什么是Zero-Copy?

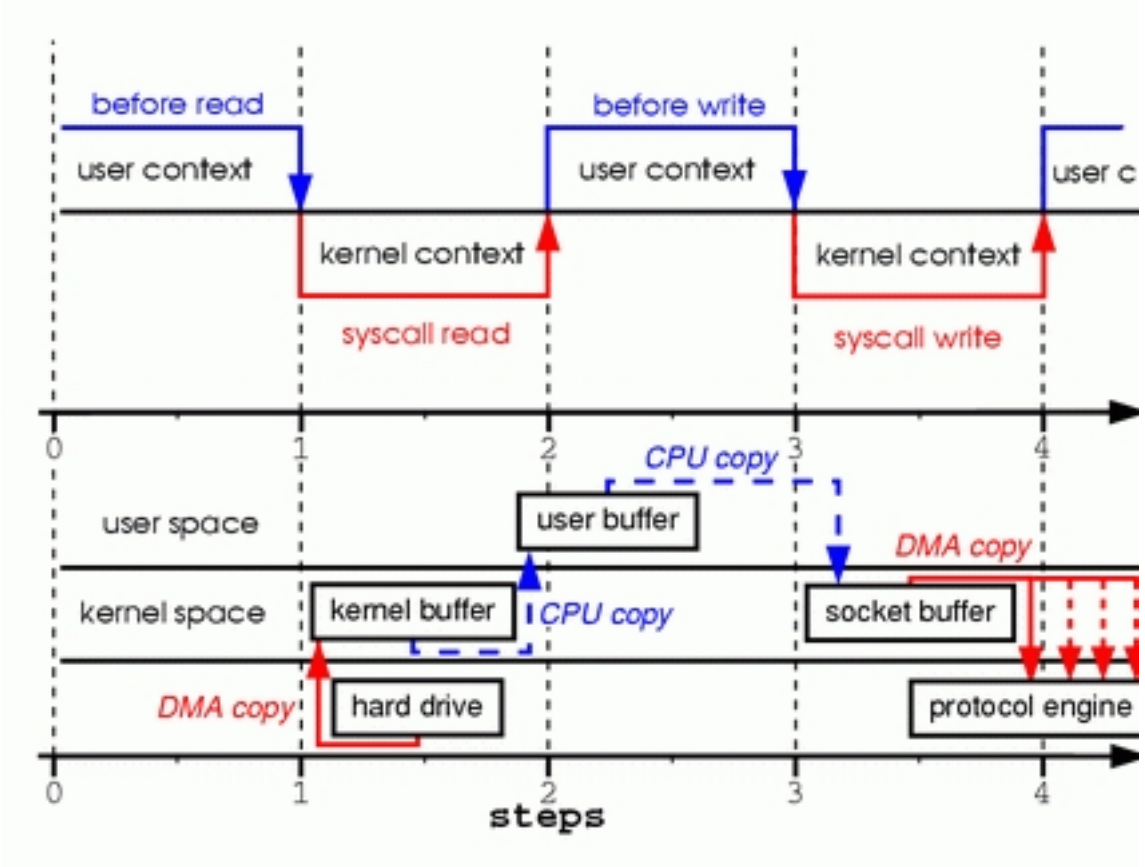
概述

考虑这样一种常用的情形：你需要将静态内容（类似图片、文件）展示给用户。那么这个情形就意味着你需要先将静态内容从磁盘中拷贝出来放到一个内存buf中，然后将这个buf通过socket传输给用户，进而用户或者静态内容的展示。这看起来再正常不过了，但是实际上这是很低效的流程，我们把上面的这种情形抽象成下面的过程：

```
read(file, tmp_buf, len);
write(socket, tmp_buf, len);
```

- 1
- 2

首先调用read将静态内容，这里假设为文件A，读取到tmp_buf, 然后调用write将tmp_buf写入到socket中，如图：



在这个过程中文件A的经历了4次copy的过程：

1. 首先，调用read时，文件A拷贝到了kernel模式；
2. 之后，CPU控制将kernel模式数据copy到user模式下；
3. 调用write时，先将user模式下的内容copy到kernel模式下的socket的

buffer中；

4. 最后将kernel模式下的socket buffer的数据copy到网卡设备中传送；

从上面的过程可以看出，数据白白从kernel模式到user模式走了一圈，浪费了2次copy(第一次，从kernel模式拷贝到user模式；第二次从user模式再拷贝回kernel模式，即上面4次过程的第2和3步骤。)。而且上面的过程中kernel和user模式的上下文的切换也是4次。

