04 | 零拷贝：如何高效地传输文件?

2020-05-06 陶辉

系统性能调优必知必会 进入课程 >

讲述：陶辉
时长 13:46 大小 11.04M

你好，我是陶辉。

上一讲我们谈到，当索引的大小超过内存时，就会用磁盘存放索引。磁盘的读写速度远慢于内存，所以才针对磁盘设计了减少读写次数的 B 树索引。

磁盘是主机中最慢的硬件之一，常常是性能瓶颈，所以优化它能获得立竿见影的效果。

因此，针对磁盘的优化技术层出不穷，比如零拷贝、直接 IO、异步 IO 等等。这些优化技术为了降低操作时延、提升系统的吞吐量，围绕着内核中的磁盘高速缓存（也叫 PageCache），去减少 CPU 和磁盘设备的工作量。

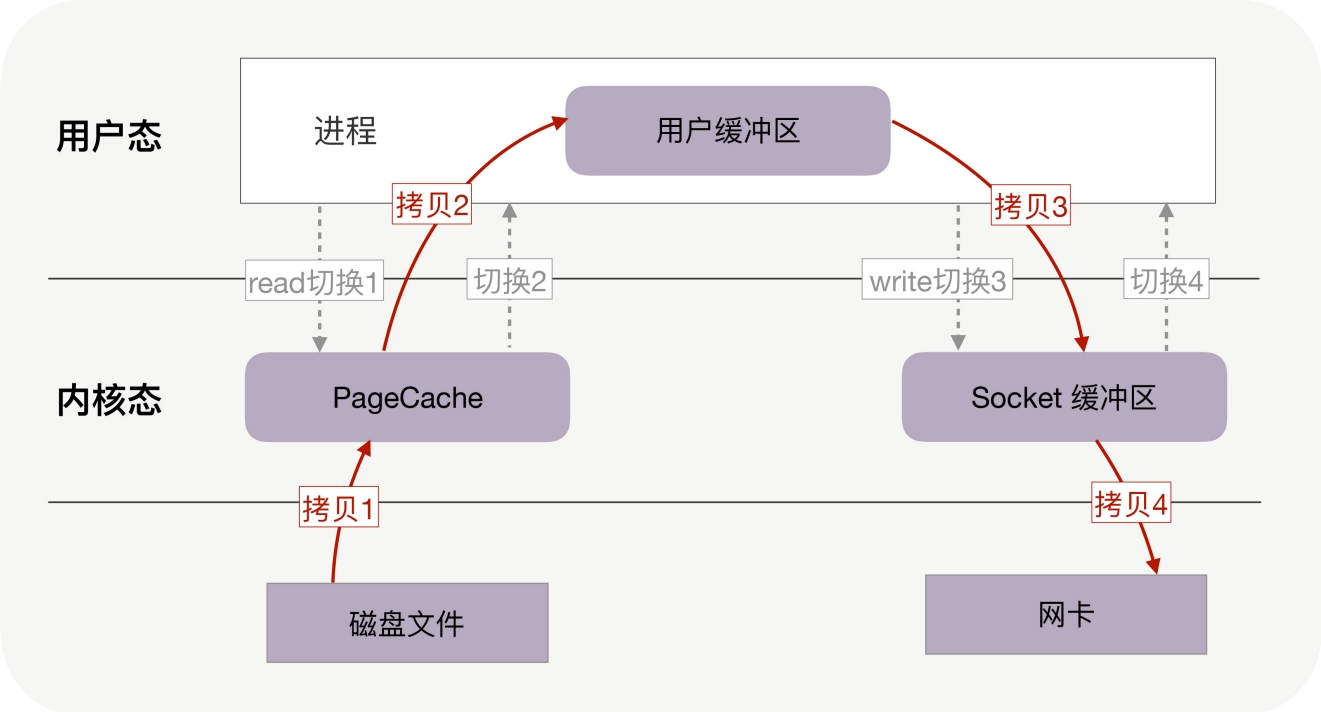
这些磁盘优化技术和策略虽然很有效，但是理解它们并不容易。只有搞懂内核操作磁盘的流程，灵活正确地使用，才能有效地优化磁盘性能。

这一讲，我们就通过解决“如何高效地传输文件”这个问题，来分析下磁盘是如何工作的，并且通过优化传输文件的性能，带你学习现在热门的零拷贝、异步 IO 与直接 IO 这些磁盘优化技术。

