他目录项之间的目录关系。
- 索引节点，记录了文件的元数据。
- 逻辑块，是由连续磁盘扇区构成的最小读写单元，用来存储文件数据。

- 超级块，用来记录文件系统整体的状态，如索引节点和逻辑块的使用情况等。

其中，目录项是一个内存缓存；而超级块、索引节点和逻辑块，都是存储在磁盘中的持久化数据。

那么，进一步想，磁盘又是怎么工作的呢？又有哪些指标可以用来衡量它的性能呢？

接下来，我就带你一起看看，Linux 磁盘 I/O 的工作原理。

磁盘 机械磁盘和固态硬盘

磁盘是可以持久化存储的设备，根据存储介质的不同，常见磁盘可以分为两类：机械磁盘和固态硬盘。

第一类，机械磁盘，也称为硬盘驱动器（Hard Disk Driver），通常缩写为 HDD。机械磁盘主要由盘片和读写磁头组成，数据就存储在盘片的环状磁道中。在读写数据前，需要移动读写磁头，定位到数据所在的磁道，然后才能访问数据。

显然，如果 I/O 请求刚好连续，那就不需要磁道寻址，自然可以获得最佳性能。这其实就是我们熟悉的，连续 I/O 的工作原理。与之相对应的，当然就是随机 I/O，它需要不停地移动磁头，来定位数据位置，所以读写速度就会比较慢。

第二类，固态硬盘（Solid State Disk），通常缩写为 SSD，由固态电子元器件组成。固态硬盘不需要磁道寻址，所以，不管是连续 I/O，还是随机 I/O 的性能，都比机械磁盘要好得多。

其实，无论机械磁盘，还是固态硬盘，相同磁盘的随机 I/O 都要比连续 I/O 慢很多，原因也很明显。

先擦除再写入

- 对机械磁盘来说，我们刚刚提到过的，由于随机 I/O 需要更多的磁头寻道和盘片旋转，它的性能自然要比连续 I/O 慢。
- 而对固态硬盘来说，虽然它的随机性能比机械硬盘好很多，但同样存在“先擦除再写入”的限制。随机读写会导致大量的垃圾回收，所以相对应的，随机 I/O 的性能比起连续 I/O 来，也还是差了很多。
- 此外，连续 I/O 还可以通过预读的方式，来减少 I/O 请求的次数，这也是其性能优异的一个原因。很多性能优化的方案，也都会从这个角度出发，来优化 I/O 性能。

随机读取与连续读取，预读取

此外，机械磁盘和固态硬盘还分别有一个最小的读写单位。

机械磁盘的最小读写单位是扇区，一般大小为512字节，而固态硬盘的最小读写单位是页，通常大小是4KB，8KB

- 机械磁盘的最小读写单位是扇区，一般大小为 512 字节。
- 而固态硬盘的最小读写单位是页，通常大小是 4KB、8KB 等。

在上一节中，我也提到过，如果每次都读写 512 字节这么小的单位的话，效率很低。所以，文件系统会把连续的扇区或页，组成逻辑块，然后以逻辑块作为最小单元来管理数据。常见的逻辑块的大小是 4KB，也就是说，连续 8 个扇区，或者单独的一个页，都可以组成一个逻辑块。

除了可以按照存储介质来分类，另一个常见的分类方法，是按照接口来分类，比如可以把硬盘分为 IDE (Integrated Drive Electronics)、SCSI (Small Computer System Interface)、SAS (Serial Attached SCSI)、SATA (Serial ATA)、FC (Fibre Channel) 等。

不同的接口，往往分配不同的设备名称。比如，IDE 设备会分配一个 `hd` 前缀的设备名，SCSI 和 SATA 设备会分配一个 `sd` 前缀的设备名。如果是多块同类型的磁盘，就会按照 a、b、c 等的字母顺序来编号。

除了磁盘本身的分类外，当你把磁盘接入服务器后，按照不同的使用方式，又可以把它们划分为多种不同的架构。

最简单的，就是直接作为独立磁盘设备来使用。这些磁盘，往往还会根据需要，划分为不同的逻辑分区，每个分区再用数字编号。比如我们前面多次用到的 `/dev/sda`，还可以分成两个分区 `/dev/sda1` 和 `/dev/sda2`。

冗余独立磁盘阵列 RAID

另一个比较常用的架构，是把多块磁盘组合成一个逻辑磁盘，构成冗余独立磁盘阵列，也就是 RAID (Redundant Array of Independent Disks)，从而可以提高数据访问的性能，并且增强数据存储的可靠性。

