developerWorks 中国 正在向 IBM Developer 过渡。 我们将为您呈现一个全新的界面和更新的主题领域，并一如既往地提供您希望获得的精彩内容。

Linux 中的零拷贝技术，第 2 部分

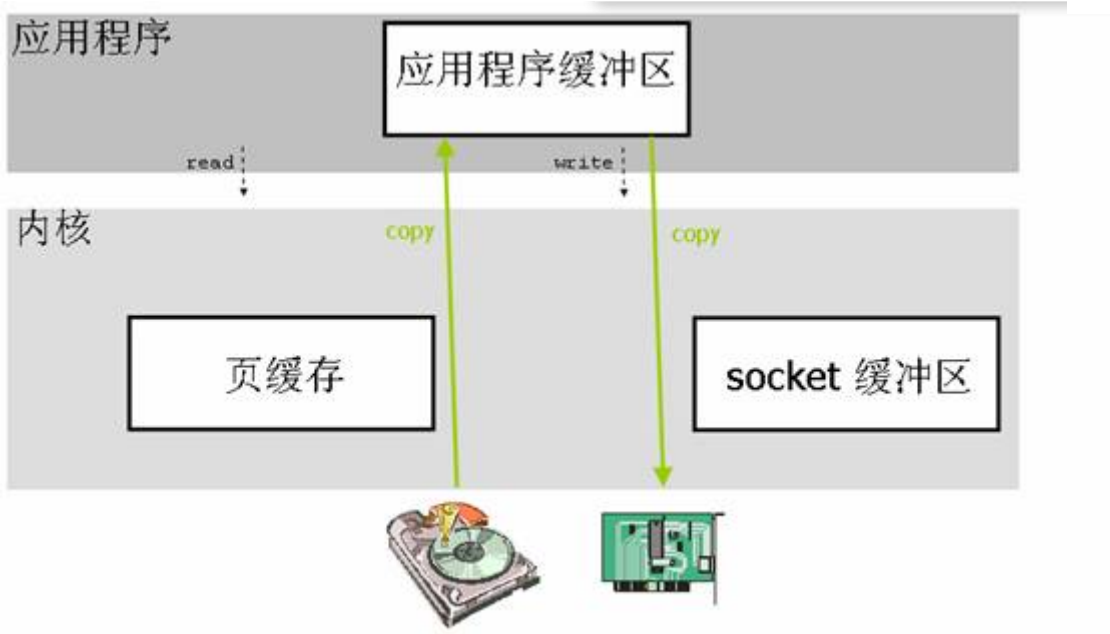
技术实现

黄 晓晨 和 冯 瑞
2011 年 1 月 27 日发布

Linux 中的直接 I/O

如果应用程序可以直接访问网络接口存储，那么在应用程序访问数据之前存储总线就不需要被是最小的。应用程序或者运行在用户模式下的库函数可以直接访问硬件设备的存储，操作系统工作之外，不参与数据传输过程中的其它任何事情。直接 I/O 使得数据可以直接在应用程序和要操作系统内核页缓存的支持。关于直接 I/O 技术的具体实现细节可以参看 developerWorks _ 机制的介绍”，本文不做过多描述。

图 1. 使用直接 I/O 的数据传输



针对数据传输不需要经过应用程序地址空间的零拷贝技术

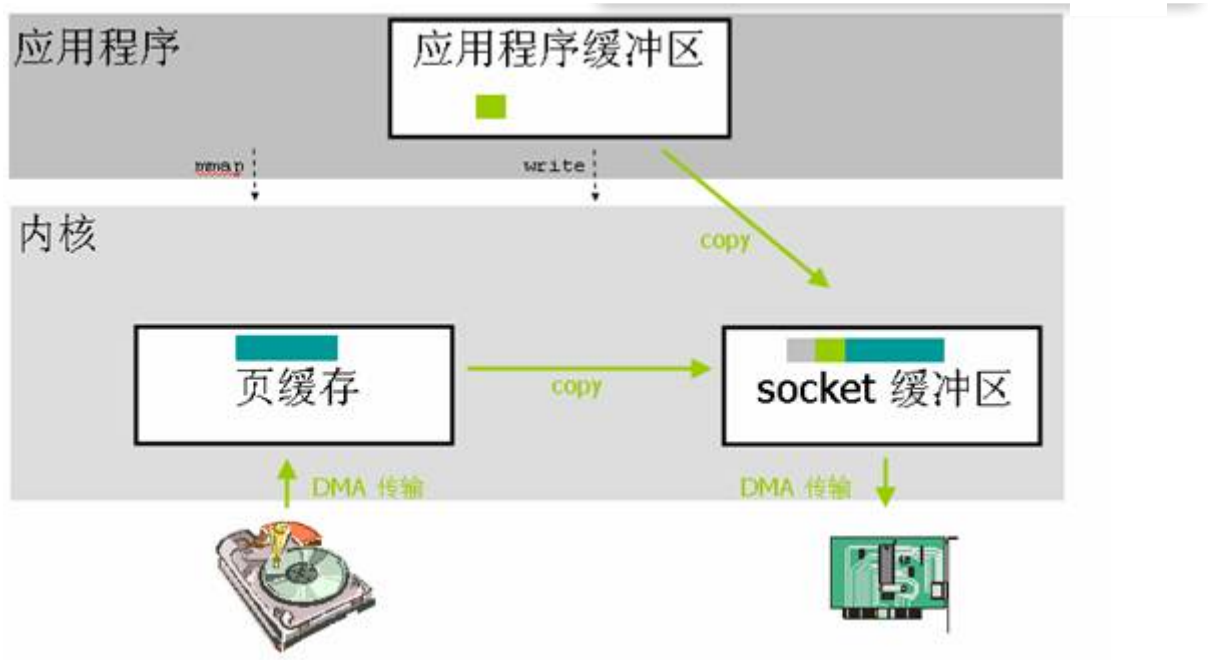
利用 mmap()