你会如何实现文件传输？

服务器提供文件传输功能，需要将磁盘上的文件读取出来，通过网络协议发送到客户端。如果需要你自己编码实现这个文件传输功能，你会怎么实现呢？

通常，你会选择最直接的方法：从网络请求中找出文件在磁盘中的路径后，如果这个文件比较大，假设有 320MB，可以在内存中分配 32KB 的缓冲区，再把文件分成一万份，每份只有 32KB，这样，从文件的起始位置读入 32KB 到缓冲区，再通过网络 API 把这 32KB 发送到客户端。接着重复一万次，直到把完整的文件都发送完毕。如下图所示：



不过这个方案性能并不好，主要有两个原因。

首先，它至少**经历了 4 万次用户态与内核态的上下文切换。**因为每处理 32KB 的消息，就需要一次 read 调用和一次 write 调用，每次系统调用都得先从用户态切换到内核态，等内

核完成任务后，再从内核态切换回用户态。可见，每处理 32KB，就有 4 次上下文切换，重复 1 万次后就有 4 万次切换。

上下文切换的成本并不小，虽然一次切换仅消耗几十纳秒到几微秒，但高并发服务会放大这类时间的消耗。

其次，这个方案做了 **4 万次内存拷贝，对 320MB 文件拷贝的字节数也翻了 4 倍，到了 1280MB。**很显然，过多的内存拷贝无谓地消耗了 CPU 资源，降低了系统的并发处理能力。

所以要想提升传输文件的性能，需要从**降低上下文切换的频率和内存拷贝次数**两个方向入手。

零拷贝如何提升文件传输性能?

首先，我们来看如何降低上下文切换的频率。

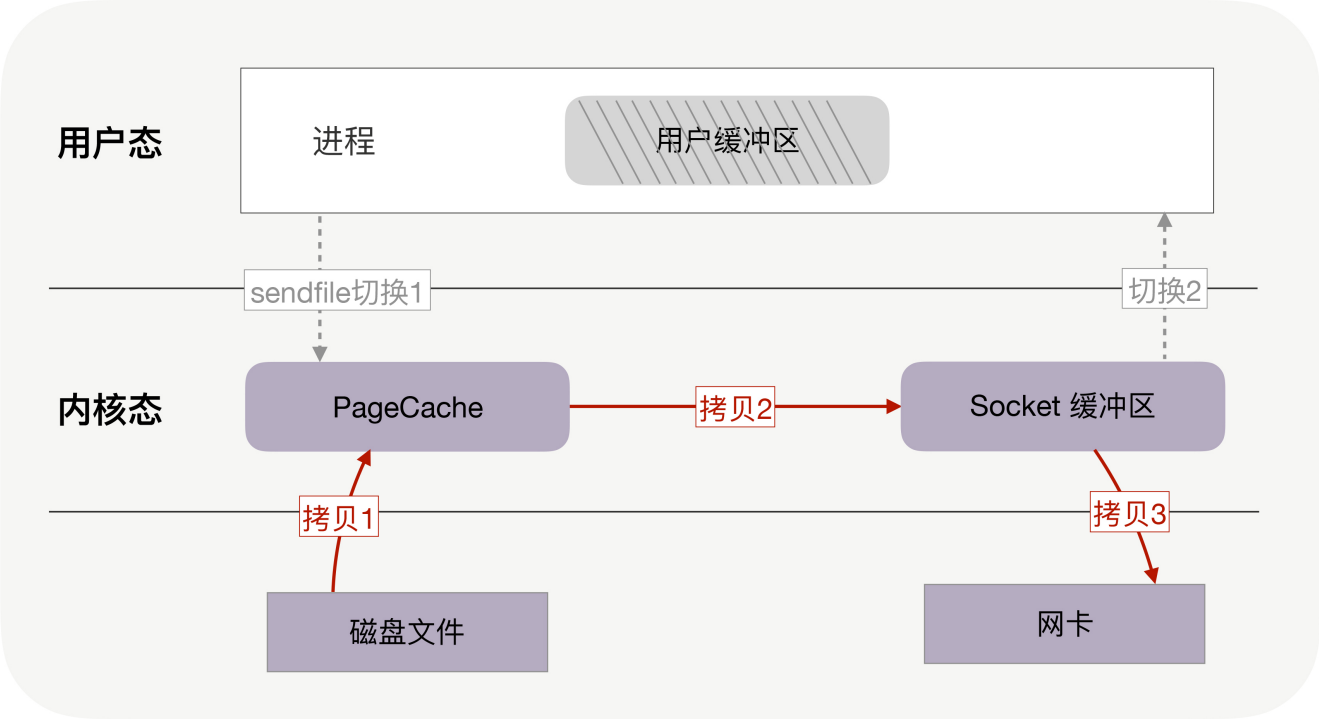
为什么读取磁盘文件时，一定要做上下文切换呢？这是因为，读取磁盘或者操作网卡都由操作系统内核完成。内核负责管理系统上的所有进程，它的权限最高，工作环境与用户进程完全不同。只要我们的代码执行 read 或者 write 这样的系统调用，一定会发生 2 次上下文切换：首先从用户态切换到内核态，当内核执行完任务后，再切换回用户态交由进程代码执行。

