通过zero copy来实现高效的数据传输

这段时间在学习一些系统底层的知识，真后悔大学没有好好学习操作系统，导致好多文章看不懂。说到这不得不吐槽一下，像介绍系统层次的一些书籍好多都是中文翻译版，而大部分的中文翻译版大都语句晦涩，难懂，而且极易被误导。网上也有一些介绍文章，好多是连作者自己都没搞明白抑或是简单的复制粘贴，真是越看越迷糊。当然不乏有好的翻译书籍。不仅仅是我个人，好多大牛也都建议这样的书籍直接读英文原版好一些。有英语问题也没办法，程序员学习能力强，英语不行就干英语^_^。再回到主题，看到一篇不错的文章，然后推荐给朋友，朋友一看英文的就懒得看，所以这里LZ打算将这篇文章用自己三脚猫的英语简单的翻译一下，以巩固自己的所学，同时也分享给大家。以下是正文。

这篇文章介绍了一个有大量io操作的运行在linux或者unix平台上的Java程序，如何用zero copy技术来提高IO性能。zero copy可以避免缓冲区间数据拷贝的次数，也可以减少用户态和内核态之间的的切换。

大部分web服务器都要处理大量的静态内容，而其中大部分都是从磁盘文件中读取数据然后写到socket中。这种操作对cpu的消耗是比较小的，但也是十分低效的：内核首先从磁盘文件读取数据，然后从内核空间将数据传到用户空间，应用程序又将数据从用户空间返回到内核空间然后传输给socket(如果好奇数据为何如此来回传输，请继续看下文)。实际上，应用程序就相当于是个低效的中间者，从磁盘拿数据放到socket。

每次数据在内核空间和用户空间传输就一次拷贝过程，这是需要占用一定的cpu周期和内存资源的。幸运的是你可以通过一个叫zero copy的技术来消除这些拷贝过程。使用了zero copy技术的应用程序的数据传输过程就是内核从磁盘文件读取数据直接传输到socket中，不再经过应用程序这个中间者。zero copy大大改善了应用程序的性能并且减少了用户态和内核态之间的切换次数。

在linux或者unix系统上，Java类库通过java.nio.channels.FileChannel的transferTo()方法来应用zero copy。你可以通过这个方法把一个channel中读取到的字节传输到另一个channel，不再需要数据流经应用程序。在这篇文章中，我们首先展示了使用传统数据复制方式的一些情况，然后又通过