在 Linux 中，减少拷贝次数的一种方法是调用 mmap() 来代替调用 read，比如：

```
1 tmp_buf = mmap(file, len);
2 write(socket, tmp_buf, len);
```

首先，应用程序调用了 mmap() 之后，数据会先通过 DMA 拷贝到操作系统内核的缓冲区中去。这个缓冲区，这样，操作系统内核和应用程序存储空间就不需要再进行任何的数据拷贝操作。系统内核将数据从原来的内核缓冲区中拷贝到与 socket 相关的内核缓冲区中。接下来，数据从擎中去，这是第三次数据拷贝操作。

图 2. 利用 mmap() 代替 read()



通过使用 mmap() 来代替 read(), 已经可以减半操作系统需要进行数据拷贝的次数。当大量数据
个比较好的效率。但是，这种改进也是需要代价的，使用 mma()p 其实是存在潜在的问题的。
用 write() 系统调用，如果此时其他的进程截断了这个文件，那么 write() 系统调用将会被总线
正在执行的是一个错误的存储访问。这个信号将会导致进程被杀死，解决这个问题可以通过以

1. 为 SIGBUS 安装一个新的信号处理器，这样，write() 系统调用在它被中断之前就返回已经
   成 success。但是这种方法也有其缺点，它不能反映出产生这个问题的根源所在，因为 BI
   一些很严重的错误。
2. 第二种方法是通过文件租借锁来解决这个问题的，这种方法相对来说更好一些。我们可以
   锁，当另外一个进程尝试对用户正在进行传输的文件进行截断的时候，内核会发送给用户
   信号，这个信号会告诉用户内核破坏了用户加在那个文件上的写或者读租借锁，那么 write
   会被 SIGBUS 信号杀死，返回值则是中断前写的字节数，errno 也会被设置为 success。文
   射之前设置。

使用 mmap 是 POSIX 兼容的，但是使用 mmap 并不一定能获得理想的数据传输性能。数据传
操作，而且映射操作也是一个开销很大的虚拟存储操作，这种操作需要通过更改页表以及冲刷
持存储的一致性。但是，因为映射通常适用于较大范围，所以对于相同长度的数据来说，映射
带来的开销。

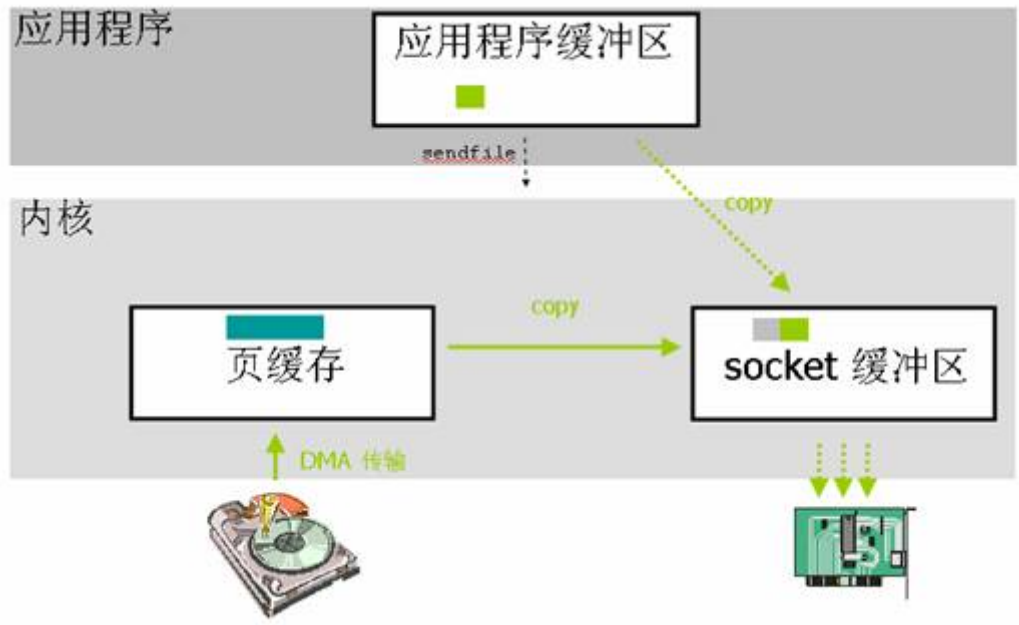
sendfile()