因此，如果想减少上下文切换次数，就一定要减少系统调用的次数。解决方案就是把 read、write 两次系统调用合并成一次，在内核中完成磁盘与网卡的数据交换。

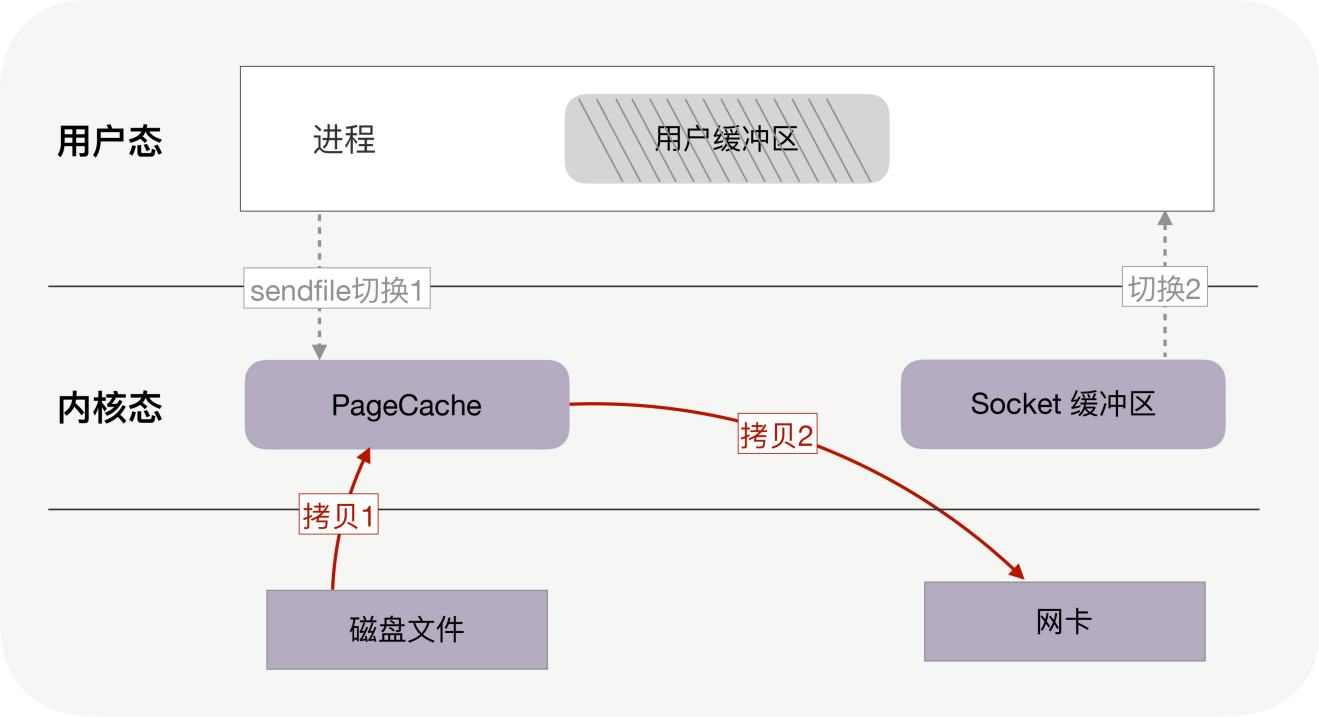
其次，我们应该考虑如何减少内存拷贝次数。

每周期中的 4 次内存拷贝，其中与物理设备相关的 2 次拷贝是必不可少的，包括：把磁盘内容拷贝到内存，以及把内存拷贝到网卡。但另外 2 次与用户缓冲区相关的拷贝动作都不是必需的，因为在把磁盘文件发到网络的场景中，**用户缓冲区没有必须存在的理由。**

