## **前言**

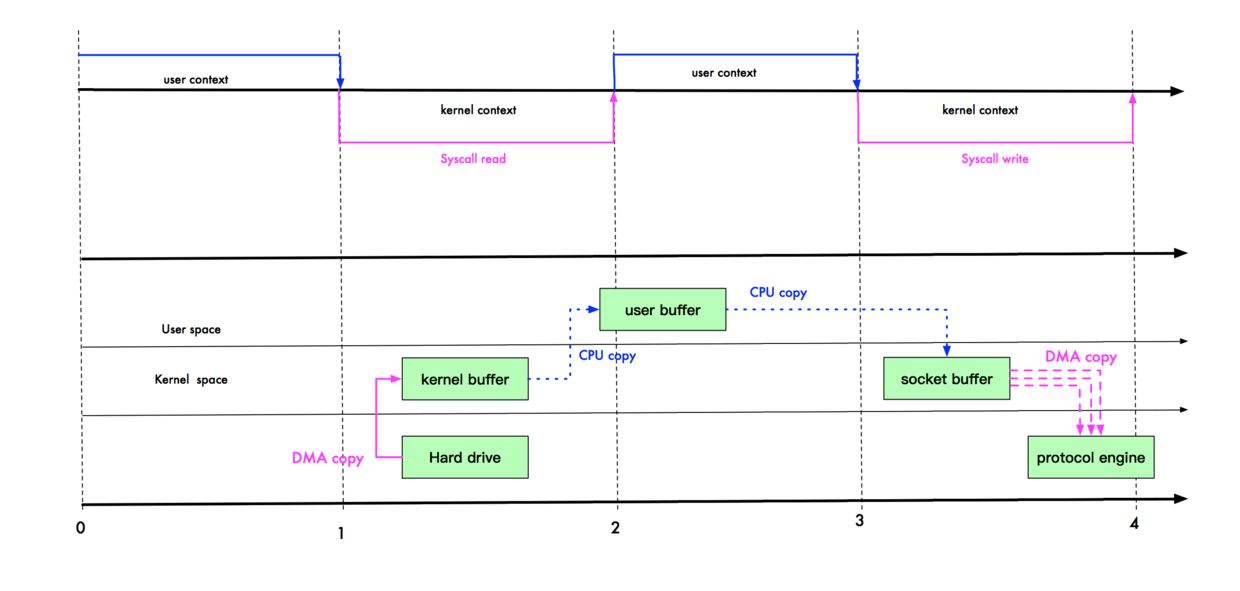
零拷贝是服务器网络编程的关键，任何性能优化都离不开。在 Java 程序员的世界，常用的零拷贝有 mmap 和 sendFile。那么，他们在 OS 里，到底是怎么样的一个的设计？本文将简单聊聊 mmap 和 sendFile 这两个零拷贝。

## **传统数据读写的劣势**

初学 Java 时，我们在学习 IO 和 网络编程时，会使用以下代码：

|  |
| --- |
| File file = new File("index.html");  RandomAccessFile raf = new RandomAccessFile(file, "rw");  byte[] arr = new byte[(int) file.length()];  raf.read(arr);  Socket socket = new ServerSocket(8080).accept();  socket.getOutputStream().write(arr); |

我们会调用 read 方法读取 index.html 的内容—— 变成字节数组，然后调用 write 方法，将 index.html 字节流写到 socket 中，那么，我们调用这两个方法，在 OS 底层发生了什么呢？我这里借鉴了一张其他文字的图片，尝试解释这个过程。