transferTo来使用zero copy实现一个更高性能的方式。

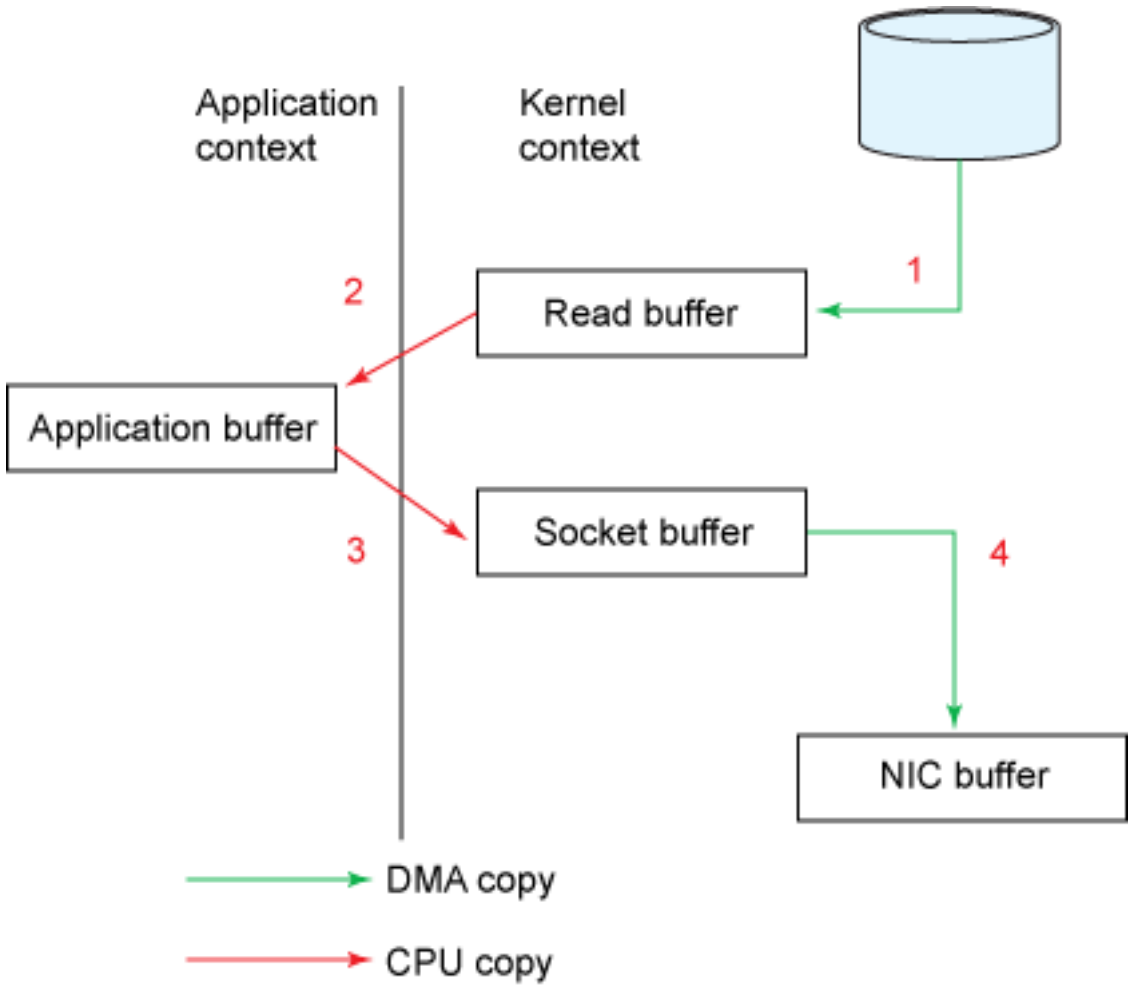
传统的数据传输方式：

像这种从文件读取数据然后将数据通过网络传输给其他的程序的方式（大部分应用服务器都是这种方式，包括web服务器处理静态内容时，ftp服务器，邮件服务器等等）其核心操作就是如下两个调用：

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

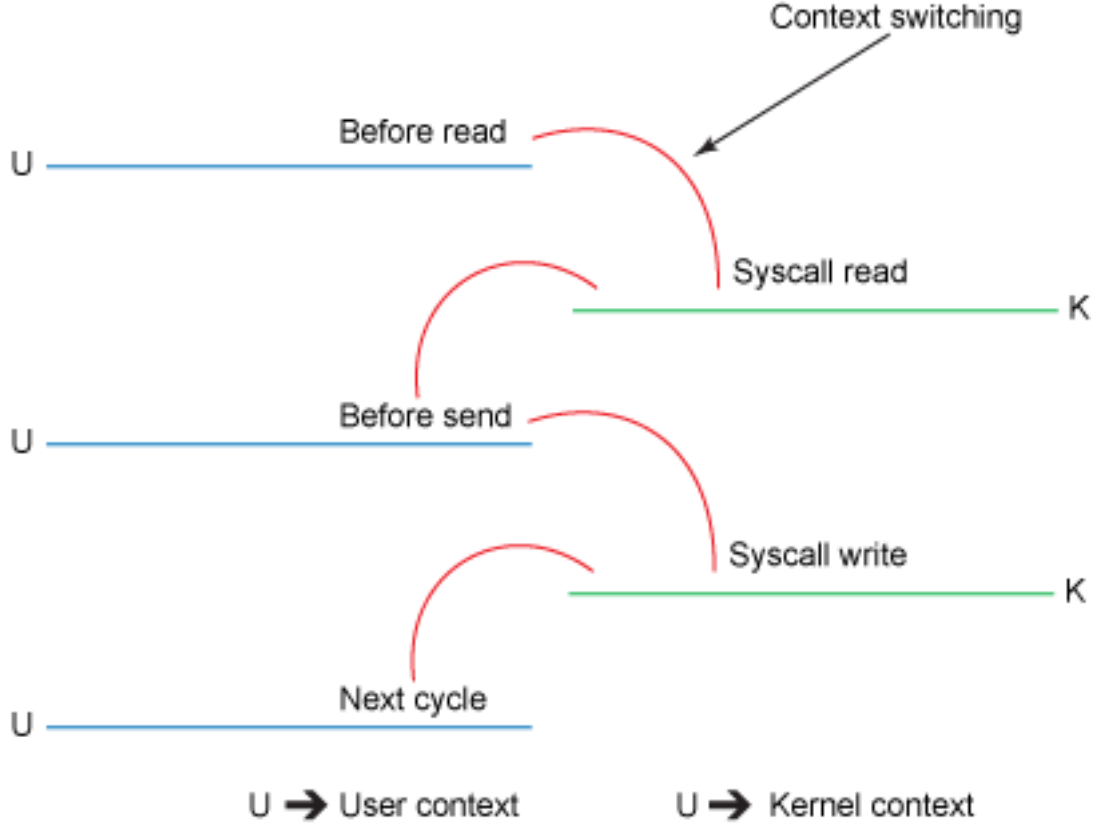
其上操作看上去只有两个简单的调用，但是其内部过程却要经历四次用户态和内核态的切换以及四次的数据复制操作。