如果内核在读取文件后，直接把 PageCache 中的内容拷贝到 Socket 缓冲区，待到网卡发送完毕后，再通知进程，这样就只有 2 次上下文切换，和 3 次内存拷贝。



如果网卡支持 SG-DMA（The Scatter-Gather Direct Memory Access）技术，还可以再去除 Socket 缓冲区的拷贝，这样一共只有 2 次内存拷贝。



实际上，这就是零拷贝技术。

它是操作系统提供的新函数，同时接收文件描述符和 TCP socket 作为输入参数，这样执行时就可以完全在内核态完成内存拷贝，既减少了内存拷贝次数，也降低了上下文切换次数。

而且，零拷贝取消了用户缓冲区后，不只降低了用户内存的消耗，还通过最大化利用 socket 缓冲区中的内存，间接地再一次减少了系统调用的次数，从而带来了大幅减少上下文切换次数的机会！

你可以回忆下，没用零拷贝时，为了传输 320MB 的文件，在用户缓冲区分配了 32KB 的内存，把文件分成 1 万份传送，然而，**这 32KB 是怎么来的？**为什么不是 32MB 或者 32 字节呢？这是因为，在没有零拷贝的情况下，我们希望内存的利用率最高。如果用户缓冲区过大，它就无法一次性把消息全拷贝给 socket 缓冲区；如果用户缓冲区过小，则会导致过多的 read/write 系统调用。

那用户缓冲区为什么不与 socket 缓冲区大小一致呢？这是因为，**socket 缓冲区的可用空间是动态变化的**，它既用于 TCP 滑动窗口，也用于应用缓冲区，还受到整个系统内存的影响（我在《Web 协议详解与抓包实战》第 5 部分课程对此有详细介绍，这里不再赘述）。尤其在长肥网络中，它的变化范围特别大。

零拷贝使我们不必关心 socket 缓冲区的大小。比如，调用零拷贝发送方法时，尽可以把发送字节数设为文件的所有未发送字节数，例如 320MB，也许此时 socket 缓冲区大小为 1.4MB，那么一次性就会发送 1.4MB 到客户端，而不是只有 32KB。这意味着对于 1.4MB 的 1 次零拷贝，仅带来 2 次上下文切换，而不使用零拷贝且用户缓冲区为 32KB 时，经历了 176 次（4 * 1.4MB/32KB）上下文切换。

综合上述各种优点，**零拷贝可以把性能提升至少一倍以上！**对文章开头提到的 320MB 文件的传输，当 socket 缓冲区在 1.4MB 左右时，只需要 4 百多次上下文切换，以及 4 百多次内存拷贝，拷贝的数据量也仅有 640MB，这样，不只请求时延会降低，处理每个请求消耗的 CPU 资源也会更少，从而支持更多的并发请求。

此外，零拷贝还使用了 PageCache 技术，通过它，零拷贝可以进一步提升性能，我们接下来看看 PageCache 是如何做到这一点的。

PageCache，磁盘高速缓存

回顾上文中的几张图，你会发现，读取文件时，是先把磁盘文件拷贝到 PageCache 上，再拷贝到进程中。为什么这样做呢？有两个原因所致。

第一，由于磁盘比内存的速度慢许多，所以我们应该想办法把读写磁盘替换成读写内存，比如把磁盘中的数据复制到内存中，就可以用读内存替换读磁盘。但是，内存空间远比磁盘要小，内存中注定只能复制一小部分磁盘中的数据。

选择哪些数据复制到内存呢？通常，刚被访问的数据在短时间内再次被访问的概率很高（这也叫“时间局部性”原理），用 PageCache 缓存最近访问的数据，当空间不足时淘汰最久未被访问的缓存（即 LRU 算法）。读磁盘时优先到 PageCache 中找一找，如果数据存在便直接返回，这便大大提升了读磁盘的性能。

第二，读取磁盘数据时，需要先找到数据所在的位置，对于机械磁盘来说，就是旋转磁头到数据所在的扇区，再开始顺序读取数据。其中，旋转磁头耗时很长，为了降低它的影响，PageCache 使用了**预读功能**。

也就是说，虽然 read 方法只读取了 0-32KB 的字节，但内核会把其后的 32-64KB 也读取到 PageCache，这后 32KB 读取的成本很低。如果在 32-64KB 淘汰出 PageCache 前，进程读取到它了，收益就非常大。这一讲的传输文件场景中这是必然发生的。