为了简化用户接口，同时还要继续保留 mmap()/write() 技术的优点：减少 CPU 的拷贝次数，
sendfile() 这个系统调用。

sendfile() 不仅减少了数据拷贝操作，它也减少了上下文切换。首先：sendfile() 系统调用利用
操作系统内核缓冲区中，然后数据被拷贝到与 socket 相关的内核缓冲区
接下来，DMA

拷贝到协议引擎中去。如果在用户调用 sendfile () 系统调用进行数据传输的过程中有其他进程
统调用会简单地返回给用户应用程序中断前所传输的字节数，errno 会被设置为 success。如果
文件加上了租借锁，那么 sendfile() 的操作和返回状态将会和 mmap()/write () 一样。

图 3. 利用 sendfile () 进行数据传输



sendfile() 系统调用不需要将数据拷贝或者映射到应用程序地址空间中去，所以 sendfile() 只是
所访问数据进行处理的情况。相对于 mmap() 方法来说，因为 sendfile 传输的数据没有越过用
线，所以 sendfile () 也极大地减少了存储管理的开销。但是，sendfile () 也有很多局限性，如

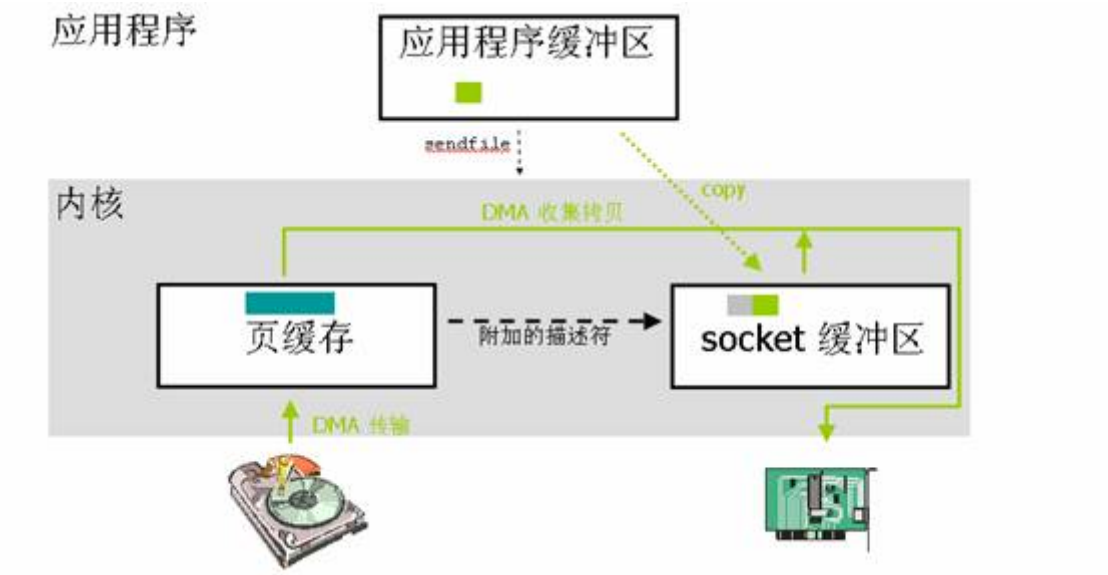
- sendfile() 局限于基于文件服务的网络应用程序，比如 web 服务器。据说，在 Linux 内核中
台上使用 sendfile() 的 Apache 程序。
- 由于网络传输具有异步性，很难在 sendfile () 系统调用的接收端进行配对的实现方式，所以
这种技术。
- 基于性能的考虑来说，sendfile () 仍然需要有一次从文件到 socket 缓冲区的 CPU 拷贝操作
的数据所污染。

带有 DMA 收集拷贝功能的 sendfile()

上小节介绍的 sendfile() 技术在进行数据传输仍然还需要一次多余的数据拷贝操作，通过引入
数据拷贝操作也可以避免。为了避免操作系统内核造成的数据副本，需要用到一个支持收集操
输的数据可以分散在存储的不同位置上，而不需要在连续存储中存放。这样一来，从文件中读
socket 缓冲区中去，而只是需要将缓冲区描述符传到网络协议栈中去，之后其在缓冲区中建立
DMA 收集拷贝功能将所有的数据结合成一个网络数据包。网卡的 DMA 引擎会在一次操作中从
2.4 版本中的 socket 缓冲区就可以满足这种条件，这也就是用于 Linux 中的众所周知的零拷贝
次上下文切换所带来开销，同时也减少了处理器造成的数据副本的个数。对于用户应用程序来

sendfile() 系统调用利用 DMA 引擎将文件内容拷贝到内核缓冲区去；然后，将带有文件位置和 socket 缓冲区中去，此过程不需要将数据从操作系统内核缓冲区拷贝到 socket 缓冲区中，DM 拷贝到协议引擎中去，这样就避免了最后一次数据拷贝。