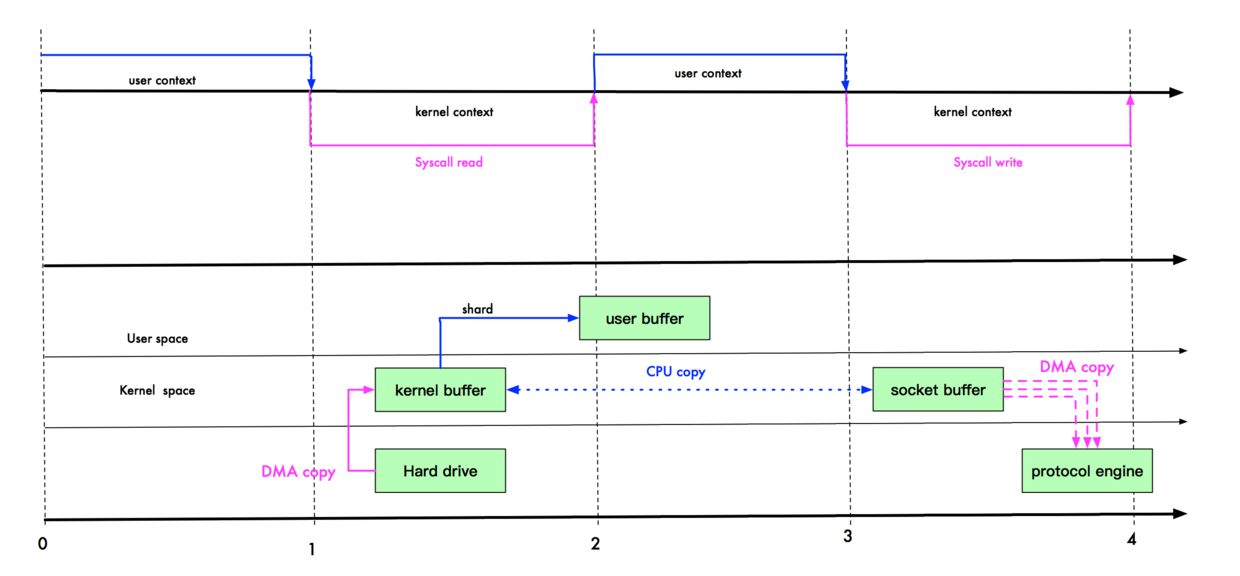
上图中，上半部分表示用户态和内核态的上下文切换。下半部分表示数据复制操作。下面说说他们的步骤：

1. read 调用导致用户态到内核态的一次变化，同时，第一次复制开始：DMA（Direct Memory Access，直接内存存取，即不使用 CPU 拷贝数据到内存，而是 DMA 引擎传输数据到内存，用于解放 CPU） 引擎从磁盘读取 index.html 文件，并将数据放入到内核缓冲区。
2. 发生第二次数据拷贝，即：将内核缓冲区的数据拷贝到用户缓冲区，同时，发生了一次用内核态到用户态的上下文切换。
3. 发生第三次数据拷贝，我们调用 write 方法，系统将用户缓冲区的数据拷贝到 Socket 缓冲区。此时，又发生了一次用户态到内核态的上下文切换。
4. 第四次拷贝，数据异步的从 Socket 缓冲区，使用 DMA 引擎拷贝到网络协议引擎。这一段，不需要进行上下文切换。
5. write 方法返回，再次从内核态切换到用户态。

