再谈Linux零拷贝技术

CloudDeveloper Linux云计算网络

2019-08-04

PS: 其实这篇文章其实是我很早的原创（当时公众号还叫aCloudDeveloper），但是之前的文章删了，但是网上却被流传开来，有些还直接标原创，有些把作者弄错，这里我重新发出来，做一个说明。另外，我前两天还转载过一篇类似的，看这里。

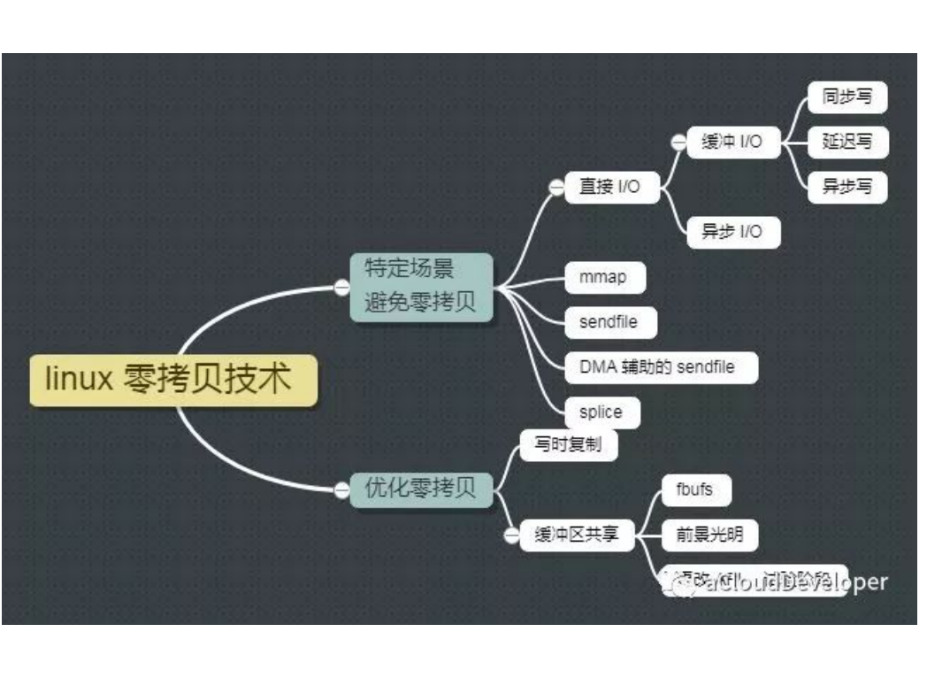
本文借鉴并总结了几种比较常见的 Linux 下的零拷贝技术，相关的引用链接见文后，大家如果觉得本文总结得太抽象，可以转到链接看详细解释。

为什么需要零拷贝

传统的 Linux 系统的标准 I/O 接口（read、write）是基于数据拷贝的，也就是数据都是 copy_to_user 或者 copy_from_user，这样做的好处是，通过中间缓存的机制，减少磁盘 I/O 的操作，但是坏处也很明显，大量数据的拷贝，用户态和内核态的频繁切换，会消耗大量的 CPU 资源，严重影响数据传输的性能，有数据表明，在Linux内核协议栈中，这个拷贝的耗时甚至占到了数据包整个处理流程的57.1%。

什么是零拷贝

零拷贝就是这个问题的一个解决方案，通过尽量避免拷贝操作来缓解 CPU 的压力。Linux 下常见的零拷贝技术可以分为两大类：一是针对特定场景，去掉不必要的拷贝；二是去优化整个拷贝的过程。由此看来，零拷贝并没有真正做到“0”拷贝，它更多是一种思想，很多的零拷贝技术都是基于这个思想去做的优化。



