深入理解零拷贝技术

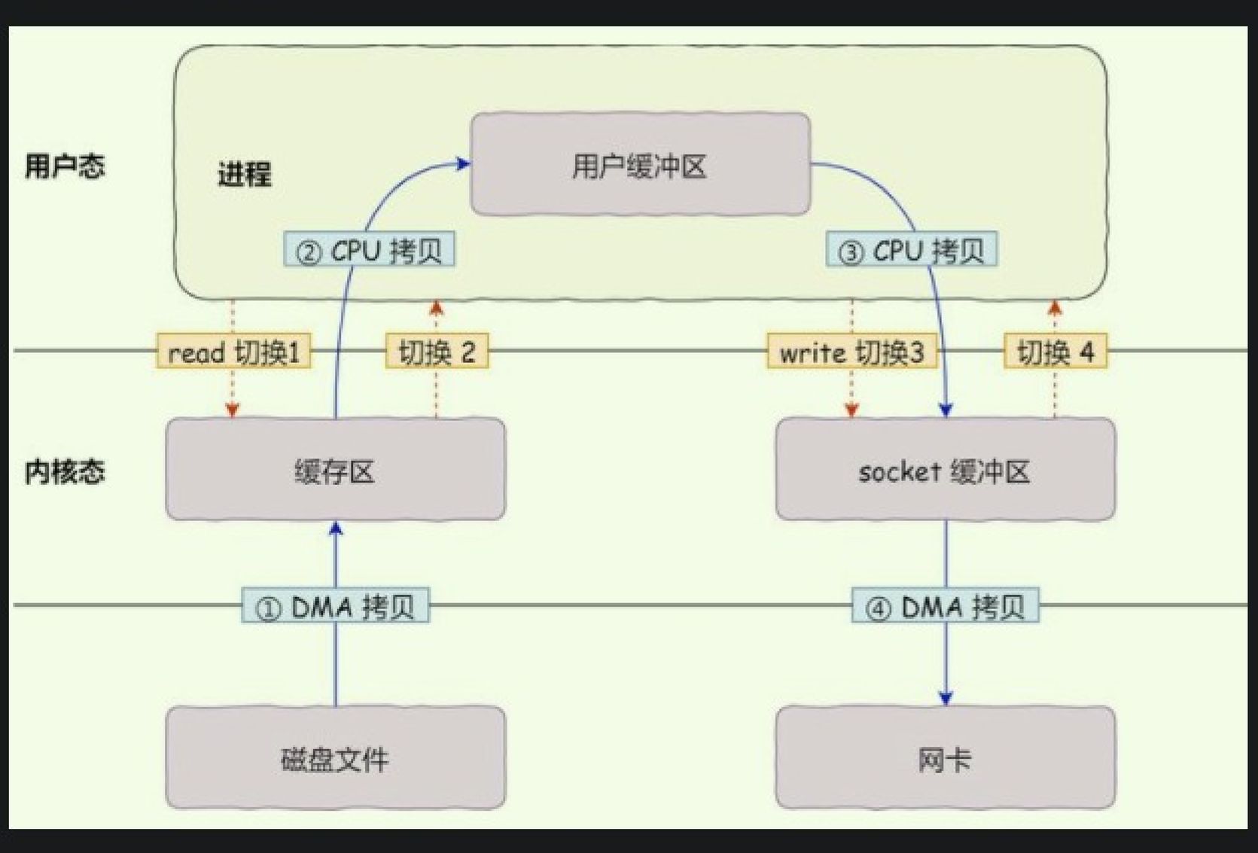
前言

> 零拷贝技术是指计算机执行操作时，CPU不需要先将数据从某处内存复制到另一个特定区域。这种技术通常用于通过网络传输文件时节省CPU周期和内存带宽。

原始的网络请求，需要数次在用户态和内核态之间切换以及数据的拷贝，这无疑大大影响了处理的效率，零拷贝技术就是为解决这一问题而诞生的。