幸运的是，你可以用一种叫做Zero-Copy的技术来去掉这些无谓的copy。应用程序用Zero-Copy来请求kernel直接把disk的data传输给socket，而不是通过应用程序传输。Zero-Copy大大提高了应用程序的性能，并且减少了kernel和user模式上下文的切换。

详述

Zero-Copy技术省去了将操作系统的read buffer拷贝到程序的buffer，以及从程序buffer拷贝到socket buffer的步骤，直接将read buffer拷贝到socket buffer. Java NIO中的FileChannal.transferTo()方法就是这样的实现，这个实现是依赖于操作系统底层的sendFile()实现的。

```
public void transferTo(long position, long count, WritableByteChannel targe
```

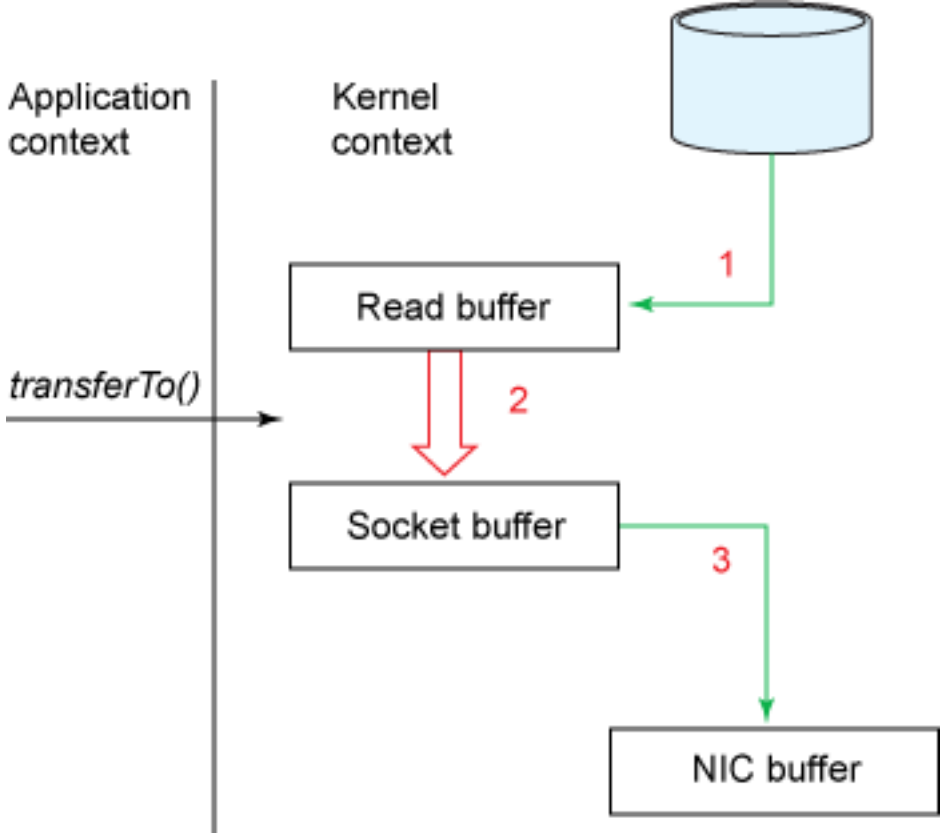
- 1

他底层的调用时系统调用sendFile()方法：

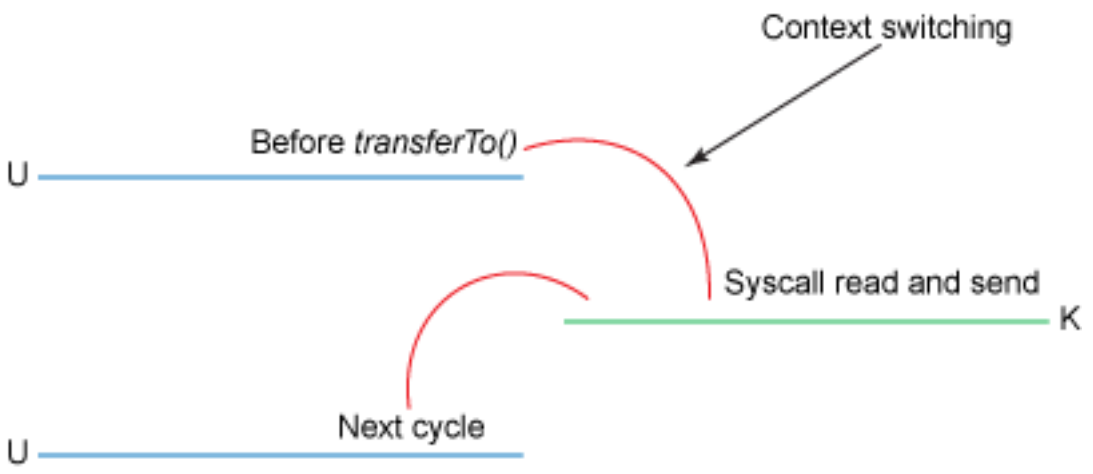