零拷贝的几种方法

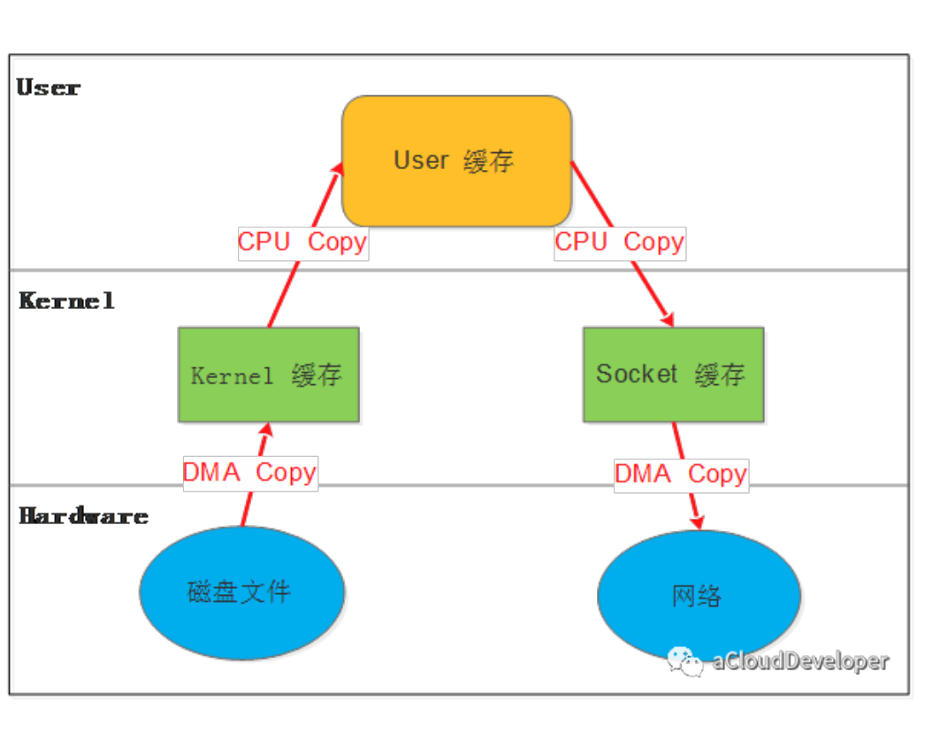
原始数据拷贝操作

在介绍之前，先看看 Linux 原始的数据拷贝操作是怎样的。如下图，假如一个应用需要从某个磁盘文件中读取内容通过网络发出去，像这样：

```
while((n = read(diskfd, buf, BUF_SIZE)) > 0)

write(sockfd, buf , n);
```

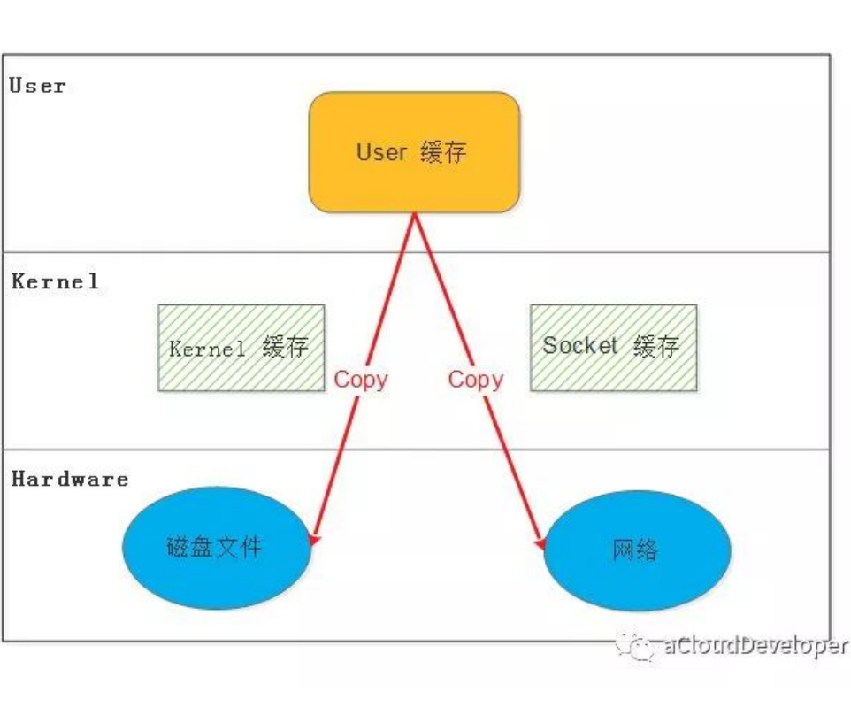
那么整个过程就需要经历：1）read 将数据从磁盘文件通过 DMA 等方式拷贝到内核开辟的缓冲区；2）数据从内核缓冲区复制到用户态缓冲区；3）write 将数据从用户态缓冲区复制到内核协议栈开辟的 socket 缓冲区；4）数据从 socket 缓冲区通过 DMA 拷贝到网卡上发出去。