我们常见的高性能组件（Netty、Kafka等），其内部基本都应用了零拷贝，在学习这些组件之前，有必要先了解什么是零拷贝。

传统文件传输 read + write



> DMA拷贝：指外部设备不通过CPU而直接与系统内存交换数据的接口技术

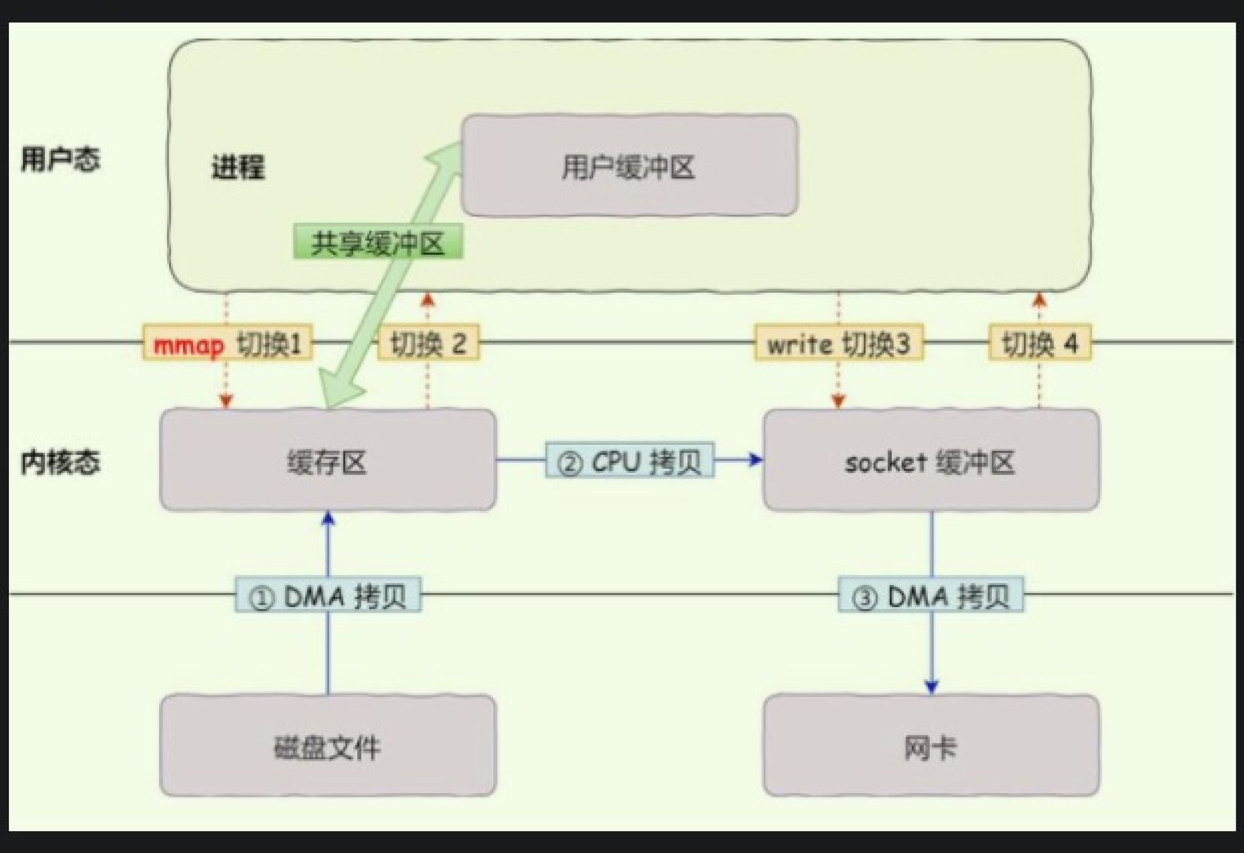
如上图所示，传统的网络传输，需要进行**4次用户态和内核态切换，4次数据拷贝（2次CPU拷贝，2次DMA拷贝）**