根据容量、性能和可靠性需求的不同，RAID 一般可以划分为多个级别，如 RAID0、RAID1、RAID5、RAID10 等。

- RAID0 有最优的读写性能，但不提供数据冗余的功能。
- 而其他级别的 RAID，在提供数据冗余的基础上，对读写性能也有一定程度的优化。

最后一种架构，是把这些磁盘组合成一个网络存储集群，再通过 NFS、SMB、iSCSI 等网络存储协议，暴露给服务器使用。

其实在 Linux 中，**磁盘实际上是作为一个块设备来管理的**，也就是以块为单位读写数据，并且支持随机读写。每个块设备都会被赋予两个设备号，分别是主、次设备号。主设备号用在驱动程序中，用来区分设备类型；而次设备号则是用来给多个同类设备编号。

通用块层

跟我们上一节讲到的虚拟文件系统 VFS 类似，为了减小不同块设备的差异带来的影响，Linux 通过一个统一的通用块层，来管理各种不同的块设备。

通用块层，其实是处在文件系统和磁盘驱动中间的一个块设备抽象层。它主要有两个功能。

- 第一个功能跟虚拟文件系统的功能类似。向上，为文件系统和应用程序，提供访问块设备的标准接口；向下，把各种异构的磁盘设备抽象为统一的块设备，并提供统一框架来管理这些设备的驱动程序。
- 第二个功能，通用块层还会给文件系统和应用程序发来的 I/O 请求排队，并通过重新排序、请求合并等方式，提高磁盘读写的效率。

其中，对 I/O 请求排序的过程，也就是我们熟悉的 I/O 调度。事实上，Linux 内核支持四种 I/O 调度算法，分别是 NONE、NOOP、CFQ 以及 DeadLine。这里我也分别介绍一下。

第一种 NONE，更确切来说，并不能算 I/O 调度算法。因为它完全不使用任何 I/O 调度器，对文件系统和应用程序的 I/O 其实不做任何处理，常用在虚拟机中（此时磁盘 I/O 调度完全由物理机负责）。

第二种 NOOP，是最简单的一种 I/O 调度算法。它实际上是一个先入先出的队列，只做一些基本的请求合并，常用于 SSD 磁盘。

第三种 CFQ (Completely Fair Scheduler)，也被称为完全公平调度器，是现在很多发行版的默认 I/O 调度器，它为每个进程维护了一个 I/O 调度队列，并按照时间片来均匀分布每个进程的 I/O 请求。

类似于进程 CPU 调度，CFQ 还支持进程 I/O 的优先级调度，所以它适用于运行大量进程的系统，像是桌面环境、多媒体应用等。

最后一种 DeadLine 调度算法，分别为读、写请求创建了不同的 I/O 队列，可以提高机械磁盘的吞吐量，并确保达到最终期限 (deadline) 的请求被优先处理。DeadLine 调度算法，多用在 I/O 压力比较重的场景，比如数据库等。