```
#include <sys/socket.h>
ssize_t sendfile(int out_fd, int in_fd, off_t *offset, size_t count);
```

- 1
- 2

下图展示了在transferTo()之后的数据流向：



下图展示了在使用transferTo()之后的上下文切换：



使用了Zero-Copy技术之后，整个过程如下：

1. transferTo()方法使得文件A的内容直接拷贝到一个read buffer（kernel buffer）中；
2. 然后数据(kernel buffer)拷贝到socket buffer中。
3. 最后将socket buffer中的数据拷贝到网卡设备（protocol engine）中传输；
   这显然是一个伟大的进步：这里把上下文的切换次数从4次减少到2次，同时也把数据copy的次数从4次降低到了3次。

但是这是Zero-Copy么，答案是否定的。

进阶

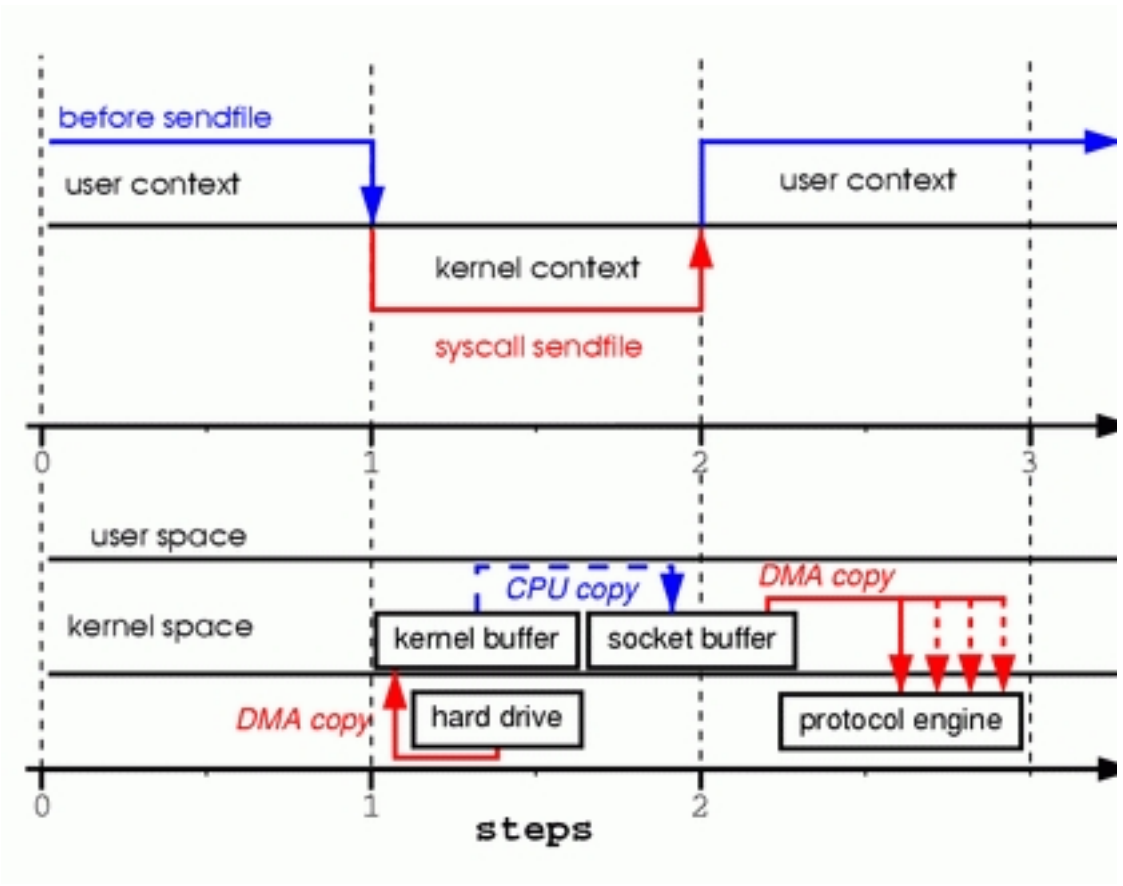
Linux 2.1内核开始引入了sendfile函数（上一节有提到）,用于将文件通过