上下文的切换涉及到操作系统，相对CPU速度是非常耗时的，而且仅仅一次文件传输，竟然需要4次数据拷贝，造成CPU资源极大的浪费

不难看出，传统网络传输涉及很多冗余且无意义的操作，导致应用在高并发情况下，性能指数级下降，表现异常糟糕

为了解决这一问题，零拷贝技术诞生了，他其实是一个抽象的概念，但其本质就是通过减少上下文切换和数据拷贝次数来实现的

mmap + write



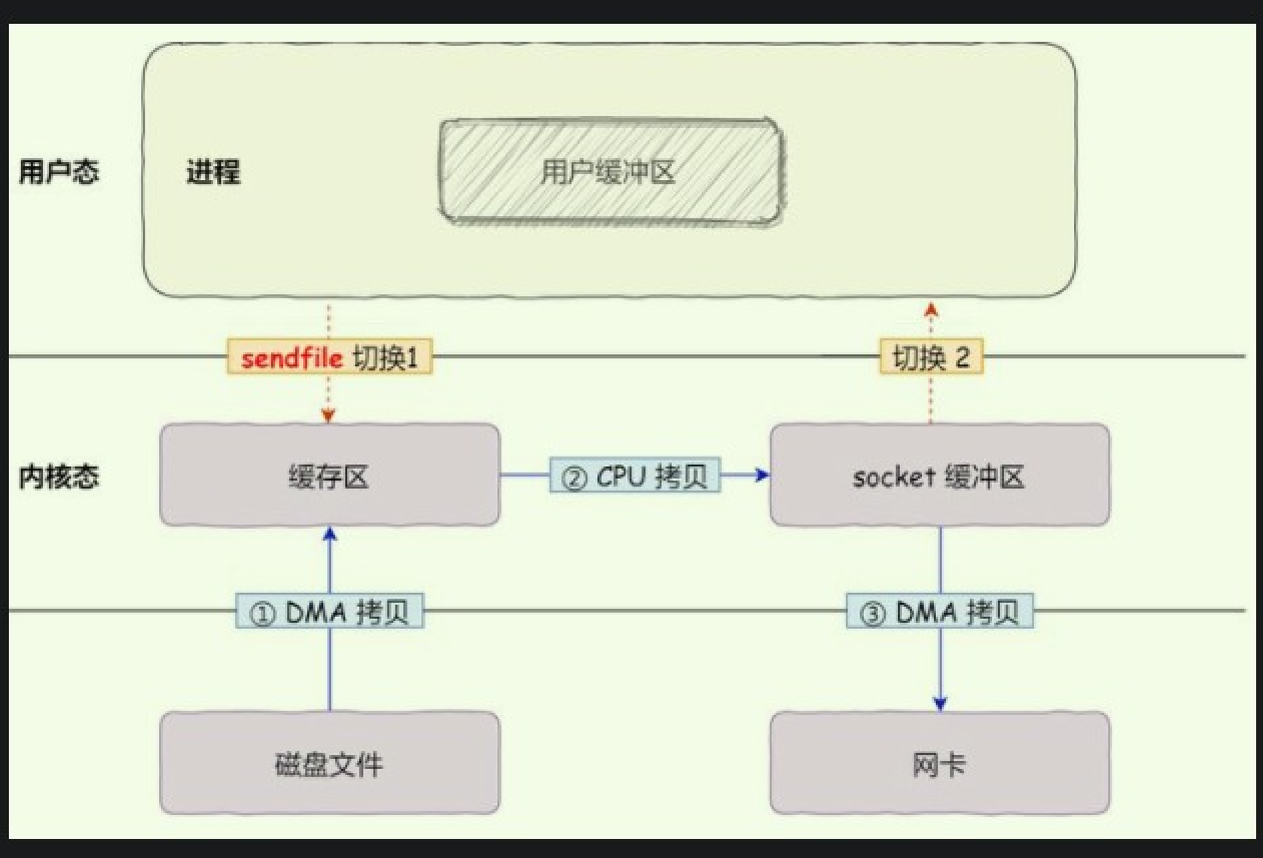
如上图所示，mmap技术传输文件，需要进行**4次用户态和内核态切换，3次数据拷贝（1次CPU拷贝、两次DMA拷贝）**

相对于传统数据传输，mmap减少了一次CPU拷贝，其具体过程如下：

1. 应用进程调用 mmap()，DMA 会把磁盘的数据拷贝到内核的缓冲区里，应用进程跟操作系统内核「共享」这个缓冲区
2. 应用进程再调用 write()，操作系统直接将内核缓冲区的数据拷贝到 socket 缓冲区中，这一切都发生在内核态，由 CPU 来搬运数据
3. 最后，把内核的 socket 缓冲区里的数据，拷贝到网卡的缓冲区里，这个过程是由 DMA 搬运的