可见，整个过程发生了至少四次数据拷贝，其中两次是 DMA 与硬件通讯来完成，CPU 不直接参与，去掉这两次，仍然有两次 CPU 数据拷贝操作。

方法一：用户态直接 I/O

这种方法可以使应用程序或者运行在用户态下的库函数直接访问硬件设备，数据直接跨过内核进行传输，内核在整个数据传输过程除了会进行必要的虚拟存储配置工作之外，不参与其他任何工作，这种方式能够直接绕过内核，极大提高了性能。



缺陷：

1）这种方法只能适用于那些不需要内核缓冲区处理的应用程序，这些应用程序通常在进程地址空间有自己的数据缓存机制，称为自缓存应用程序，如数据库管理系统就是一个代表。

2）这种方法直接操作磁盘 I/O，由于 CPU 和磁盘 I/O 之间的执行时间差距，会造成资源的浪费，解决这个问题需要和异步 I/O 结合使用。

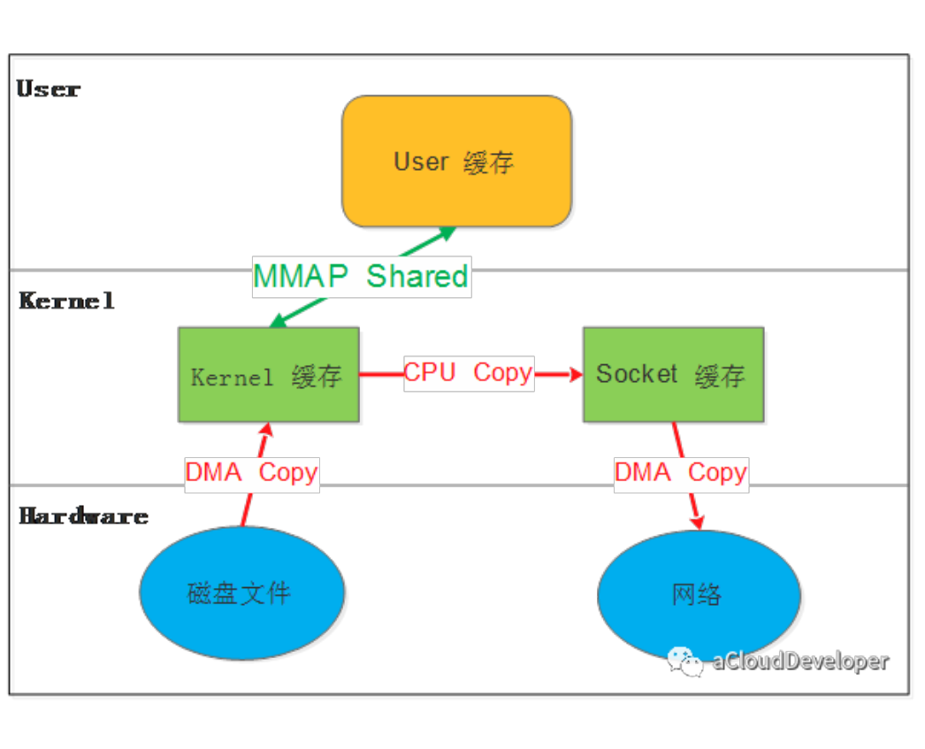
方法二：mmap

这种方法，使用 mmap 来代替 read，可以减少一次拷贝操作，如下：

```
buf = mmap(diskfd, len);

write(sockfd, buf, len);
```