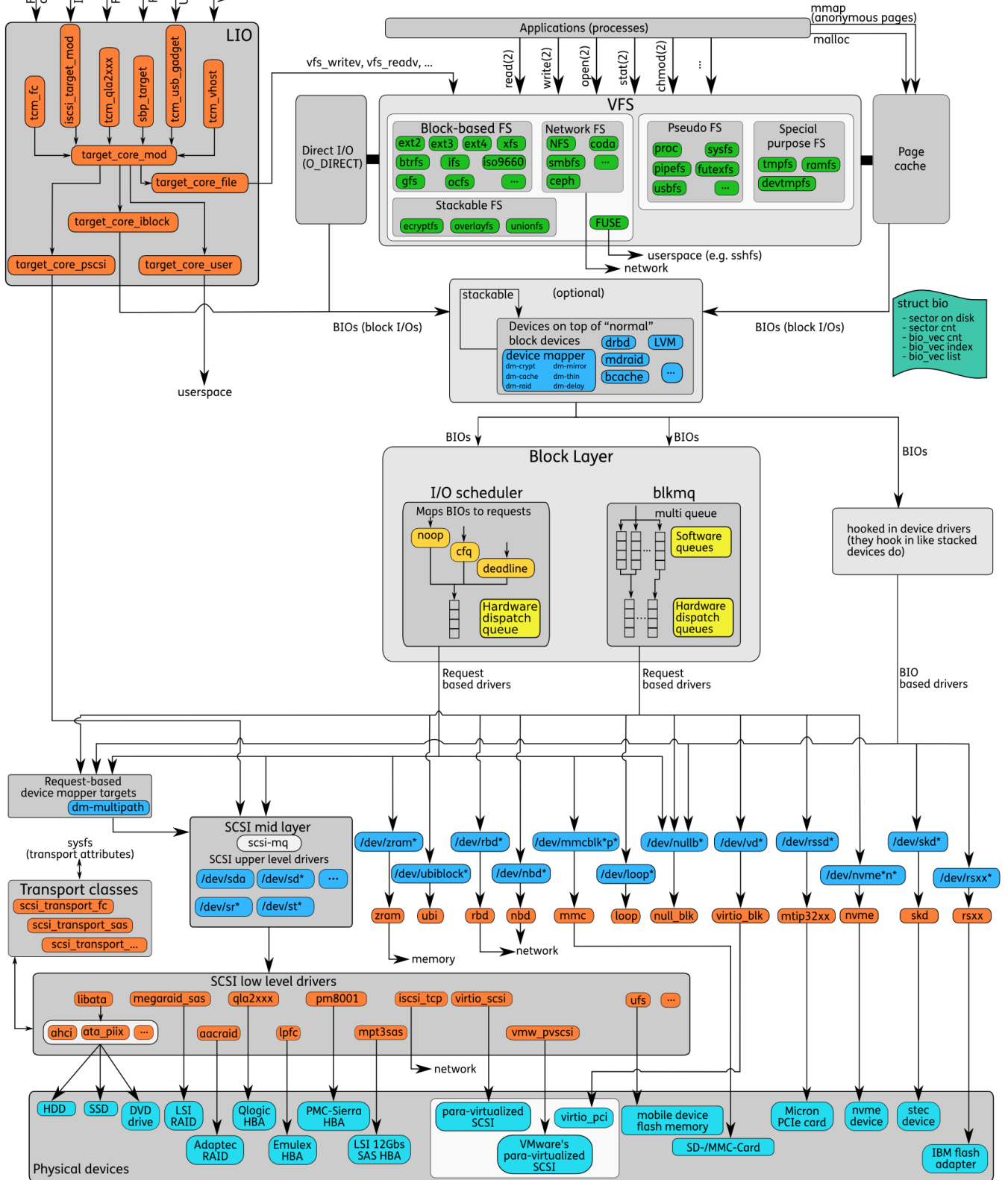
I/O 栈

清楚了磁盘和通用块层的工作原理，再结合上一期我们讲过的文件系统原理，我们就可以整体来看 Linux 存储系统的 I/O 原理了。

我们可以把 Linux 存储系统的 I/O 栈，由上到下分为三个层次，分别是文件系统层、通用块层和设备层。这三个 I/O 层的关系如下图所示，这其实也是 Linux 存储系统的 I/O 栈全景图。

The Linux Storage Stack Diagram

version 4.10, 2017-03-10
outlines the Linux storage stack as of Kernel version 4.10



The Linux Storage Stack Diagram
http://www.thomas-krenn.com/en/wiki/Linux_Storage_Stack_Diagram
Created by Werner Fischer and Georg Schönberger
License: CC-BY-SA 3.0, see <http://creativecommons.org/licenses/by-sa/3.0/>

THOMAS
KRENN®

(图片来自 [Linux Storage Stack Diagram](http://www.thomas-krenn.com/en/wiki/Linux_Storage_Stack_Diagram))

根据这张 I/O 栈的全景图，我们可以更清楚地理解，存储系统 I/O 的工作原理。

- 文件系统层，包括虚拟文件系统和其他各种文件系统的实现。它为上层的应用程序，提供标准的文件访问接口；对下会通过通用块层，来存储和管理磁盘数据。
- 通用块层，包括块设备 I/O 队列和 I/O 调度器。它会对文件系统的 I/O 请求进行排队，再通过重新排序和请求合并，然后才要发送给下一级的设备层。
- 设备层，包括存储设备和相应的驱动程序，负责最终物理设备的 I/O 操作。

存储系统的 I/O，通常是整个系统中最慢的一环。所以，Linux 通过多种缓存机制来优化 I/O 效率。

比方说，为了优化文件访问的性能，会使用页缓存、索引节点缓存、目录项缓存等多种缓存机制，以减少对下层块设备的直接调用。

同样，为了优化块设备的访问效率，会使用缓冲区，来缓存块设备的数据。

不过，抽象的原理讲了这么多，具体操作起来，应该怎么衡量磁盘的 I/O 性能呢？我先卖个关子，下节课我们一起来看，最常用的磁盘 I/O 性能指标，以及 I/O 性能工具。