图 4. 带有 DMA 收集拷贝功能的 sendfile



通过这种方法，CPU 在数据传输的过程中不但避免了数据拷贝操作，理论上，CPU 也永远不会 CPU 的性能来说起到了积极的作用：首先，高速缓冲存储器没有受到污染；其次，高速缓冲存 冲存储器在 DMA 进行数据传输前或者传输后不需要被刷新。然而实际上，后者实现起来非常困 一部分，这也就是说一般的读操作可以访问它，而且该访问也可以是通过传统方式进行的。只 么高速缓冲存储器的一致性就需要通过 DMA 传输之前冲刷新高速缓冲存储器来维护。而且，这 要硬件以及设备驱动程序支持的。

splice()

splice() 是 Linux 中与 mmap() 和 sendfile() 类似的一种方法。它也可以用于用户应用程 间的数据传输。splice() 适用于可以确定数据传输路径的用户应用程序，它不需要利用用户地址 输操作。那么，当数据只是从一个地方传送到另一个地方，过程中所传输的数据不需要经过用 就成为了一种比较好的选择。splice() 可以在操作系统地址空间中整块地移动数据，从而减少大 splice() 进行数据传输可以通过异步的方式来进行，用户应用程序可以先从系统调用返回，而操 过程继续进行下去。splice() 可以被看成是类似于基于流的管道的实现，管道可以使得两个文件 则可以控制两个设备（或者协议栈）在操作系统内核中的相互连接。

splice() 系统调用和 sendfile() 非常类似，用户应用程序必须拥有两个已经打开的文件描述符， 表示输出设备。与 sendfile() 不同的是，splice() 允许任意两个文件之间互相连接，而并不只是 于从一个文件描述符发送数据到 socket 这种特例来说，一直都是使用 sendfile() 这个系统调用 制，它并不仅限于 sendfile() 的功能。也就是说，sendfile() 只是 splice() 的一个子集，在 Linux 的实现已经没有了，但是这个 API 以及相应的功能还存在，只不过 API 以及相应的功能是利用

在数据传输的过程中，splice() 机制交替地发送相关的文件描述符的读写操作，并且可以将读缓
了一种简单的流控制，通过预先定义的水印（watermark）来阻塞写请求。有实验表明，利用
另一个磁盘会增加 30% 到 70% 的吞吐量，数据传输的过程中， CPU 的负载也会减少一半。

Linux 2.6.17 内核引入了 splice() 系统调用，但是，这个概念在此之前] 其实已经存在了很长一
提出了这个概念，它被看成是一种改进服务器端系统的 I/O 性能的一种技术，尽管在之后的若
统调用从来没有在主流的 Linux 操作系统内核中实现过，一直到 Linux 2.6.17 版本的出现。sp
其中两个是文件描述符，一个表示文件长度，还有一个用于控制如何进行数据拷贝。splice 系
异步方式来实现。在使用异步方式的时候，用户应用程序会通过信号 SIGIO 来获知数据传输已
如下所示：

```
long splice(int fdin, int fdout, size_t len, unsigned int flags);
```

调用 splice() 系统调用会导致操作系统内核从数据源 fdin 移动最多 len 个字节的数据到 fdout 中
过操作系统内核空间，需要最少的拷贝次数。使用 splice() 系统调用需要这两个文件描述符中的
备的。不难看出，这种设计具有局限性，Linux 的后续版本针对这一问题将会有所改进。参数 f
法，当前的 flags 有如下这些取值：

- SPLICE_F_NONBLOCK：splice 操作不会被阻塞。然而，如果文件描述符没有被设置为不可
  splice 有可能仍然被阻塞。
- SPLICE_F_MORE：告知操作系统内核下一个 splice 系统调用将会有更多的数据传来。
- SPLICE_F_MOVE：如果输出是文件，这个值则会使得操作系统内核尝试从输入管道缓冲区
  这个数据传输过程没有任何数据拷贝操作发生。

Splice() 系统调用利用了 Linux 提出的管道缓冲区（pipe buffer）机制，这就是为什么这个系统
少有一个必须要指代管道设备的原因。为了支持 splice 这种机制，Linux 在用于设备和文件系统
下边这两个定义：

```
ssize_t (*splice_write)(struct inode *pipe, strucuct file *out,
                        size_t len, unsigned int flags);
ssize_t (*splice_read)(struct inode *in, strucuct file *pipe,
                        size_t len, unsigned int flags);
```