应用程序调用 mmap ，磁盘文件中的数据通过 DMA 拷贝到内核缓冲区，接着操作系统会将这个缓冲区与应用程序共享，这样就不用往用户空间拷贝。应用程序调用write ，操作系统直接将数据从内核缓冲区拷贝到 socket 缓冲区，最后再通过 DMA 拷贝到网卡发出去。



缺陷：

1）mmap 隐藏着一个陷阱，当 mmap 一个文件时，如果这个文件被另一个进程所截获，那么 write 系统调用会因为访问非法地址被 SIGBUS 信号终止，SIGBUS 默认会杀死进程并产生一个 coredump，如果服务器被这样终止了，那损失就可能不小了。

解决这个问题通常使用文件的租借锁：首先为文件申请一个租借锁，当其他进程想要截断这个文件时，内核会发送一个实时的 RT_SIGNAL_LEASE 信号，告诉当前进程有进程在试图破坏文件，这样 write 在被 SIGBUS 杀死之前，会被中断，返回已经写入的字节数，并设置 errno 为 success。

通常的做法是在 mmap 之前加锁，操作完之后解锁：

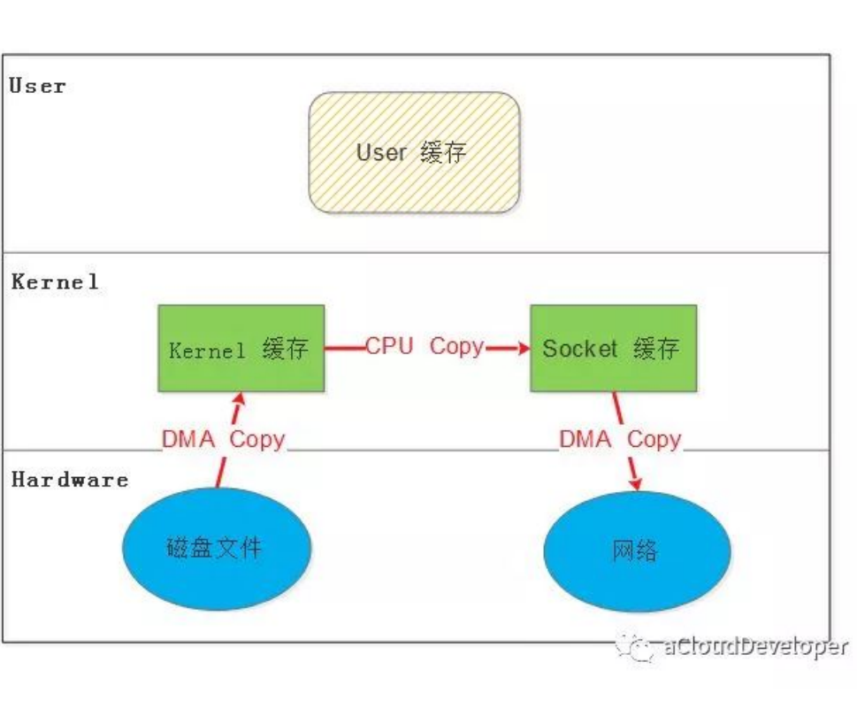
方法三：sendfile

从Linux 2.1版内核开始，Linux引入了sendfile，也能减少一次拷贝。

```
#include<sys/sendfile.h>