小结

在今天的文章中，我们梳理了 Linux 磁盘 I/O 的工作原理，并了解了由文件系统层、通用块层和设备层构成的 Linux 存储系统 I/O 栈。

其中，通用块层是 Linux 磁盘 I/O 的核心。向上，它为文件系统和应用程序，提供访问了块设备标准接口；向下，把各种异构的磁盘设备，抽象为统一的块设备，并会对文件系统和应用程序发来的 I/O 请求进行重新排序、请求合并等，提高了磁盘访问的效率。

思考

最后，我想邀请你一起来聊聊，你所理解的磁盘 I/O。我相信你很可能已经碰到过，文件或者磁盘的 I/O 性能问题，你是怎么分析这些问题的呢？你可以结合今天的磁盘 I/O 原理和上一节的文件系统原理，记录你的操作步骤，并总结出思路。

欢迎在留言区和我讨论，也欢迎把这篇文章分享给你的同事、朋友。我们一起在实战中演练，在交流中进步。



Linux 性能优化实战

10 分钟帮你找到系统瓶颈

倪朋飞

微软资深工程师
Kubernetes 项目维护者



新版升级：点击「 请朋友读」，10位好友免费读，邀请订阅更有**现金**奖励。

©版权归极客邦科技所有，未经许可不得转载

上一篇 23 | 基础篇：Linux 文件系统是怎么工作的？

下一篇 25 | 基础篇：Linux 磁盘I/O是怎么工作的（下）

写留言

精选留言



DJH

1

有一个纠正一下：ISCSI访问的是块设备，不是NAS。

2019-01-14

作者回复

是的，这里有些不准确，原来的意思是想表达网络存储，用 NAS 这样的专业术语就不准确了。谢谢指出。

2019-01-14



Cranliu

1

最最常用的是iostat了吧？还有pidstat，sar等

2019-01-14

作者回复

是的 这三个都是最常用的

2019-01-14



coyang

0

- 1.用iostat看磁盘的await, utils, iops, bandwidth
- 2.用smartctl看磁盘的健康 status
- 3.用iotop/pidstat找出持续读写的进程做优化

2019-01-15



Maxwell

0

之前讲的buffer和cache就是缓存磁盘IO机制吧?

2019-01-15



我来也

0

[D24打卡]

以前其实并没太注意到磁盘I/O的性能.

平常就正常存储下程序的日志,程序的日志量也不大.

但是有一次在某云上,生产环境中的程序卡了持续3分多钟.

后来才发现是程序中有人打了一个超级大(其实也就2-3M)的日志,😓.

然后程序就在这里阻塞了几分钟.

2019-01-14



萨拉热窝的棒小伙儿

0

老师, 想问一下, 如果申请的测试环境资源与生产环境资源是有差异的, 那么在测试环境上做性能测试的话, 是否可以按照资源差异同比例缩小, 这个通过准则?

例如, 生产环境10个cpu, 5G 内存, 期望能并发100用户, 满足1秒响应。。。测试环境5个cpu, 2.5G, , 那么并发50用户, 满足1秒响应就行了, , 有这个说法嘛?

2019-01-14

| 作者回复

没有, 性能跟资源之间的关系不是线性的

2019-01-14



小老鼠

0

固态硬盘有扇区和磁道吗?

2019-01-14

| 作者回复

没有, SSD读写的基本单位是页

2019-01-14



JohnT3e

0

工作中经常看到使用多线程读写单个磁盘中不同文件的实现, 个人认为这种方法并不能有效地提高性能, 因为每个线程读写磁盘的位置可能差异很大, 且每个线程的数据的空间局部性和时间局部性存在差异, 导致磁盘调度优化不足. 不知道对不对

2019-01-14

| 作者回复

实际上这是可以提高性能的，一方面，文件的读写是有操作系统缓存的，每次文件读写调用并不是立刻对应一个磁盘IO操作，在文件系统和块设备层依然可以作很多优化（比如最基本的排序、合并）。另一方面，如果是逐个来写的话，多个线程的工作就有可能被这一系列的文件读写阻塞，而分开来每个线程就只需要阻塞自己所读写的文件（假设采用阻塞I/O）