这两个新的操作可以根据 flags 的设定在 pipe 和 in 或者 out 之间移动 len 个字节。Linux 文件系
可以使用的操作，而且还实现了一个 generic_splice_sendpage() 函数用于和 socket 之间的接口

对应用程序地址空间和内核之间的数据传输进行优化的零拷贝技

前面提到的几种零拷贝技术都是通过尽量避免用户应用程序和操作系统内核缓冲区之间的数据
贝技术的应用程序通常都要局限于某些特殊的情况：要么不能在操作系统内核中处理数据，要
据。而这一小节提出的零拷贝技术保留了传统在用户应用程序地址空间和操作系统内核地址空
输上进行优化。我们知道，数据在系统软件和硬件之间的传递可以通过 DMA 传输来提高效率，
统之间进行数据传输这种情况来说，并没有类似的工具可以使用。本节介绍的技术就是针对这

利用写时复制

在某些情况下，Linux 操作系统内核中的页缓存可能会被多个应用程序所共享，操作系统有可能
区中的页面映射到操作系统内核地址空间中去。如果某个应用程序想要对这共享的数据调用
坏内核缓冲区中的共享数据，传统的 write() 系统调用并没有提供任何显示的加锁操作，Linux
来保护数据。

什么是写时复制

写时复制是计算机编程中的一种优化策略，它的基本思想是这样的：如果有多个应用程序需要
这些应用程序分配指向这块数据的指针，在每一个应用程序看来，它们都拥有这块数据的一份
要对自己的这份数据拷贝进行修改的时候，就需要将数据真正地拷贝到该应用程序的地址空间
了一份真正的私有数据拷贝，这样做是为了避免该应用程序对这块数据做的更改被其他应用程
说是透明的，如果应用程序永远不会对所访问的这块数据进行任何更改，那么就永远不需要将
间中去。这也是写时复制的最主要的优点。

写时复制的实现需要 MMU 的支持，MMU 需要知晓进程地址空间中哪些特殊的页面是只读的，
候，MMU 就会发出一个异常给操作系统内核，操作系统内核就会分配新的物理存储空间，即将
理存储位置相对应。

写时复制的最大好处就是可以节约内存。不过对于操作系统内核来说，写时复制增加了其处理

数据传输的实现及其局限性

数据发送端

对于数据传输的发送端来说，实现相对来说是比较简单的，对与应用程序缓冲区相关的物理页
操作系统内核的地址空间，并标识为“ write only ”。当系统调用返回的时候，用户应用程序和
的数据。在操作系统已经传送完所有的数据之后，应用程序就可以对这些数据进行写操作。如
前对数据进行写操作，那么就会产生异常，这个时候操作系统就会将数据拷贝到应用程序自己
端的映射。数据传输完成之后，对加锁的页面进行解锁操作，并重置 COW 标识。

数据接收端

对于数据接收端来说，该技术的实现则需要处理复杂得多的情况。如果 read() 系统调用是在数
序是被阻塞的，那么 read() 系统调用就会告知操作系统接收到的数据包中的数据应该存放到什
有必要进行页面重映射，网络接口卡可以提供足够的支持让数据直接存入用户应用程序的缓冲
在 read() 系统调用发出之前，操作系统不知道该把数据写到哪里，因为它不知道用户应用程序
必须要先把数据存放到自己的缓冲区中去。

局限性

写时复制技术有可能会导致操作系统的处理开销很大．所有相关的缓冲区都必须要进行页对齐
要是整数个的。对于发送端来说，这不会造成什么问题。但是对于接收端来说，它需要有能力
包的尺寸大小要合适，大小需要恰到好处能够覆盖一整页的数据，这就限制了那些 MTU 大小大
和 ATM。其次，为了在没有任何中断的情况下将页面重映射到数据包的流，数据包中的数据部
接收数据的情况来说，为了将数据高效地移动到用户地址空间中去，可以使用这样一种方法：
据包可以被分割成包头和数据两部分，数据被存放在一个单独的缓冲区内，虚拟存储系统然后
冲区去。使用这种方法需要满足两个先决条件，也就是上面提到过的：一是应用程序缓冲区必
是连续的；二是传来的数据有一页大小的时候才可以对数据包进行分割。事实上，这两个先决
缓冲区不是页对齐的，或者数据包的大小超过一个页，那么数据就需要被拷贝。对于数据发送
对于应用程序来说是写保护的，应用程序仍然需要避免使用这些忙缓冲区，这是因为写时拷贝
没有端到端这一级别的通知，那么应用程序很难会知道某缓冲区是否已经被释放还是仍然在被

这种零拷贝技术比较适用于那种写时复制事件发生比较少的情况，因为写时复制事件所产生的
生的开销。实际情况中，大多数应用程序通常都会多次重复使用相同的缓冲区，所以，一次使
址空间解除页面的映射，这样会提高效率。考虑到同样的页面可能会被再次访问，所以保留页
是，这种映射保留不会减少由于页表往返移动和 TLB 冲刷所带来的开销，这是因为每次页面由
时候，页面的只读标志都要被更改。

