一、典型IO调用的问题

一个典型的web服务器传送静态文件（如CSS，JS，图片等）的过程如下：

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

首先调用read将文件从磁盘读取到tmp_buf，然后调用write将tmp_buf写入到socket，在这过程中会出现四次数据 copy，过程如图1所示

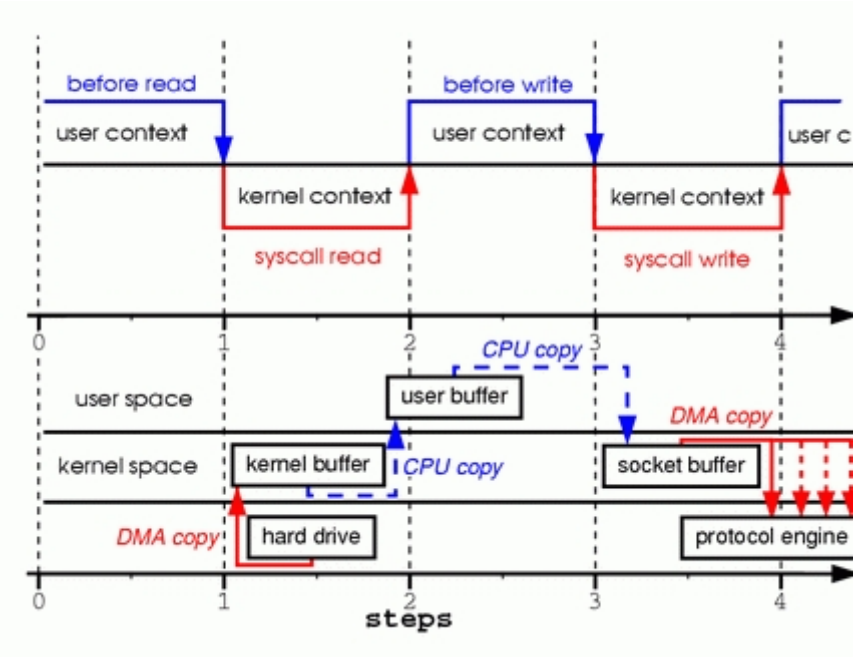

图1

1。当调用read系统调用时，通过DMA（Direct Memory Access）将数据copy到内核模式

2。然后由CPU控制将内核模式数据copy到用户模式下的 buffer中

3。read调用完成后，write调用首先将用户模式下 buffer中的数据copy到内核模式下的socket buffer中

4。最后通过DMA copy将内核模式下的socket buffer中的数据copy到网卡设备中传送。

从上面的过程可以看出，数据白白从内核模式到用户模式走了一 圈，浪费了两次copy，而这两次copy都是CPU copy，即占用CPU资源。

二、Zero-Copy&Sendfile()

Linux 2.1版本内核引入了sendfile函数，用于将文件通过socket传送。

sendfile(socket, file, len);