从这两点可以看到 PageCache 的优点，它在 90% 以上场景下都会提升磁盘性能，**但在某些情况下，PageCache 会不起作用，甚至由于多做了一次内存拷贝，造成性能的降低。**在这些场景中，使用了 PageCache 的零拷贝也会损失性能。

具体是什么场景呢？就是在传输大文件的时候。比如，你有很多 GB 级的文件需要传输，每当用户访问这些大文件时，内核就会把它们载入到 PageCache 中，这些大文件很快会把有限的 PageCache 占满。

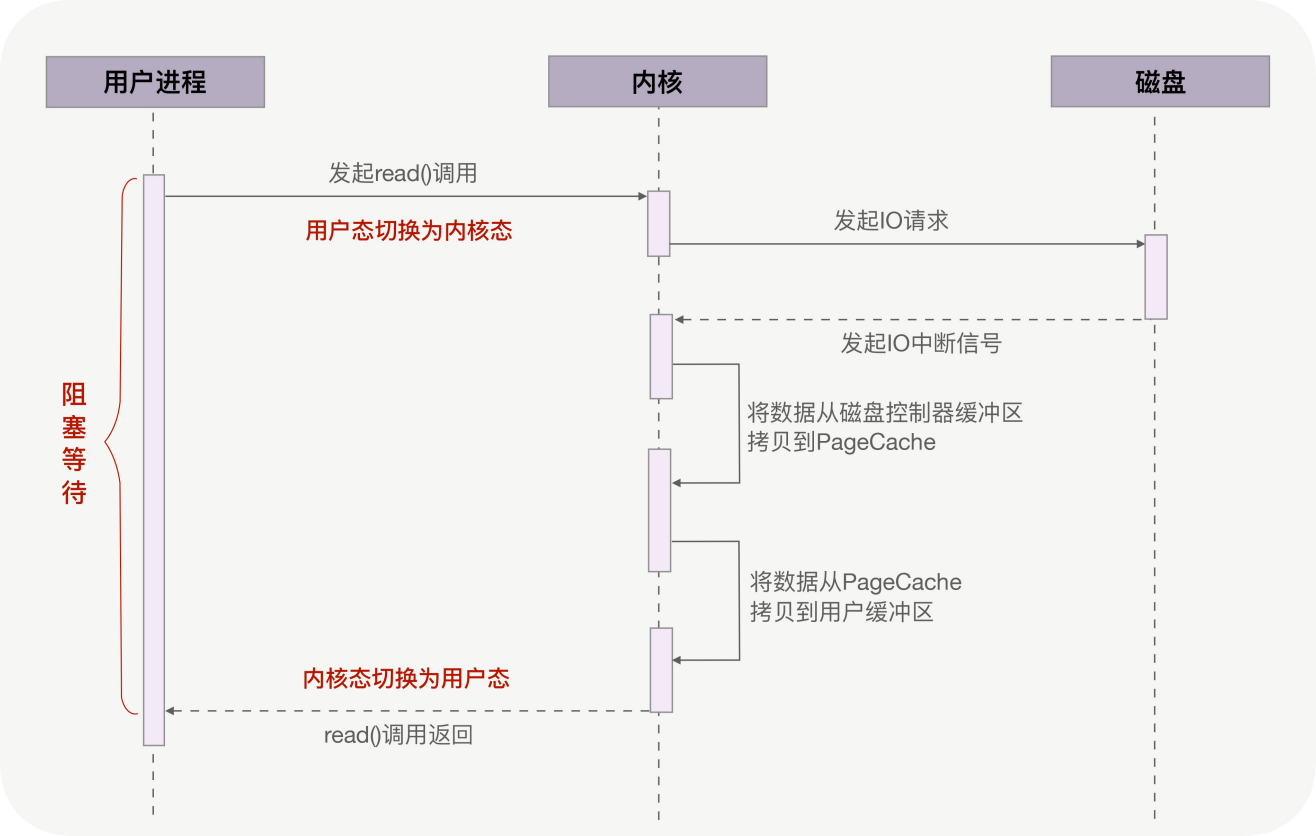
然而，由于文件太大，文件中某一部分内容被再次访问到的概率其实非常低。这带来了 2 个问题：首先，由于 PageCache 长期被大文件占据，热点小文件就无法充分使用 PageCache，它们读起来变慢了；其次，PageCache 中的大文件没有享受到缓存的好处，但却耗费 CPU 多拷贝到 PageCache 一次。