缓冲区共享

还有另外一种利用预先映射机制的共享缓冲区的方法也可以在应用程序地址空间和操作系统内
共享这种思想的架构最先在 Solaris 上实现，该架构使用了“ fbufs ”这个概念。这种方法需要修
系统内核地址空间之间的数据传递需要严格按照 fbufs 体系结构来实现，操作系统内核之间的
来完成的。每一个应用程序都有一个缓冲区池，这个缓冲区池被同时映射到用户地址空间和内
才创建它们。通过完成一次虚拟存储操作来创建缓冲区，fbufs 可以有效地减少由存储一致性维
技术在 Linux 中还停留在实验阶段。

为什么要扩展 Linux I/O API

传统的 Linux 输入输出接口，比如读和写系统调用，都是基于拷贝的，也就是说，数据需要在
冲区之间进行拷贝。对于读系统调用来说，用户应用程序呈现给操作系统内核一个预先分配好
据放到这个缓冲区内。对于写系统调用来说，只要系统调用返回，用户应[redacted]序就可以自由重

为了支持上面这种机制，Linux 需要能够为每一个操作都进行建立和删除虚拟存储映射。这种页置、cache 体系结构、TLB 未命中处理所带来的开销以及处理器是单处理器还是多处理器等多种求的时候虚拟存储 / TLB 操作所产生的开销，则会极大地提高 I/O 的性能。fbufs 就是这样一种避免虚拟存储操作。由数据显示，fbufs 这种结构在 DECStation™ 5000/200 这个单处理器工作映射方法好得多的性能。如果要使用 fbufs 这种体系结构，必须要扩展 Linux API，从而实现一

快速缓冲区（Fast Buffers）原理介绍

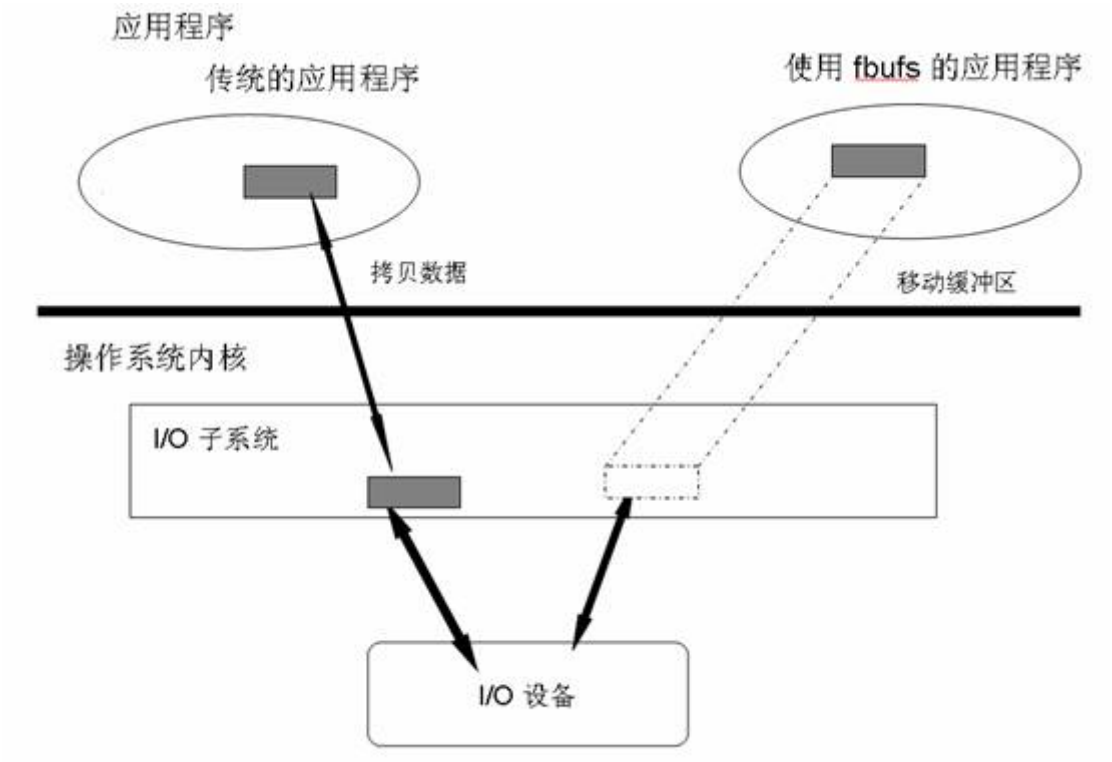
I/O 数据存放在一些被称作 fbufs 的缓冲区内，每一个这样的缓冲区都包含一个或者多个连续的是通过保护域来实现的，有如下这两种方式：

- 如果应用程序分配了 fbuf，那么应用程序就有访问该 fbuf 的权限
- 如果应用程序通过 IPC 接收到了 fbuf，那么应用程序对这个 fbuf 也有访问的权限