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

sendfile 是只发生在内核态的数据传输接口，没有用户态的参与，自然避免了用户态数据拷贝。它指定在 in_fd 和 out_fd 之间传输数据，其中，它规定 in_fd 指向的文件必须是可以 mmap 的，out_fd 必须指向一个套接字，也就是规定数据只能从文件传输到套接字，反之则不行。sendfile 不存在像 mmap 时文件被截获的情况，它自带异常处理机制。



缺陷：

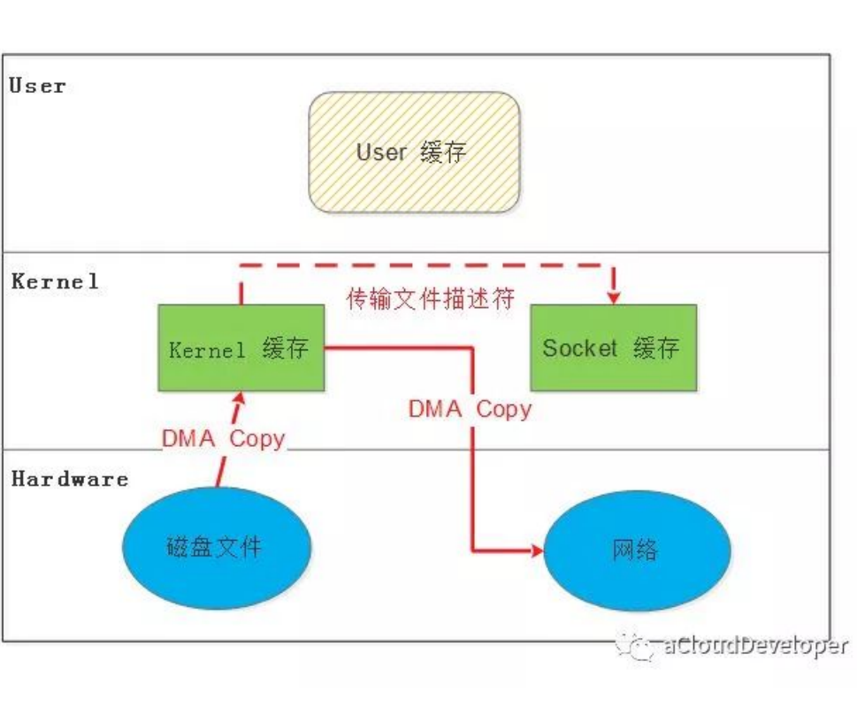
1）只能适用于那些不需要用户态处理的应用程序。

方法四：DMA 辅助的 sendfile

常规 sendfile 还有一次内核态的拷贝操作，能不能也把这次拷贝给去掉呢？

答案就是这种 DMA 辅助的 sendfile。

这种方法借助硬件的帮助，在数据从内核缓冲区到 socket 缓冲区这一步操作上，并不是拷贝数据，而是拷贝缓冲区描述符，待完成后，DMA 引擎直接将数据从内核缓冲区拷贝到协议引擎中去，避免了最后一次拷贝。



缺陷：

1）除了3.4 中的缺陷，还需要硬件以及驱动程序支持。

2）只适用于将数据从文件拷贝到套接字上。

方法五：splice

splice 去掉 sendfile 的使用范围限制，可以用于任意两个文件描述符中传输数据。