所以，高并发场景下，为了防止 PageCache 被大文件占满后不再对小文件产生作用，**大文件不应使用 PageCache，进而也不应使用零拷贝技术处理。**

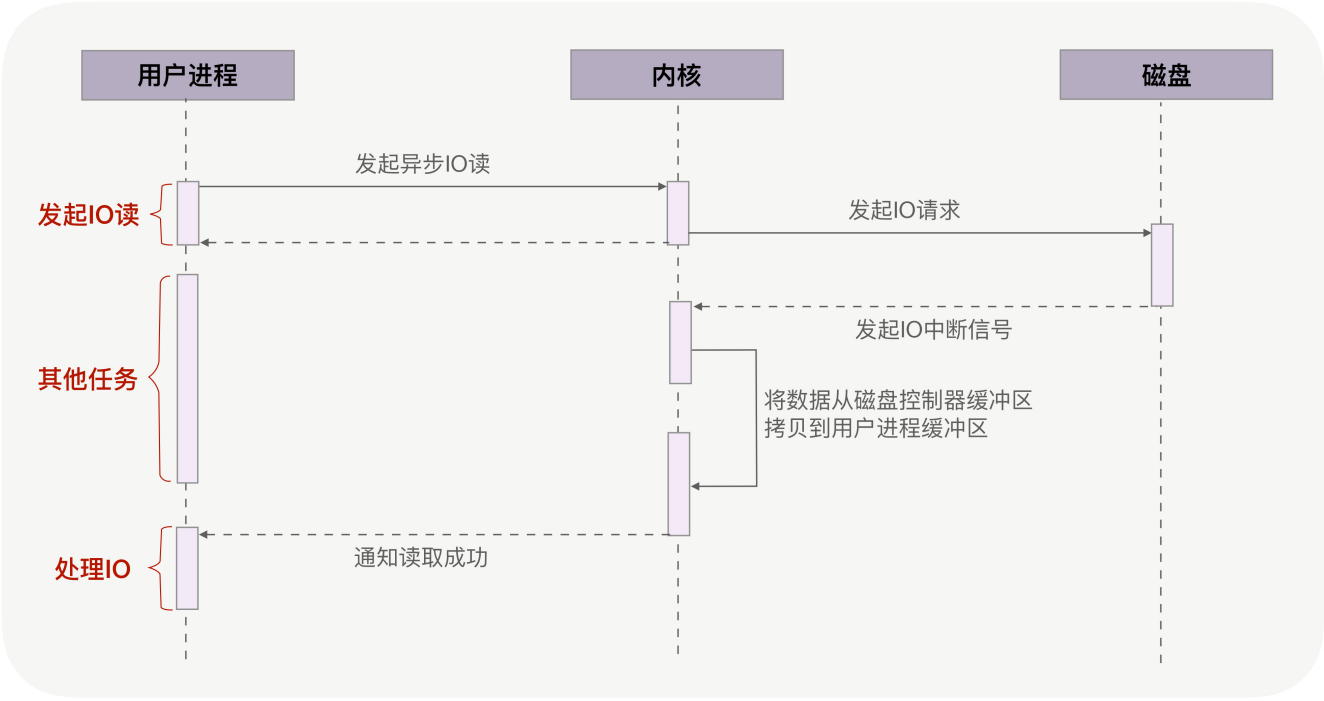
异步 IO + 直接 IO

高并发场景处理大文件时，应当使用异步 IO 和直接 IO 来替换零拷贝技术。

仍然回到本讲开头的例子，当调用 read 方法读取文件时，实际上 read 方法会在磁盘寻址过程中阻塞等待，导致进程无法并发地处理其他任务，如下图所示：



异步 IO（异步 IO 既可以处理网络 IO，也可以处理磁盘 IO，这里我们只关注磁盘 IO）可以解决阻塞问题。它把读操作分为两部分，前半部分向内核发起读请求，但**不等待数据就位就立刻返回**，此时进程可以并发地处理其他任务。当内核将磁盘中的数据拷贝到进程缓冲区后，进程将接收到内核的通知，再去处理数据，这是异步 IO 的后半部分。如下图所示：



从图中可以看到，异步 IO 并没有拷贝到 PageCache 中，这其实是异步 IO 实现上的缺陷。经过 PageCache 的 IO 我们称为缓存 IO，它与虚拟内存系统耦合太紧，导致异步 IO 从诞生起到现在都不支持缓存 IO。