显然仅仅减少一次数据拷贝，依然难以满足要求

sendfile



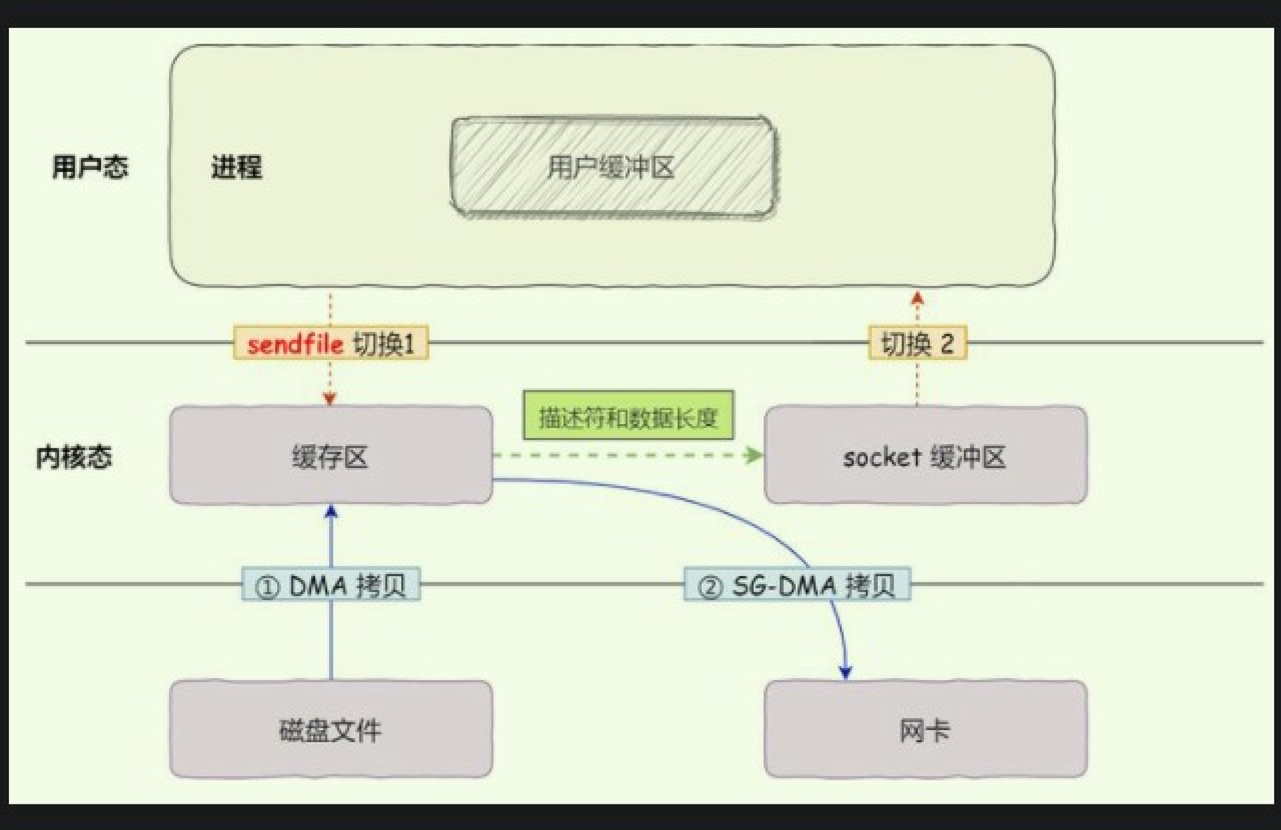
如上图所属，sendfile技术传输文件，需要进行**2次用户态和内核态的切换，3次数据拷贝（1次CPU拷贝、两次DMA拷贝）**

相对于mmap，其又减少了两次上下文的切换，具体过程如下：

1. 应用调用sendfile接口，传入文件描述符，应用程序切换至内核态，并通过 DMA 将磁盘上的数据拷贝到内核缓冲区中
2. CPU将缓冲区数据拷贝至Socket缓冲区
3. DMA将数据拷贝到网卡的缓冲区里，应用程序切换至用户态

sendfile其实是将原来的两步读写操作进行了合并，从而减少了2次上下文的切换，但其仍然不是真正意义上的“零”拷贝

sendfile + SG-DMA



从 Linux 内核 `2.4` 版本开始起，对于支持网卡支持 SG-DMA 技术的情况下，`sendfile()` 系统调用的过程发生了点变化，如上图所示，sendfile + SG-DMA技术传输文件，需要进行**2次用户态和内核态的切换，2次数据拷贝（1次DMA拷贝，1次SG-DMA拷贝）**

具体过程如下：

1. 通过 DMA 将磁盘上的数据拷贝到内核缓冲区里；
2. 缓冲区描述符和数据长度传到 socket 缓冲区，这样网卡的 SG-DMA 控制器就可以直接将内核缓存中的数据拷贝到网卡的缓冲区里，此过程不需要将数据从操作系统内核缓冲区拷贝到 socket 缓冲区中，这样就减少了一次数据拷贝；