```
#define _GNU_SOURCE         /* See feature_test_macros(7) */

#include <fcntl.h>

ssize_t splice(int fd_in, loff_t *off_in, int fd_out, loff_t *off_out, size_t len, unsigned int flags);
```

但是 splice 也有局限，它使用了 Linux 的管道缓冲机制，所以，它的两个文件描述符参数中至少有一个必须是管道设备。

splice 提供了一种流控制的机制，通过预先定义的水印（watermark）来阻塞写请求，有实验表明，利用这种方法将数据从一个磁盘传输到另外一个磁盘会增加 30%-70% 的吞吐量，CPU负责也会减少一半。

缺陷：

1）同样只适用于不需要用户态处理的程序

2）传输描述符至少有一个是管道设备。

方法六：写时复制

在某些情况下，内核缓冲区可能被多个进程所共享，如果某个进程想要这个共享区进行 write 操作，由于 write 不提供任何的锁操作，那么就会对共享区中的数据造成破坏，写时复制就是 Linux 引入来保护数据的。

写时复制，就是当多个进程共享同一块数据时，如果其中一个进程需要对这份数据进行修改，那么就需要将其拷贝到自己的进程地址空间中，这样做并不影响其他进程对这块数据的操作，每个进程要修改的时候才会进行拷贝，所以叫写时拷贝。这种方法在某种程度上能够降低系统开销，如果某个进程永远不会对所访问的数据进行更改，那么也就永远不需要拷贝。

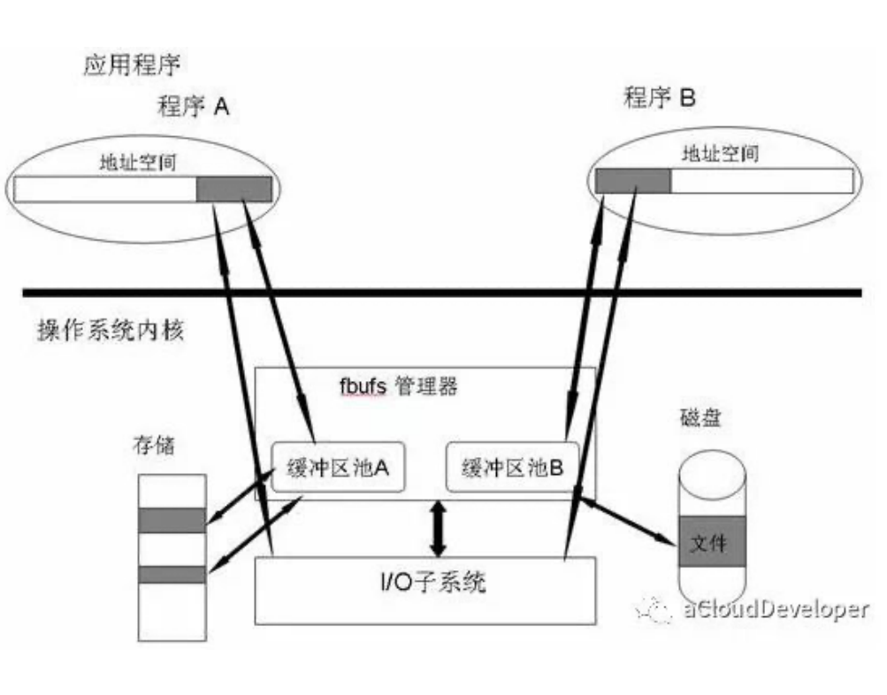
缺陷：

需要 MMU 的支持，MMU 需要知道进程地址空间中哪些页面是只读的，当需要往这些页面写数据时，发出一个异常给操作系统内核，内核会分配新的存储空间来供写入的需求。