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的详细过程图2所示：

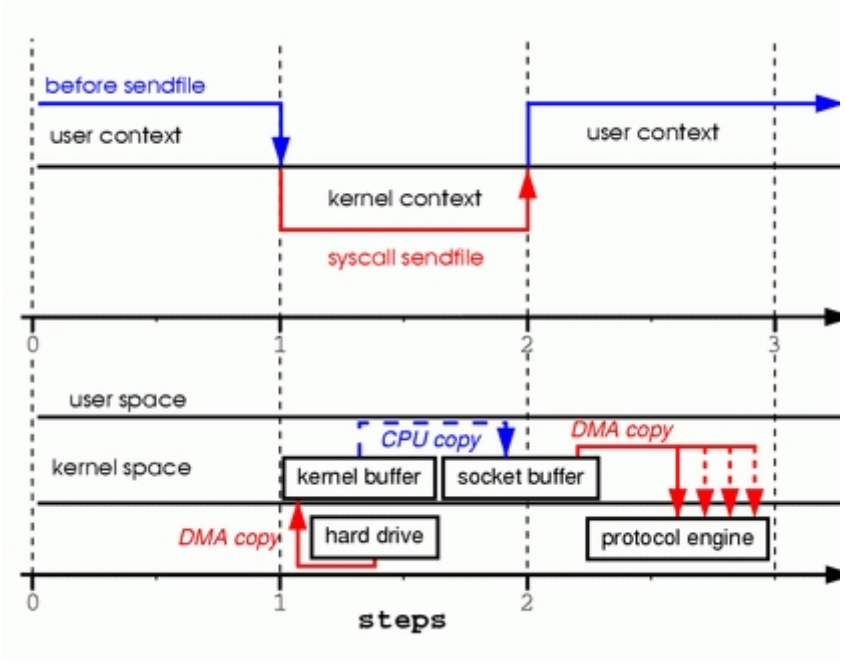

图2

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 copy将数据从磁盘读取到kernel buffer中

2。然后通过CPU copy将数据从kernel buffer copy到sokcet buffer中

3。最终通过DMA copy将socket buffer中数据copy到网卡buffer中发送

sendfile与read/write方式相比，少了 一次模式切换一次CPU 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是没必要的。

为此，Linux2.4内核对sendfile做了改进，如图3所示

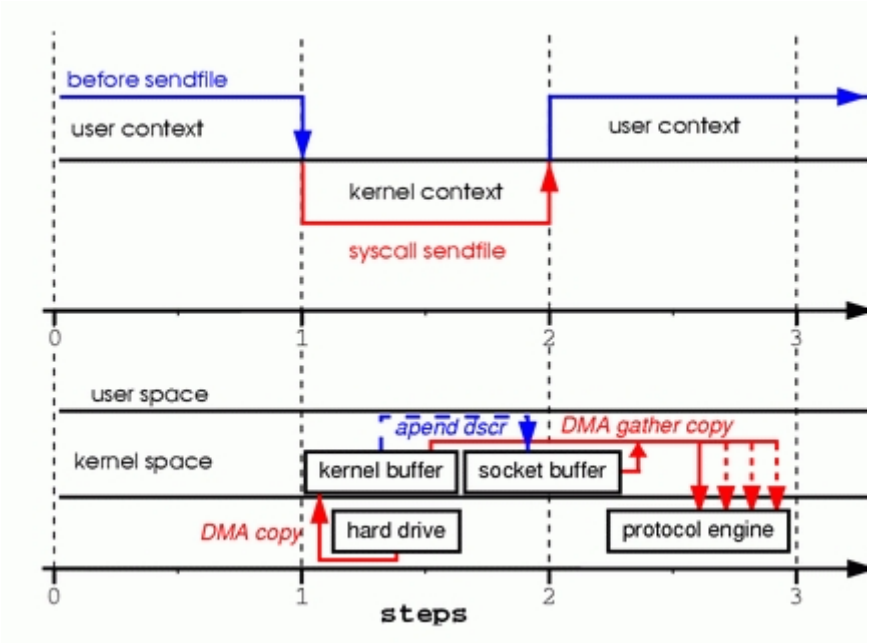

图3

改进后的处理过程如下：

1。DMA copy将磁盘数据copy到kernel buffer中

2。向socket buffer中追加当前要发送的数据在kernel buffer中的位置和偏移量

3。DMA gather copy根据socket buffer中的位置和偏移量直接将kernel buffer中的数据copy到网卡上。

经过上述过程，数据只经过了2次copy就从磁盘传送出去了。

（可能有人要纠结“不是说Zero－Copy么？怎么还有两次copy啊”，事实上这个Zero copy是针对内核来讲的，数据在内核模式下是Zero－copy的。话说回来，文件本身在瓷盘上要真是完全Zero－copy就能传送，那才见鬼了 呢）。

当前许多高性能http server都引入了sendfile机制，如nginx，lighttpd等。

三、Java NIO中的transferTo()

Java NIO中

FileChannel.transferTo(long position, long count, WriteableByteChannel target)

方法将当前通道中的数据传送到目标通道target中，在支持Zero-Copy的linux系统中，transferTo()的实现依赖于 sendfile()调用。