socket传送。

```
sendfile(socket, file, len);
```

- 1

该函数通过一次系统调用完成了文件的传送，减少了原来read/write方式的模式切换。此外更是减少了数据的copy, sendfile的详细过程如图：



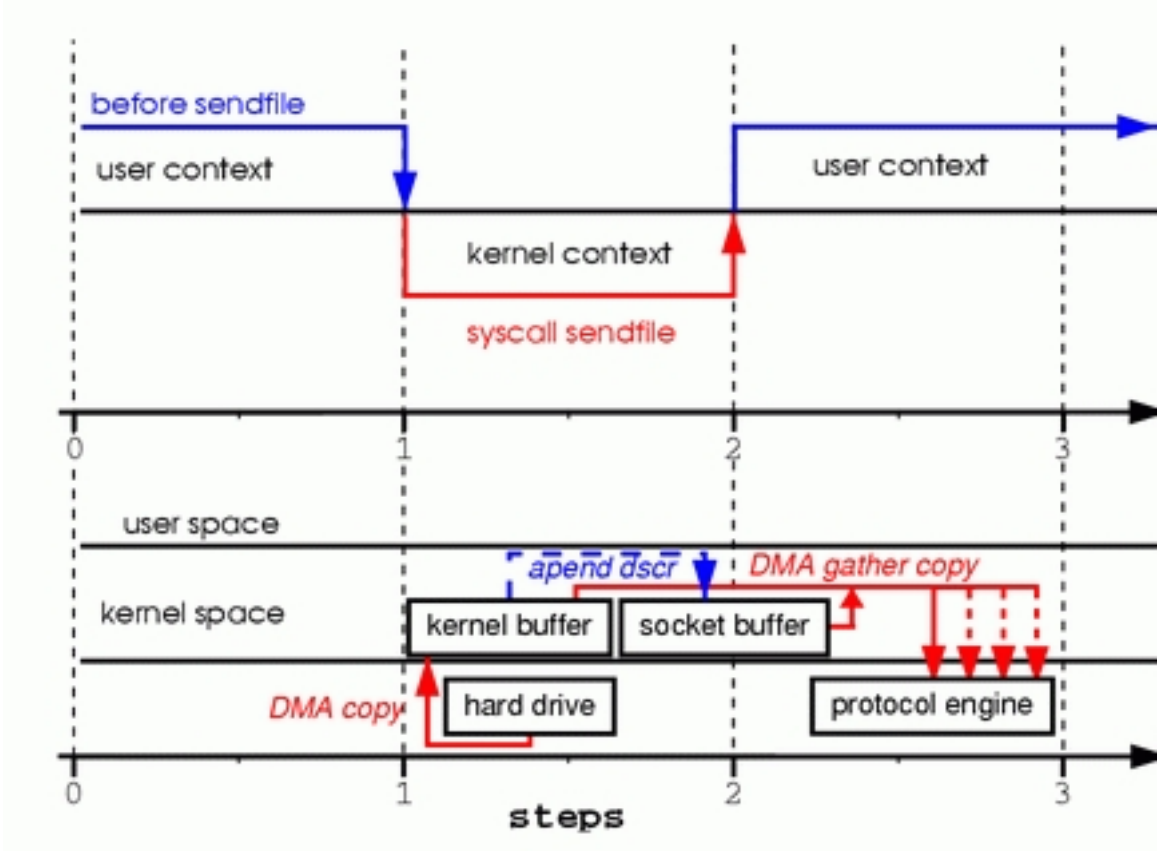
通过sendfile传送文件只需要一次系统调用，当调用sendfile时：

1. 首先（通过DMA）将数据从磁盘读取到kernel buffer中；
2. 然后将kernel buffer拷贝到socket buffer中；
3. 最后将socket buffer中的数据copy到网卡设备（protocol engine）中发送；