图一展示了数据从文件到socket的内部流程：



图一,传统的数据复制方式

图二是用户态和内核态的切换过程：



图二,传统方式的上下文切换过程

这些步骤涉及到如下过程:

1、read()的调用引起了从用户态到内核态的切换（看图二），内部是通过sys_read()（或者类似的方法）发起对文件数据的读取。数据的第一次复制是通过DMA(直接内存访问)将磁盘上的数据复制到内核空间的缓冲区中。

2、数据从内核空间的缓冲区复制到用户空间的缓冲区后，read()方法也就返回了。此时内核态又切换回用户态，现在数据也已经复制到了用户地址空间的缓存中。

3、socket的send()方法的调用又会引起用户态到内核的切换，第三次数据复制又将数据从用户空间缓冲区复制到了内核空间的缓冲区，这次数据被放在了不同于之前的内核缓冲区中，这个缓冲区与数据将要被传输到的socket关联。

4、send()系统调用返回后，就产生了第四次用户态和内核态的切换。随着DMA单独异步的将数据从内核态的缓冲区中传输到协议引擎发送到网络上，有了第四次数据复制。

使用内核空间的缓冲区做中介(而不是直接将数据传输到用户空间)或许看上去是低效的，然而内核缓冲区做中介的引入就是为了改善进程的性能。从当应用程序读取文件数据这方面来说，如果读取的数据小于这个中介缓冲区的容量，那么中介缓冲区就可以提前缓存一大部分数据以供程序下次读取使用，从而提高性能。从应用程序写数据来说，这个中介缓冲区可以用来实

现异步功能（当数据缓冲区数据满了后再写出去，较少了系统调用的次数）。

不幸的是，这种方式也有它自己的瓶颈。当应用程序读取的数据比这个中介缓冲区的容量大很多的时候，数据就会在磁盘、内核空间、用户空间之间复制多次后才最终被传给应用程序。

零拷贝技术就是通过消除这种多余的数据拷贝来改善性能的。

使用zero copy的数据传输方式：

如果你再看一下传统的方式，你会发现实际上第二次和第三次数据拷贝是没有必要的。应用程序除了缓存一下数据然后传回到socket的缓冲区中啥也没干。我们可以通过直接从内核缓冲区把数据传输到socket关联的缓冲区来代替传统的方式。transferTo()方法可以帮你实现。下面是这个方法的定义：

```
1  public void transferTo(long position,long count,WritableByteChannel
   target);
```

transferTo()方法将数据从一个channel传输到另一个可写的channel上，其内部实现依赖于操作系统对zero copy技术的支持。在unix操作系统和各种linux的发型版本中，这种功能最终是通过sendfile()系统调用实现。下边就是这个方法的定义：

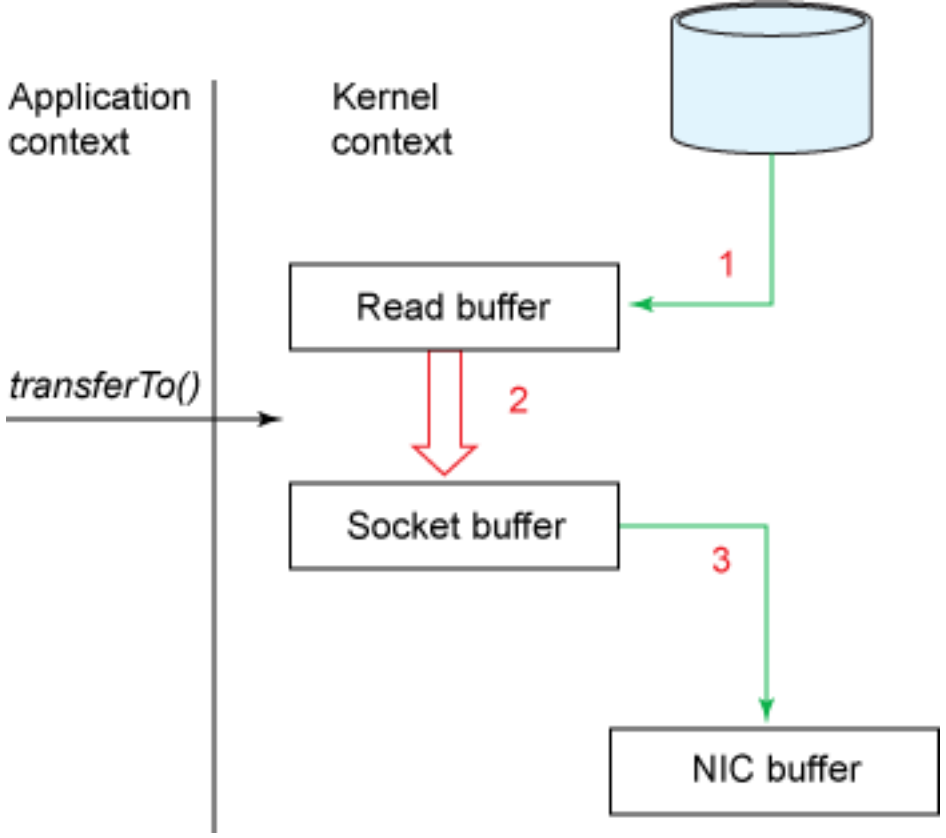
```
1  #include <sys/socket.h>

2  ssize_t sendfile(int out_fd, int in_fd, off_t *offset, size_t count);
```

