链滴

零拷贝的原理

作者：vcjmhg

原文链接：https://ld246.com/article/1627440922239

来源网站：链滴

概述

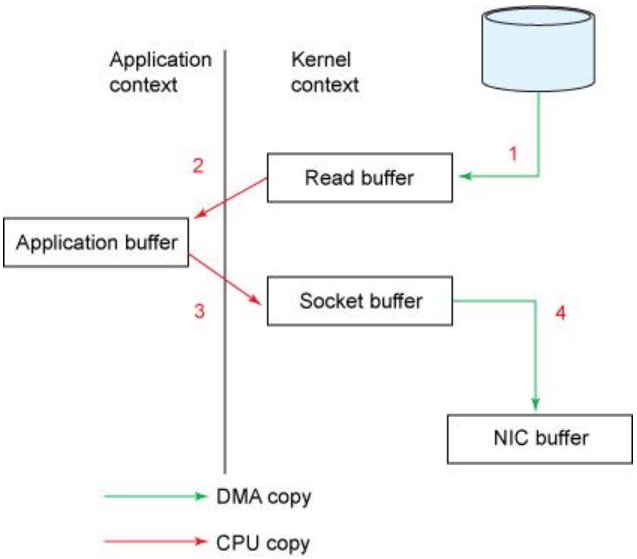
如果学习过Kafka的小伙伴，想必对零拷贝技术并不陌生，Kafka对Producer和Consumer能有这么的处理能力，很大程度上就是依赖于对零拷贝的支持。零拷贝是什么呢？它和传统模式有什么区别呢我们该如何用呢？接下来这篇文章将会就这些问题给您娓娓道来。

传统方式

在我们编写程序将从磁盘中读取数据发送到网络上时，很容易写出下边这段代码：

```
File.read(fileDesc, buf, len);
Socket.send(socket, buf, len);
```

这一过程实发生了**四次数据拷贝**：



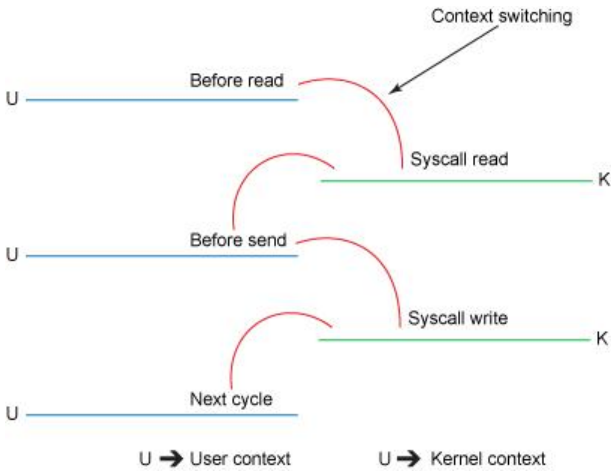
原文链接：零拷贝的原理

1. 第一次将数据从磁盘**“拷贝”**到内核缓存中
2. 第二次将数据从内核缓存中**“拷贝”**到用户缓存中
3. 第三次将数据从用户缓存**“拷贝”**到Socket缓存中
4. 第四次将数据Socket缓存**“拷贝”**到网卡缓存中，最后网卡将数据发送到网络空间上。

为什么会频繁的在内核缓存和用户缓存之间“拷贝”呢?

这是因为操作系统在设计之初基于安全性的考量，将内存分成了两部分：一部分是内核空间（kernel-pace）对应内核态，一部分是用户空间（user-space）对应用户态，一些敏感性的操作比如对底层件设备的操作必须在内核态下完成。

而且这个过程中也伴随着四次上下文的切换：



1. 程序在调用 read()方法之前肯定处于用户态，当执行read()时，程序将会从**用户态转成核心态。**
2. read()执行完成后，程序将会从**核心态->用户态**，并返回执行结果
3. 程序在调用 send()方法时，会从**用户态->核心态**
4. send()方法执行完成后，程序将会从**核心态->用户态**，发送完成

毫无意义，多次的数据拷贝和上下文的转化必然会带来很大的时间开销，我们有什么办法来减少非必的拷贝（比如用户态和核心态的切换）以及上下文的切换呢?

零拷贝

这就引入了我们今天说主角，**零拷贝**技术。

零拷贝（Zero-copy）技术指在计算机执行操作时，CPU 不需要先将数据从一个内存区域复制到另一

原文链接：零拷贝的原理

内存区域，从而可以减少上下文切换以及 CPU 的拷贝时间。

它的作用是在数据报从网络设备到用户程序空间传递的过程中，减少数据拷贝次数，减少系统调用，现 CPU 的零参与，彻底消除 CPU 在这方面的负载。

简单来说，零拷贝技术就是通过减少内存间（用户空间和内核空间）**非必要的拷贝**，从而提高速度的术。

目前零拷贝的技术主要有三种：

- **直接IO**：数据直接跨过内核，在用户地址空间与 I/O 设备之间传递，内核只是进行必要的虚拟存储置等辅助工作，比较常用的场景是在数据库上。
- **避免内核和用户空间之间的数据拷贝**：
  - mmap
  - sendfile
  - splice && tee
  - sockemap
- **Copy on Write**：是一种写时拷贝技术或者说是一种思想，数据不需要提前拷贝，而是当需要修改时候再进行部分拷贝。可参考另一篇文章《"Copy-on-Write模式"》

其中我们在Java中常用应该是mmap技术，比如在Kafka中就是通过mmap技术实现的。

什么是mmap技术呢?