绕过 PageCache 的 IO 是个新物种，我们把它称为直接 IO。对于磁盘，异步 IO 只支持直接 IO。

直接 IO 的应用场景并不多，主要有两种：第一，应用程序已经实现了磁盘文件的缓存，不需要 PageCache 再次缓存，引发额外的性能消耗。比如 MySQL 等数据库就使用直接 IO；第二，高并发下传输大文件，我们上文提到过，大文件难以命中 PageCache 缓存，又带来额外的内存拷贝，同时还挤占了小文件使用 PageCache 时需要的内存，因此，这时应该使用直接 IO。

当然，直接 IO 也有一定的缺点。除了缓存外，内核（IO 调度算法）会试图缓存尽量多的连续 IO 在 PageCache 中，最后**合并**成一个更大的 IO 再发给磁盘，这样可以减少磁盘的寻址操作；另外，内核也会**预读**后续的 IO 放在 PageCache 中，减少磁盘操作。直接 IO 绕过了 PageCache，所以无法享受这些性能提升。

有了直接 IO 后，异步 IO 就可以无阻塞地读取文件了。现在，大文件由异步 IO 和直接 IO 处理，小文件则交由零拷贝处理，至于判断文件大小的阈值可以灵活配置（参见 Nginx 的 directio 指令）。

小结

基于用户缓冲区传输文件时，过多的内存拷贝与上下文切换次数会降低性能。零拷贝技术在内核中完成内存拷贝，天然降低了内存拷贝次数。它通过一次系统调用合并了磁盘读取与网络发送两个操作，降低了上下文切换次数。尤其是，由于拷贝在内核中完成，它可以最大化使用 socket 缓冲区的可用空间，从而提高了一次系统调用中处理的数据量，进一步降低了上下文切换次数。

零拷贝技术基于 PageCache，而 PageCache 缓存了最近访问过的数据，提升了访问缓存数据的性能，同时，为了解决机械磁盘寻址慢的问题，它还协助 IO 调度算法实现了 IO 合并与预读（这也是顺序读比随机读性能好的原因），这进一步提升了零拷贝的性能。几乎所有操作系统都支持零拷贝，如果应用场景就是把文件发送到网络中，那么我们应当选择使用了零拷贝的解决方案。

不过，零拷贝有一个缺点，就是不允许进程对文件内容作一些加工再发送，比如数据压缩后再发送。另外，当 PageCache 引发负作用时，也不能使用零拷贝，此时可以用异步 IO+直接 IO 替换。我们通常会设定一个文件大小阈值，针对大文件使用异步 IO 和直接 IO，而对小文件使用零拷贝。

事实上 PageCache 对写操作也有很大的性能提升，因为 write 方法在写入内存中的 PageCache 后就会返回，速度非常快，由内核负责异步地把 PageCache 刷新到磁盘中，这里不再展开。

这一讲我们从零拷贝出发，看到了文件传输场景中内核在幕后所做的工作。这里面的性能优化技术，要么减少了磁盘的工作量（比如 PageCache 缓存），要么减少了 CPU 的工作量（比如直接 IO），要么提高了内存的利用率（比如零拷贝）。你在学习其他磁盘 IO 优化技术时，可以延着这三个优化方向前进，看看究竟如何降低时延、提高并发能力。

思考题

最后，留给你一个思考题，异步 IO 一定不会阻塞进程吗？如果阻塞了进程，该如何解决呢？欢迎你在留言区与大家一起探讨。

感谢阅读，如果你觉得这节课对你有一些启发，也欢迎把它分享给你的朋友。

上一篇 03 | 索引：如何用哈希表管理亿级对象？

下一篇 05 | 协程：如何快速地实现高并发服务？

精选留言 (18)

写留言

忆水寒
2020-05-06

我觉得有两种情况：
1、如果这里的异步IO指的是传统的异步阻塞IO，比如select，epoll等。这种情况下是可能阻塞进程的，在内核通知用户进程数据准备好了以后，用户进程发起read调用，此时内核拷贝数据期间，进程实际上是阻塞的。
解决方案：可以在内核完成拷贝后再通知用户进程，或者使用mmap方式，用户态和内...