可以通过调用transferTo()方法来替代上边的File.read()、Socket.send()

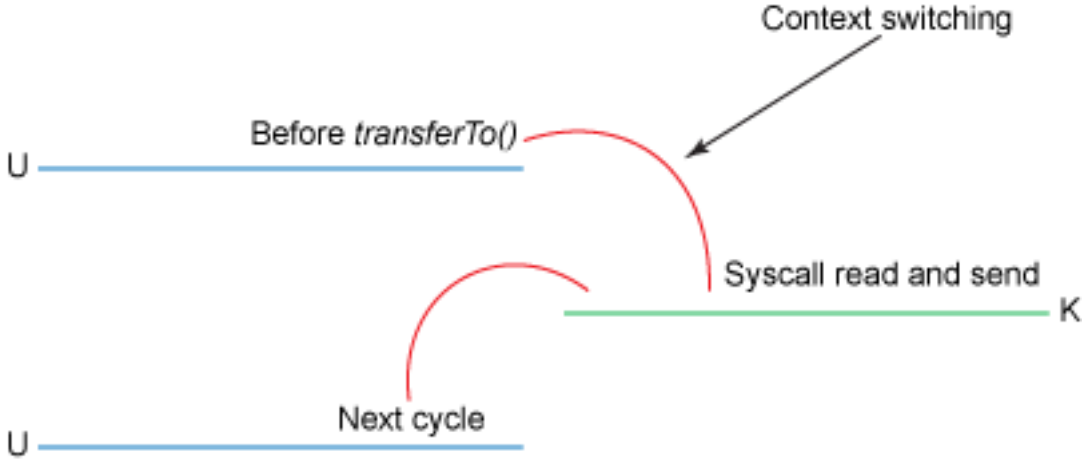
```
1  transferTo(position, count, writableChannel);
```