这个过程就是第二节（详述）中的那个步骤。

sendfiel与read/write模式相比，少了一次copy。但是从上述过程中也可以发现从kernel buffer中将数据copy到socket buffer是没有必要的。

Linux2.4 内核对sendfile做了改进，如图：

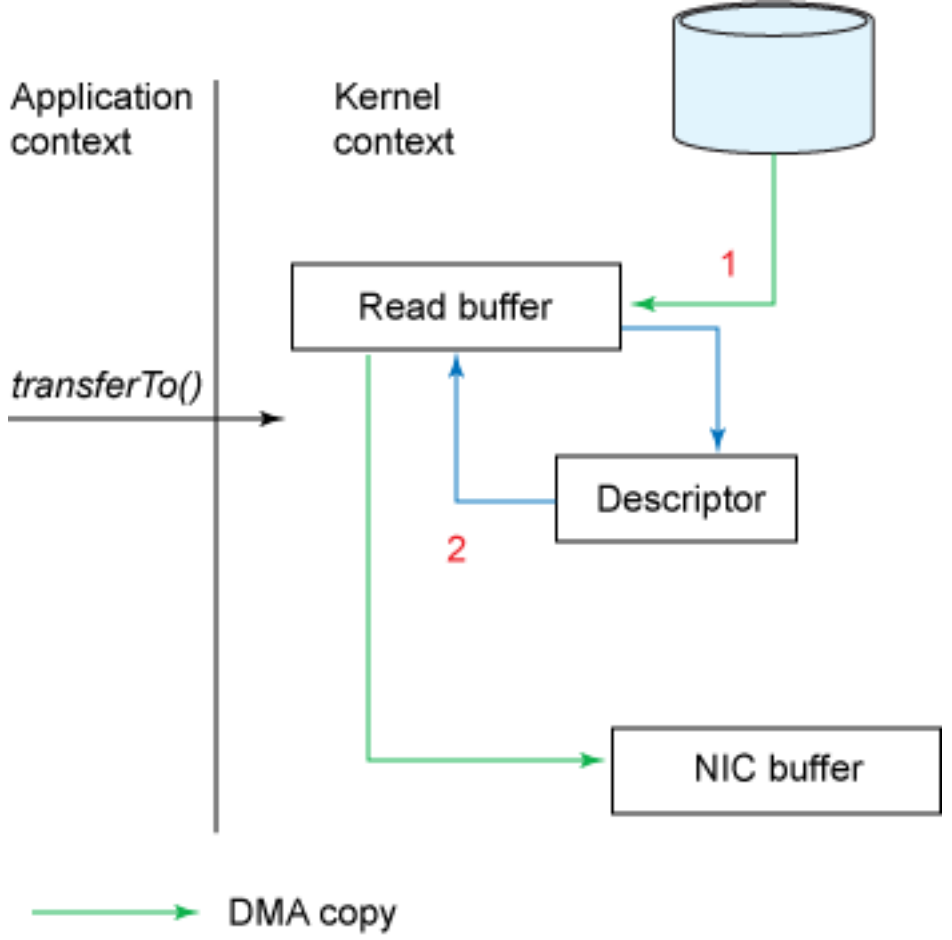


改进后的处理过程如下：

1. 将文件拷贝到kernel buffer中；
2. 向socket buffer中追加当前要发生的数据在kernel buffer中的位置和偏移量；
3. 根据socket buffer中的位置和偏移量直接将kernel buffer的数据copy到网卡设备（protocol engine）中；

经过上述过程，数据只经过了2次copy就从磁盘传送出去了。
这个才是真正的Zero-Copy(这里的零拷贝是针对kernel来讲的，数据在kernel模式下是Zero-Copy)。

正是Linux2.4的内核做了改进，Java中的TransferTo()实现了Zero-Copy,如下图：



Zero-Copy技术的使用场景有很多，比如Kafka, 又或者是Netty等，可以大大提升程序的性能。

参考资料

1. Zero-Copy&sendfile浅析
2. Efficient data transfer through zero copy
3. Kafka Zero-Copy 使用分析
4. 理解Netty中的零拷贝（Zero-Copy）机制

欢迎支持笔者新书：《RabbitMQ实战指南》以及关注微信公众号：朱小厮的博客。