展开

2 9

每天晒白牙
2020-05-07

今日得到
服务器提供文件传输功能，首先从磁盘读取文件，然后通过网络协议发送给客户端。最直接的办法是根据客户端的请求从磁盘上找到文件位置，然后从磁盘把部分文件（一般文件比较大时，需要对文件进行切分）读入到缓冲区，然后再通过网络把数据发送给客户端。方法的不足...
展开

3

问题大师
2020-05-06

从图中可以看到，异步 IO 并没有拷贝到 PageCache 中，这其实是异步 IO 实现上的缺陷。经过 PageCache 的 IO 我们称为缓存 IO，它与虚拟内存系统耦合太紧，导致异步 IO 从诞生起到现在都不支持缓存 IO。
陶老师 我在异步IO图中，看到的是把磁盘数据拷贝到PageCache，是图错了嘛，如果是直接IO的话，是直接拷贝到用户进程缓存区嘛，这个过程就是绕过了内核态嘛
展开

作者回复: 你好问题大师，谢谢你的提醒，图上的文字错啦，我马上联系编辑小姐姐更正！

5

2

Bitstream
2020-05-06

到现在一共更了5讲，除了开篇，每讲都是干货。不是因为不知道老师讲的知识点，而是您讲的很系统，以场景带理论，学习起来很高效。
展开

作者回复: 你好Bitstream，谢谢你的反馈，我会继续按照这个思路写后续的几讲

2

test
2020-05-06

有异步阻塞IO，取决于进程在IO读取点时候在做什么
展开

2

上校
2020-05-10

陶辉老师，你好，可不可以对应的知识点给些代码的参考呢？开源项目哪里用到了？或者github有对应的实现？

展开

1

Ken
2020-05-08

长肥网络定义

一个具有大带宽时延乘积的网络也被称之为长胖网络（long fat network，简写为LFN，经常发音为“elephen”）。根据RFC 1072中的定义，如果一个网络的带宽时延乘积显著大于105比特（12500字节），该网络被认为是长肥网络。

展开

1

每天晒白牙
2020-05-07

疑惑点
1.在介绍零拷贝时，降低上下文切换频率提到的将 read 和 write 两次系统调用合并为一次是通过 sendfile 实现吗?
2.异步 IO 没有使用 PageCache，因为与虚拟内存系统耦合太紧，这块没有看明白
...

展开

1

test
2020-05-06

真的是每篇都是干货，买对了。

展开

1

不重要
2020-05-11

老师，拷贝模式下，我还是没有看懂32K这个缓冲大小是以什么标准作为定量的？

zlw
2020-05-11

大文件太大会导致pagecache被占满，好处是可以减少读取上下文切换，那分多次读入pagecache可以避免这个问题？异步直接io一次是读取大量数据到用户空间，但发送时还是

得按照socket缓冲区情况分很多次发送?

C家族铁粉
2020-05-08

看到陶辉老师在部落里说，最开始文章有6000字，后来不停删删删，变成了现在的版本，太可惜了，删减掉的部分可以考虑放到其他地方供读者阅读啊。

展开

作者回复: 没有啦，我思维中的知识是网状的，文字是线性的，你在线性阅读中能够坚持下去，最后还原为树状、网状知识，其实要求挺高的。之前我的行文枝节太多，很难让多数读者坚持下来，是编辑小姐姐对我各种指导，集中炮火攻击一点，才有现在比较通顺流畅的文章，^_^

那时刻
2020-05-07

异步IO可能会阻塞进程，我理解的是内核向磁盘发送完读请求之后，才返回到调用方。可以采用多线程读文件的方式来解决？类似于Netty处理网络数据采用的reactive方式。

hanazawakana
2020-05-06

windows的IOCP才是真正的异步IO吧，linux现在的AIO应该不能做到真正的不阻塞吧

hanazawakana
2020-05-06

直接IO还是需要4次内核态用户态切换吗

展开

1

石皮皮
2020-05-06

请教您个问题，陶老师。我们有个业务要上传二百多个文件，用户量是几十万，这种情形下怎么很好的控制(或估量)文件传输所占用的带宽呢?

展开

一步

2020-05-06

PageCache 中的大文件没有享受到缓存的好处，但却耗费 CPU 多拷贝到 PageCache 一次

为什么说多拷贝一次 到 PageCache 呢？ 不是都是会从磁盘拷贝数据到内核态呢？ 只是零拷贝会把数据拷贝到 PageCache 中；不使用 零拷贝 也会把数据拷贝到内核态其他地...

展开

2

木木匠

2020-05-06

课后思考题:
我觉得异步IO还是会阻塞进程，在图中的时序图中，当进程收到IO处理完毕的通知后，进程需要进行IO处理，这个时候需要从内核态复制数据到用户态，这个过程是阻塞的。
解决方法:我只想到一个思路，不知道是否可行，如果能够异步IO能把数据复制到用户态之后再通知，那就是真正的异步非阻塞了。...

展开