如你所见，复制拷贝操作太多了。如何优化这些流程？

## **mmap 优化**

mmap 通过内存映射，将文件映射到内核缓冲区，同时，用户空间可以共享内核空间的数据。这样，在进行网络传输时，就可以减少内核空间到用户控件的拷贝次数。如下图：

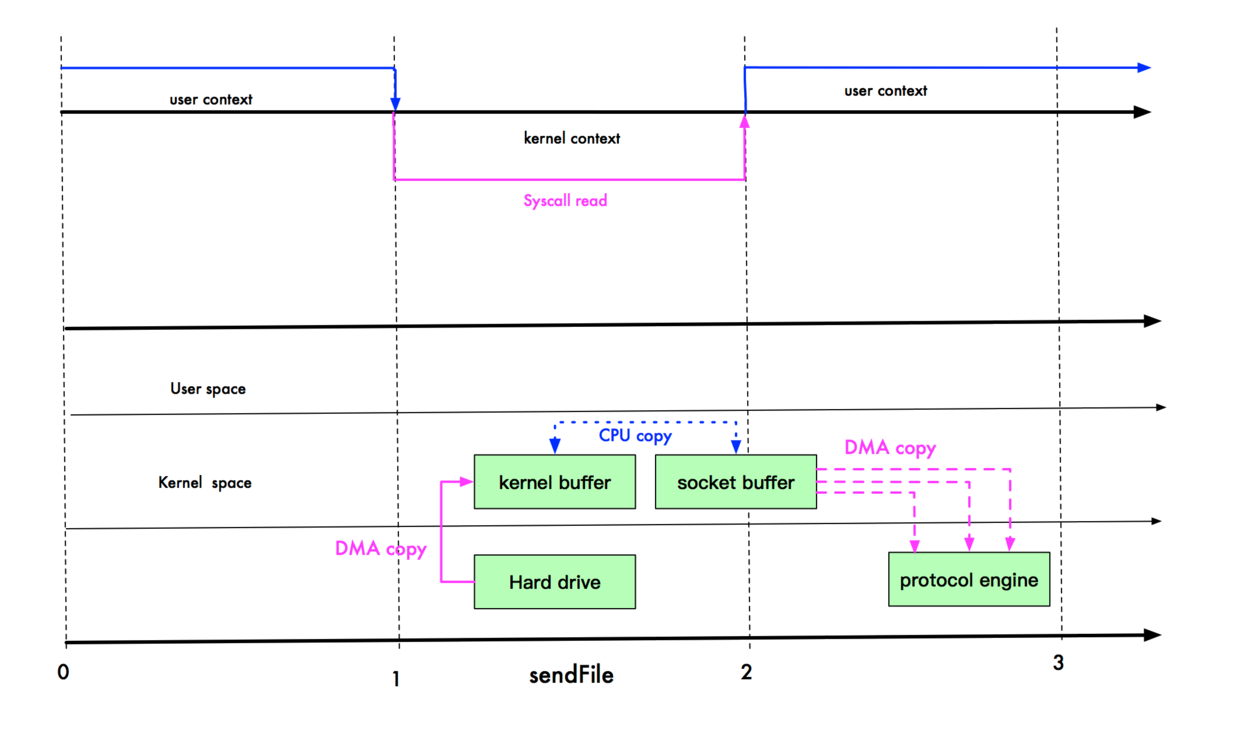


如上图，user buffer 和 kernel buffer 共享 index.html。如果你想把硬盘的 index.html 传输到网络中，再也不用拷贝到用户空间，再从用户空间拷贝到 Socket 缓冲区。

现在，你只需要从内核缓冲区拷贝到 Socket 缓冲区即可，这将减少一次内存拷贝（从 4 次变成了 3 次），但不减少上下文切换次数。

## **sendFile**

那么，我们还能继续优化吗？ Linux 2.1 版本 提供了 sendFile 函数，其基本原理如下：数据根本不经过用户态，直接从内核缓冲区进入到 Socket Buffer，同时，由于和用户态完全无关，就减少了一次上下文切换。