Memory Mapped Files：简称 mmap，也有叫 MMFile 的，使用 mmap 的目的是将内核中读缓冲（read buffer）的地址与用户空间的缓冲区（user buffer）进行映射。从而实现内核缓冲区与应用序内存的共享，省去了将数据从内核读缓冲区（read buffer）拷贝到用户缓冲区（user buffer）的程。它的工作原理是直接利用操作系统的 Page 来实现文件到物理内存的直接映射。完成映射之后你物理内存的操作会被同步到硬盘上。

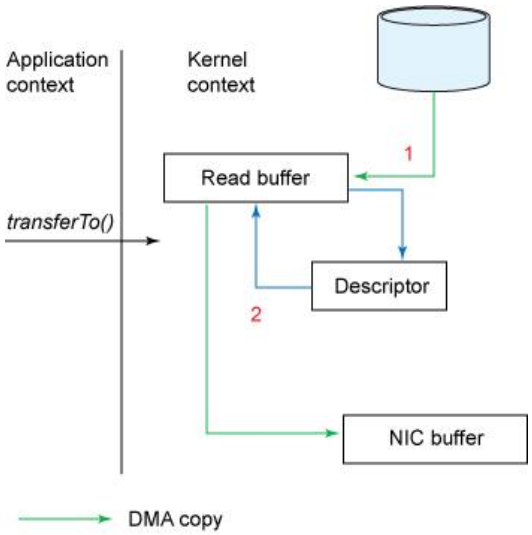
使用这种方式可以获取很大的 I/O 提升，省去了用户空间到内核空间复制的开销。

简单一句话总结，mmap通过将**用户态空间核内核空间进行映射**，省去了内核缓冲区和用户缓冲区之的内存“拷贝”进而提高文件传输速度的一种技术。

具体在使用的时候可以通过nio中transferTo方法（内部是通过系统调用实现的）直接将数据从磁盘送到网络空间上，具体代码：

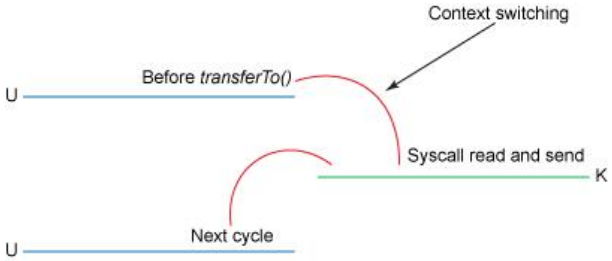
```
transferTo(position, count, writableChannel);
```

也就是说通过零拷贝技术，我们将前面的四次拷贝过程，变成了下边的两次：



1. 第一次，是通过 DMA，从硬盘直接读到操作系统内核的读缓冲区(mmap技术)里面。
2. 第二次，则是根据 Socket 的描述符信息，直接从读缓冲区里面，写入到网卡的缓冲区里面。

同时上下文切换也由原来的四种变成了现在的两种：



1. transferTo()执行前处于用户态，执行时转成内核态
2. transferTo()执行完成之后，再由内核态转成用户态

显而易见，通过零拷贝技术，我们可以减少两次内存间的拷贝操作以及两次上下文转换的操作可以大提高文件传输的速度。

我们这里借用一种IBM一篇名为《Efficient data transfer through zero copy》的图来看一下引入零贝技术之后一个提升效果。

效果对比图

文件大小	传统方式**(ms)**	tranferTo (ms)
7MB	156	45
21MB	337	128
63MB	843	387
98MB	1320	617
200MB	2124	1150
350MB	3631	1762
700MB	13498	4422

原文链接：零拷贝的原理

1GB	18399	8537

很明显引入了零拷贝技术之后大约缩短了65%的时间，速度大大加快。

应用场景（以transferTo为例）

其实从transferTo()原理上看，对于将大量数据从一个 I/O 通道复制到另一个通道的情况（例如 Web 服务器），都是合适的。而对于磁盘文件间的复制，比如从一个磁盘位置复制到另一个磁盘位置，这方式是不可用的。

总结

零拷贝技术，本质上讲就是通过减少非必要的内存拷贝以及上下文切换，来提高文件在通道间复制速的一种技术。以本文中的transferTo()方法为例，通过该技术，可以将原来四次内存间拷贝减少成两，将四次上下文切换减少成两次，大大提高复制的速度。但零拷贝技术并非万能的，它有自己的使用景，对于将大量数据从一个 I/O 通道复制到另一个通道的情况（例如 Web 服务器），都是合适的。对于磁盘文件间的复制，比如从一个磁盘位置复制到另一个磁盘位置，这种方式是不可用的。

参考

1. 《Efficient data transfer through zero copy》
2. 《终于知道 Kafka 为什么这么快了！》
3. 《DMA：为什么Kafka这么快？》