此种方式对比之前的，真正意义上去除了CPU拷贝，CPU 的高速缓存再不会被污染了，CPU 可以去执行其他的业务计算任务，同时和 DMA 的 I/O 任务并行，极大地提升系统性能。

但他的劣势也很明显，强依赖于硬件的支持

splice

Linux 在 2.6.17 版本引入 splice 系统调用，不再需要硬件支持，同时还实现了两个文件描述符之间的数据零拷贝。

splice 系统调用可以在内核空间的读缓冲区（read buffer）和网络缓冲区（socket buffer）之间建立管道（pipeline），从而避免了用户缓冲区和Socket缓冲区的 CPU 拷贝操作。

基于 splice 系统调用的零拷贝方式，整个拷贝过程会发生 **2次用户态和内核态的切换，2次数据拷贝（2次DMA拷贝）**，具体过程如下：

1. 用户进程通过 splice() 函数向内核（kernel）发起系统调用，上下文从用户态（user space）切换为内核态（kernel space）。
2. CPU 利用 DMA 控制器将数据从主存或硬盘拷贝到内核空间（kernel space）的读缓冲区（read buffer）。
3. CPU 在内核空间的读缓冲区（read buffer）和网络缓冲区（socket buffer）之间建立管道（pipeline）。
4. CPU 利用 DMA 控制器将数据从网络缓冲区（socket buffer）拷贝到网卡进行数据传输。
5. 上下文从内核态（kernel space）切换回用户态（user space），splice 系统调用执行返回。

splice 拷贝方式也同样存在用户程序不能对数据进行修改的问题。除此之外，它使用了 Linux 的管道缓冲机制，可以用于任意两个文件描述符中传输数据，但是它的两个文件描述符参数中有一个必须是管道设备

总结

本文简单介绍了 Linux 中的几种 Zero-copy 技术，随着技术的不断发展，又出现了诸如：写时复制、共享缓冲等技术，本文就不再赘述。

广义的来讲，Linux 的 Zero-copy 技术可以归纳成以下三大类：

- **减少甚至避免用户空间和内核空间之间的数据拷贝**：在一些场景下，用户进程在数据传输过程中并不需要对数据进行访问和处理，那么数据在 Linux 的 `Page Cache` 和用户进程的缓冲区之间的传输就完全可以避免，让数据拷贝完全在内核里进行，甚至可以通过更巧妙的方式避免在内核里的数据拷贝。这一类实现一般是是通过增加新的系统调用来完成的，比如 Linux 中的 mmap()，sendfile() 以及 splice() 等。
- **绕过内核的直接 I/O**：允许在用户态进程绕过内核直接和硬件进行数据传输，内核在传输过程中只负责一些管理和辅助的工作。这种方式其实和第一种有点类似，也是试图避免用户空间和内核空间之间的数据传输，只是第一种方式是把数据传输过程放在内核态完成，而这种方式则是直接绕过内核和硬件通信，效果类似但原理完全不同。
- **内核缓冲区和用户缓冲区之间的传输优化**：这种方式侧重于在用户进程的缓冲区和操作系统的页缓存之间的 CPU 拷贝的优化。这种方法延续了以往那种传统的通信方式，但更灵活。

分类： IO, Linux

标签： Java, Linux

« 上一篇： Netty入门（三）：EventLoop

posted @ 2021-09-06 20:15 新人十三 阅读(358) 评论(0) 编辑 收藏 举报

我的标签
Java(5)
Linux(3)
网络编程(3)
Netty(3)
Nacos(1)
SpringCloud Alibaba(1)
SpringCloud(1)

随笔分类
IO(2)
Linux(2)
Netty(3)
SpringCloud(1)

随笔档案
2021年9月(1)
2021年8月(6)

阅读排行榜
1. Nacos 自动更新配置不生效问题(689)
2. 深入理解零拷贝技术(358)
3. Netty入门（一）：ByteBuf(289)
4. Netty入门（三）：EventLoop(190)
5. Java基础（一）：I/O多路复用模型及Linux中的应用(89)

评论排行榜
1. Netty入门（三）：EventLoop(1)

推荐排行榜
1. 深入理解零拷贝技术(1)
2. Netty入门（三）：EventLoop(1)

最新评论
1. Re:Netty入门（三）：EventLoop
###请问博主这个排版是markdown，很美观，我也想学一下
--懂得了才能做一些改变