方法七：缓冲区共享

这种方法完全改写 I/O 操作，因为传统 I/O 接口都是基于数据拷贝的，要避免拷贝，就去掉原先的那套接口，重新改写，所以这种方法是比较全面的零拷贝技术，目前比较成熟的一个方案是最先在 Solaris 上实现的 fbuf （Fast Buffer，快速缓冲区）。

Fbuf 的思想是每个进程都维护着一个缓冲区池，这个缓冲区池能被同时映射到程序地址空间和内核地址空间，内核和用户共享这个缓冲区池，这样就避免了拷贝。



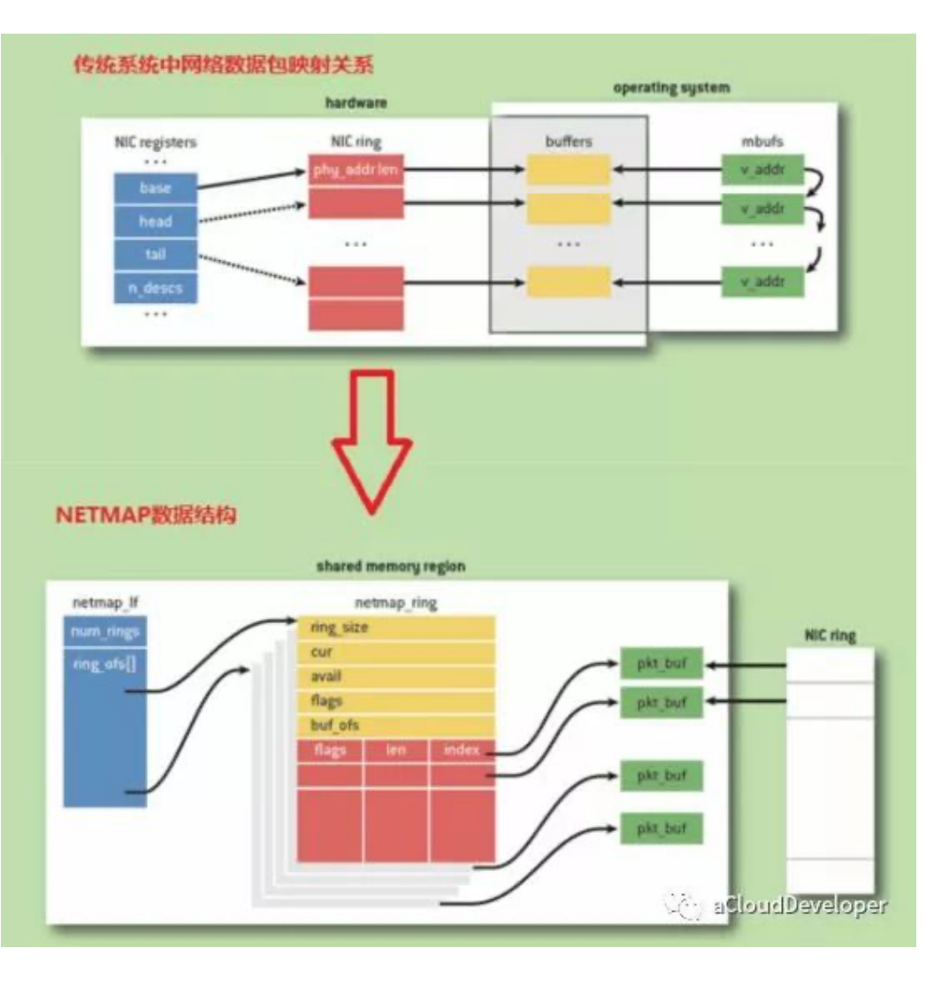
缺陷：

1）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

2）改写 API ，尚处于试验阶段。

高性能网络 I/O 框架——netmap

Netmap 基于共享内存的思想，是一个高性能收发原始数据包的框架，由Luigi Rizzo 等人开发完成，其包含了内核模块以及用户态库函数。其目标是，不修改现有操作系统软件以及不需要特殊硬件支持，实现用户态和网卡之间数据包的高性能传递。



在 Netmap 框架下，内核拥有数据包池，发送环\接收环上的数据包不需要动态申请，有数据到达网卡时，当有数据到达后，直接从数据包池中取出一个数据包，然后将数据放入此数据包中，再将数据包的描述符放入接收环中。内核中的数据包池，通过 mmap 技术映射到用户空间。用户态程序最终通过 netmap_if 获取接收发送环 netmap_ring，进行数据包的获取发送。

总结

1、零拷贝本质上体现了一种优化的思想

2、直接 I/O，mmap，sendfile，DMA sendfile，splice，缓冲区共享，写时复制……