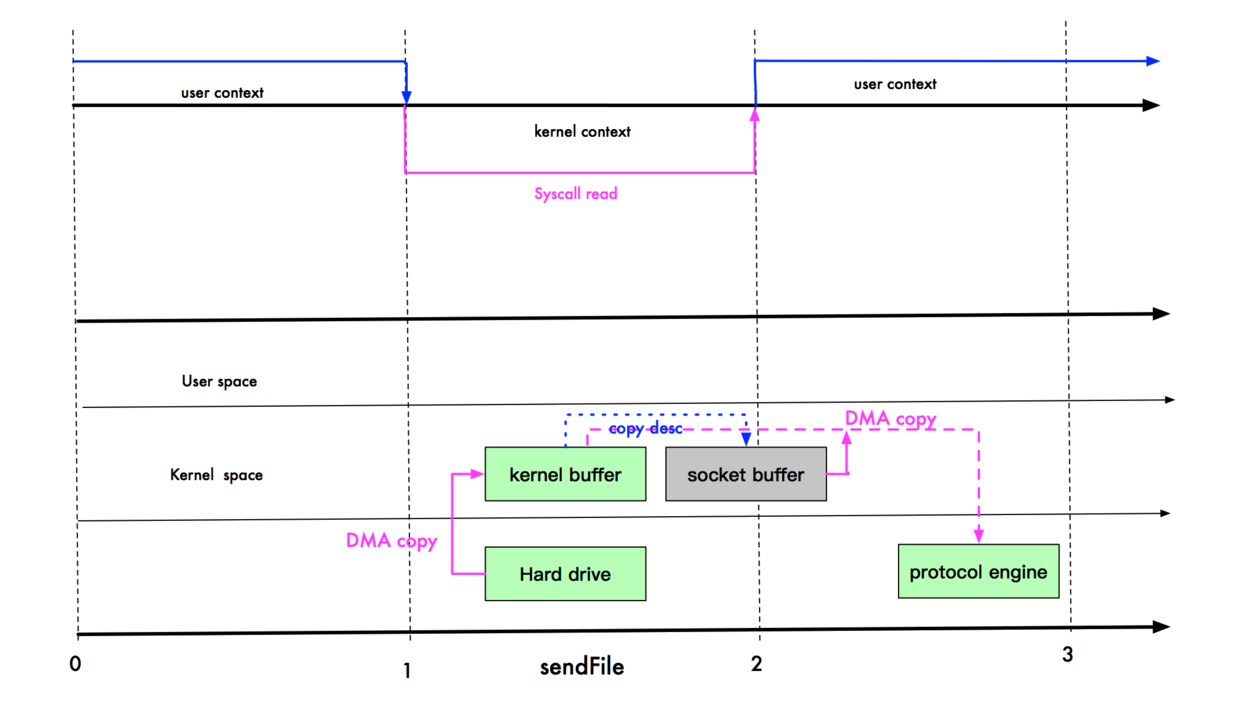
如上图，我们进行 sendFile 系统调用时，数据被 DMA 引擎从文件复制到内核缓冲区，然后调用，然后掉一共 write 方法时，从内核缓冲区进入到 Socket，这时，是没有上下文切换的，因为在一个用户空间。

最后，数据从 Socket 缓冲区进入到协议栈。

此时，数据经过了 3 次拷贝，3 次上下文切换。

那么，还能不能再继续优化呢？ 例如直接从内核缓冲区拷贝到网络协议栈？

实际上，Linux 在 2.4 版本中，做了一些修改，避免了从内核缓冲区拷贝到 Socket buffer 的操作，直接拷贝到协议栈，从而再一次减少了数据拷贝。具体如下图：



现在，index.html 要从文件进入到网络协议栈，只需 2 次拷贝：第一次使用 DMA 引擎从文件拷贝到内核缓冲区，第二次从内核缓冲区将数据拷贝到网络协议栈；内核缓存区只会拷贝一些 offset 和 length 信息到 SocketBuffer，基本无消耗。

等一下，不是说零拷贝吗？为什么还是要 2 次拷贝？

答：首先我们说零拷贝，是从操作系统的角度来说的。因为内核缓冲区之间，没有数据是重复的（只有 kernel buffer 有一份数据，sendFile 2.1 版本实际上有 2 份数据，算不上零拷贝）。例如我们刚开始的例子，内核缓存区和 Socket 缓冲区的数据就是重复的。

而零拷贝不仅仅带来更少的数据复制，还能带来其他的性能优势，例如更少的上下文切换，更少的 CPU 缓存伪共享以及无 CPU 校验和计算。

再稍微讲讲 mmap 和 sendFile 的区别。

1. mmap 适合小数据量读写，sendFile 适合大文件传输。
2. mmap 需要 4 次上下文切换，3 次数据拷贝；sendFile 需要 3 次上下文切换，最少 2 次数据拷贝。
3. sendFile 可以利用 DMA 方式，减少 CPU 拷贝，mmap 则不能（必须从内核拷贝到 Socket 缓冲区）。

在这个选择上：rocketMQ 在消费消息时，使用了 mmap。kafka 使用了 sendFile。