对于第一种情况来说，这个保护域被称作是 fbuf 的“ originator ”；对于后一种情况来说，这个”。

传统的 Linux I/O 接口支持数据在应用程序地址空间和操作系统内核之间交换，这种交换操作导如果采用 fbufs 这种方法，需要交换的是包含数据的缓冲区，这样就消除了多余的拷贝操作。应核，这样就能减少传统的 write 系统调用所产生的数据拷贝开销。同样的，应用程序通过 fbuf read 系统调用所产生的数据拷贝开销。如下图所示：

图 5. Linux I/O API

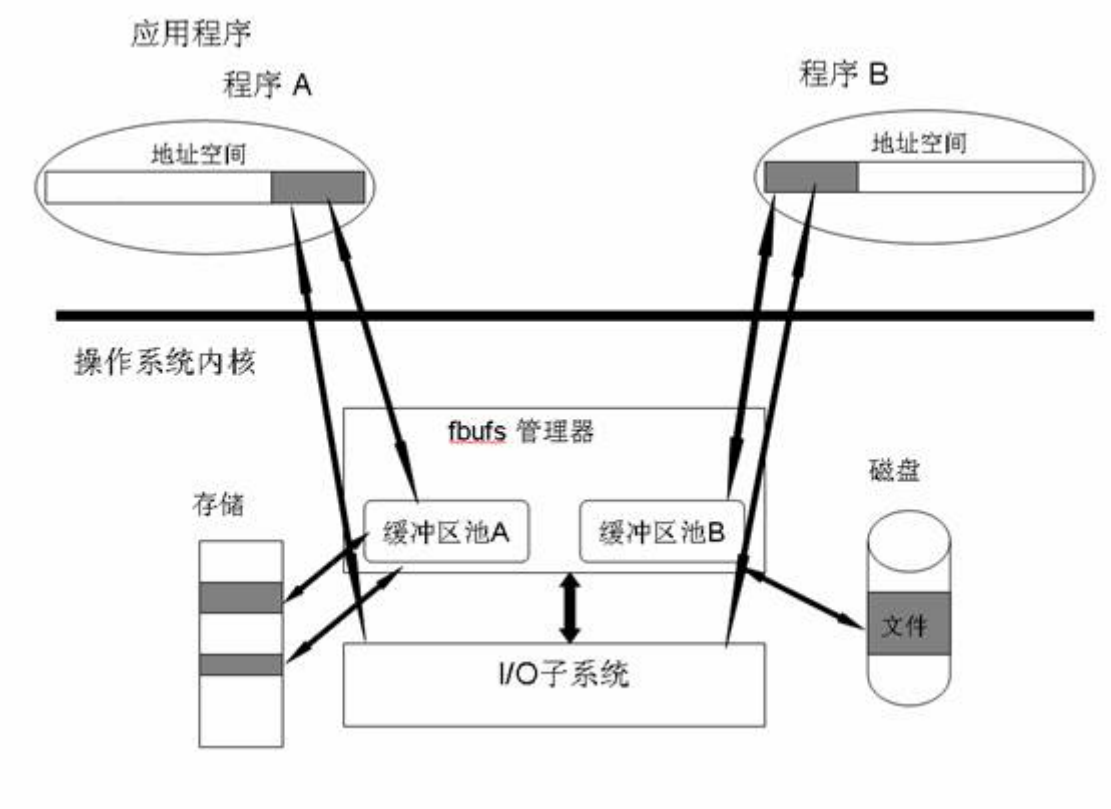


I/O 子系统或者应用程序都可以通过 fbufs 管理器来分配 fbufs。一旦分配了 fbufs，这些 fbufs 或者从 I/O 子系统传递到程序。使用完后，这些 fbufs 会被释放回 fbufs 缓冲区池。

fbufs 在实现上有如下这些特性，如图 9 所示：

- fbuf 需要从 fbufs 缓冲区池里分配。每一个 fbuf 都存在一个所属对象，要么是应用程序，要 应用程序和操作系统之间进行传递，fbuf 使用完之后需要被释放回特定的 fbufs 缓冲区池，带关于 fbufs 缓冲区池的相关信息。
- 每一个 fbufs 缓冲区池都会和一个应用程序相关联，一个应用程序最多只能与一个 fbufs 缓 访问它自己的缓冲区池。
- fbufs 不需要虚拟地址重映射，这是因为对于每个应用程序来说，它们可以重新使用相同的 的信息就可以被缓存起来，虚拟存储子系统方面的开销就可以消除。
- I/O 子系统（设备驱动程序，文件系统等）可以分配 fbufs，并将到达的数据直接放到这些 贝操作就可以避免。