2019-01-14



划时代
打卡总结

2019-01-14

0



xfan
打卡

2019-01-14

0



13001236383
FC好像是传输scisi的协议，不是硬盘接口吧

2019-01-14

0

作者回复

以前的确是这样，当现在也会用作硬件的接口，特别是高速传输的场景中

2019-01-14



ninuxer
day25打卡

所以这也就是如kafka这类队列用磁盘的顺序io实现高速队列的依据，磁盘顺序io的性能比内存的好~

2019-01-14

0

作者回复

嗯嗯 是的

2019-01-14



black_mirror
倪老师，您好：

1.从内核版本2.6.32以后，磁盘io调度算法好像只有3种，文中提到的none算法系统层没看到：

```
cat /sys/block/sda/queue/scheduler
```

```
noop anticipatory deadline [cfq]
```

2.在运行db的环境中，如MySQL，mysql的程序运行在机械硬盘上，mysql的数据目录是挂载在dell存储上(2个控制器有缓存)，存储也是由机械硬盘组成，做的raid10/raid6

通过sysbench压测，其它环境保持一致，仅修改磁盘调度算法为noop/deadline：

进行只读、读写测试，分别测试3轮，每轮10分钟，求平均值比较，50个并发线程，50张表每张100w数据，oltp场景，每轮测试后，清空系统缓存和重启mysql程序

压测结果：只读场景noop比deadline好一点点，读写场景deadline比noop好一点点

请问老师这种场景下，应该选择noop 还是 deadline？因为看到下面一篇文章有些疑惑

0

Choosing a Disk Scheduling Algorithm

The disk controller receives requests from the operating system and processes them in an order determined by a scheduling algorithm. Sometimes changing this algorithm can improve disk performance. For other hardware and workloads, it may not make a difference. The best way to decide is to test them out yourself on your workload. Deadline and completely fair queueing (CFQ) both tend to be good choices.

There are a couple of situations where the noop scheduler (a contraction of “no-op” is the best choice): if you’re in a virtualized environment, use the noop scheduler. The noop scheduler basically passes the operations through to the underlying disk controller as quickly as possible. It is fastest to do this and let the real disk controller handle any reordering that needs to happen.

Similarly, on SSDs, the noop scheduler is generally the best choice. SSDs don’t have the same locality issues that spinning disks do.

Finally, if you’re using a RAID controller with caching, use noop. The cache behaves like an SSD and will take care of propagating the writes to the disk efficiently.

选择磁盘调度算法

磁盘控制器接收来自操作系统的请求，并按调度算法确定的顺序处理它们。有时更改此算法可以提高磁盘性能。对于其他硬件和工作负载，它可能没有什么区别。决定的最佳方法是自己测试工作量。deadline和CFQ往往都是不错的选择。

有几种情况下，noop调度程序是最佳选择：如果您处于虚拟化环境中，请使用noop调度程序。noop调度程序基本上将操作尽快传递到底层磁盘控制器。这样做最快，让真正的磁盘控制器处理需要发生的任何重新排序。同样，在SSD上，noop调度程序通常是最佳选择。SSD没有与旋转磁盘相同的位置问题。

最后，如果您使用带有缓存的RAID控制器，请使用noop。缓存的行为类似于SSD，并且会有效地将写入传播到磁盘。

2019-01-14

作者回复

1. none 主要用在虚拟机中，跟发行版有关系
2. noop和deadline都可以，不过简单的测试跟应用程序的访问模式很可能不一致，建议拿应用程序的逻辑来测试（或者使用I/O重放的方法）

<https://dev.mysql.com/doc/refman/5.7/en/optimizing-innodb-diskio.html> 有一些优化的建议

2019-01-14



Orcsir

Flag

2109/01/14

2019-01-14

0



👍 🗨️ 📌



耿衣子

0

打卡，有点不懂呀，希望老师多来点案例，结合案例将原理

2019-01-14

| 作者回复

后面有的

2019-01-14



金波

0

有个问题一直没弄明白，想借此机会请老师解答下，大家经常用select来多路复用读写管道，socket等类型的文件，请问select可否用于普通磁盘文件的读写？不行的话为什么？多谢

2019-01-14

| 作者回复

碰到这种问题最好的方法是查手册，比如在 select 的文档 (<https://linux.die.net/man/3/select>) 中你可以看到：

File descriptors associated with regular files shall always select true for ready to read, ready to write, and error conditions.

也就是普通文件读、写总是 Ready 的，所以用 select 也就没有意义。当然，在编程接口上，你可以还是调用它的。

2019-01-14