图三　展示了通过transferTo实现数据传输的路径：



图三，通过transferTo()实现数据拷贝

图四　展示了内核态、用户态的切换情况：



图四，tranferTo()下上下文的切换

使用transferTo()方式所经历的步骤：

1、transferTo调用会引起DMA将文件内容复制到读缓冲区(内核空间的缓冲区)，然后数据从这个缓冲区复制到另一个与socket输出相关的内核缓冲区中。

2、第三次数据复制就是DMA把socket关联的缓冲区中的数据复制到协议引擎上发送到网络上。

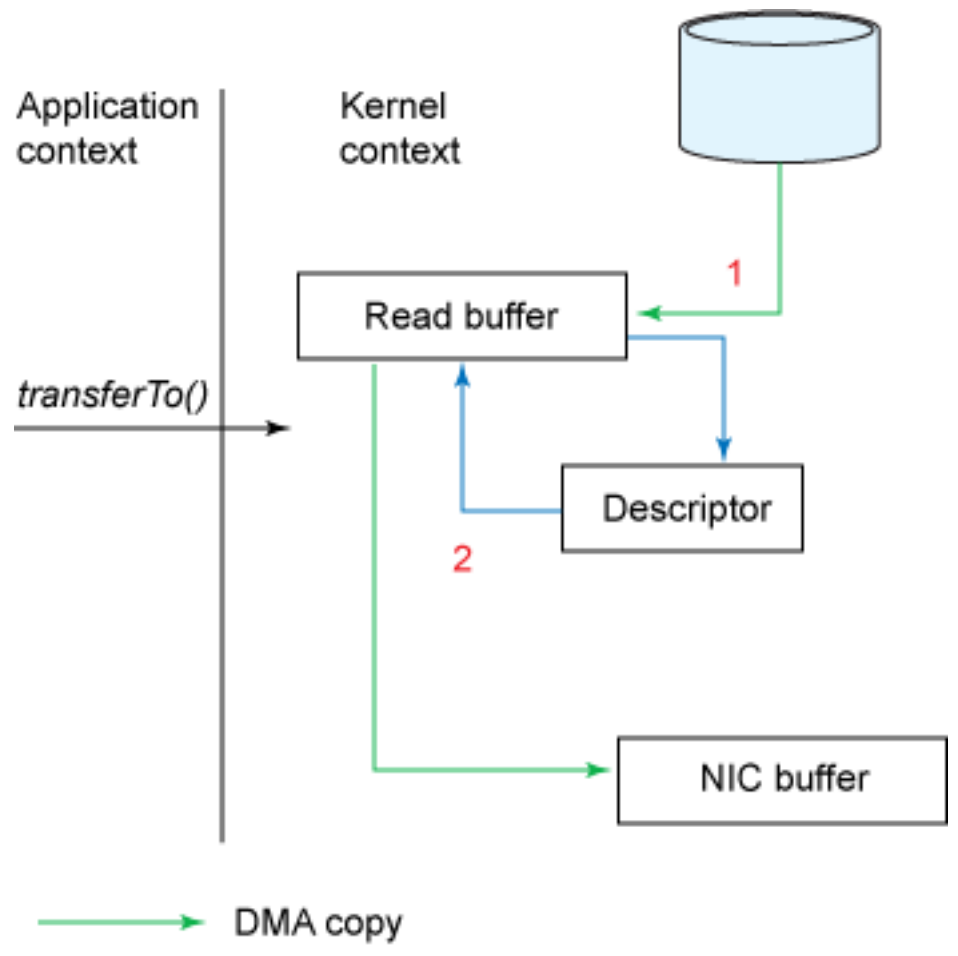
这次改善，我们是通过将内核、用户态切换的次数从四次减少到两次，

将数据的复制次数从四次减少到三次(只有一次用到cpu资源)。但这并没有达到我们零复制的目标。如果底层网络适配器支持收集操作的话，我们可以进一步减少内核对数据的复制次数。在内核为2.4或者以上版本的linux系统上，socket缓冲区描述符将被用来满足这个需求。这个方式不仅减少了内核用户态间的切换，而且也省去了那次需要cpu参与的复制过程。从用户角度来看依旧是调用transferTo()方法，但是其本质发生了变化：‘

1、调用transferTo方法后数据被DMA从文件复制到了内核的一个缓冲区中。

2、数据不再被复制到socket关联的缓冲区中了，仅仅是将一个描述符（包含了数据的位置和长度等信息）追加到socket关联的缓冲区中。DMA直接将内核中的缓冲区中的数据传输给协议引擎，消除了仅剩的一次需要cpu周期的数据复制。

图五展示了收集操作下transferTo的工作流程



图五

构建一个文件传输服务器

接下来让我们用一个在客户端和服务器段传输文件的实例来实践一下我们的zero copy技术(代码到下面下载)。TraditionalClient.java和

TraditionalServer.java 是基于传统的实现方式。TraditionalServer.java 是一个服务器程序，绑定在一个端口等待客户端的连接，然后从客户端的连接中一次读取4k字节的数据。TraditionalClient.java连接到服务器，然后从文件中读取数据通过网络传送给服务器。

同样的，TransferToServer.java 和TransferToClient.java执行相同的功能，但是使用了transfeTo()方法将文件从服务器发送到客户端。

性能比较

我们在内核为2.6版本的linux上运行了这个例子程序，并测试了在不同文件大小的情况下，传统的方式和transferTo方式所消耗的毫秒数，下边是测试结果

File size	Normal file transfer (ms)	transferTo (ms)
7MB	156	45
21MB	337	128
63MB	843	387
98MB	1320	617
200MB	2124	1150
350MB	3631	1762
700MB	13498	4422
1GB	18399	8537

我们可以看到，使用transferTo方式比传统方式少大约65%的时间消耗。对于那些需要大量读取io数据传输到另一个channel的服务器程序来说，使用zero copy方式性能上的提高是相当显著地。比如web服务器。

摘要

我们展示了transferTo比传统方式上的性能优势，在从一个channel读取相同数据发送到另一个channel的操作上。如果有一个需要在channel间大量复制数据的应用程序，使用zero copy将会有一个更大的性能提高。

下载

描述	名字	大小
文章中的简单例子程序	j-zerocopy.zip	3kb