图 6. fbufs 体系结构



前面提到，这种方法需要修改 API，如果要使用 fbufs 体系结构，应用程序和 Linux 操作系统内 如果应用程序要发送数据，那么它就要从缓冲区池里获取一个 fbuf，将数据填充进去，然后通 收到的 fbufs 可以被应用程序保留一段时间，之后，应用程序可以使用它继续发送其他的数据， 些情况下，需要对数据包内的数据进行重新组装，那么通过 fbuf 接收到数据的应用程序就需要 再者，应用程序不能对当前正在被内核处理的数据进行修改，基于这一点，fbufs 体系结构引入 对于应用程序来说，如果 fbufs 已经被发送给操作系统内核，那么应用程序就不会再处理这些

fbufs 存在的一些问题

管理共享缓冲区池需要应用程序、网络软件、以及设备驱动程序之间的紧密合作。对于数据接
到达的数据包利用 DMA 传输到由接收端分配的正确的存储缓冲区池中去。而且，应用程序稍微
储中的数据的内容，从而导致数据被破坏，但是这种问题在应用程序端是很难调试的。同时，
的存储对象关联使用，但是应用程序、网络软件以及设备驱动程序之间的紧密合作是需要其他
冲区这种技术来说，虽然这种技术看起来前景光明，但是这种技术不但需要对 API 进行更改，
并且这种技术本身也存在一些未解决的问题，这就使得这种技术目前还只是出于试验阶段。在
很大的改进，不过这种新的架构的整体安装目前看起来还是不可行的。这种预先分配共享缓冲
将数据拷贝到另外一个缓冲区中去。

总结

本系列文章介绍了 Linux 中的零拷贝技术，本文是其中的第二部分。本文对第一部分文章中提
零拷贝技术进行了更详细的介绍，主要描述了它们各自的优点，缺点以及适用场景。对于网络
受到了很多体系结构方面因素的阻碍，包括虚拟存储体系结构以及网络协议体系结构等。所以
殊的情况中才可以应用，比如文件服务或者使用某种特殊的协议进行高带宽的通信等。但是，
可行性就高得多了，这很可能是因为磁盘操作具有同步的特点，以及数据传输单元是按照页的

针对 Linux 操作系统平台提出并实现了很多种零拷贝技术，但是并不是所有这些零拷贝技术都
的。比如，fbufs 体系结构，它在很多方面看起来都很吸引人，但是使用它需要更改 API 以及
上的困难，这就使得 fbufs 还只是停留在实验的阶段。动态地址重映射技术只是需要对操作系
户软件，但是当前的虚拟存储体系结构并不能很好地支持频繁的虚拟地址重映射操作。而且为
还必须对 TLB 和一级缓存进行刷新。事实上，利用地址重映射实现的零拷贝技术适用的范围是
带来的开销往往要比 CPU 拷贝所产生的开销还要大。此外，为了完全消除 CPU 访问存储，通
种硬件的支持并不是很普及，同时也是非常昂贵的。

本系列文章的目的是想帮助读者理清这些出现在 Linux 操作系统中的零拷贝技术都是从何种角
的性能问题的。关于各种零拷贝技术的具体实现细节，本系列文章没有做详细描述。同时，零
善当中的，本系列文章并没有涵盖 Linux 上出现的所有零拷贝技术。

相关主题

- 参考本系列的第一部分：Linux 中的零拷贝技术，第 1 部分 概述。
- Zero Copy I: User-Mode Perspective, Linux Journal (2003), http://www.linuxjournal.com/
  系统中存在的零拷贝技术。

- 在 http://www.ibm.com/developerworks/cn/aix/library/au-tcpsystemcalls/这篇文章中可
  绍。
- 关于 Linux 中直接 I/O 技术的设计与实现请参考 developerWorks 上的文章 ”Linux 中直接 I
- http://lwn.net/Articles/28548/上可以找到 Linux 上零拷贝技术中关于用户地址空间访问技
- http://articles.techrepublic.com.com/5100-10878_11-1044112.html?tag=content;leftC
  sendfile() 优化数据传输的介绍。
- 关于 splice() 系统调用的介绍，可以参考 http://lwn.net/Articles/178199/。
- http://en.scientificcommons.org/43480276这篇文章介绍了如何通过在操作系统内核中直
  CPU 的可用性，并详细描述了 splice 系统调用如何在用户应用程序不参与的情况下进行异
- 在 Understanding the Linux Kernel(3rd Edition) 中，有关于 Linux 内核的实现。
- http://pages.cs.wisc.edu/~cao/cs736/slides/cs736-class18.ps这篇文章提出并详细介绍了
  机制，以及基于共享存储的数据传输机制。
- http://www.cs.inf.ethz.ch/group/stricker/sada/archive/chihaia.pdf，这篇文章列举了一些
  测试系统上实现了 fbufs 技术。
- 在 developerWorks Linux 专区寻找为 Linux 开发人员（包括 Linux 新手入门）准备的更多参
  章和教程。
- 在 developerWorks 上查阅所有 Linux 技巧和 Linux 教程。
- 随时关注 developerWorks 技术活动和 网络广播。

评论

添加或订阅评论，请先登录或注册。

有新评论时提醒我