零拷贝技术

Zero Copy

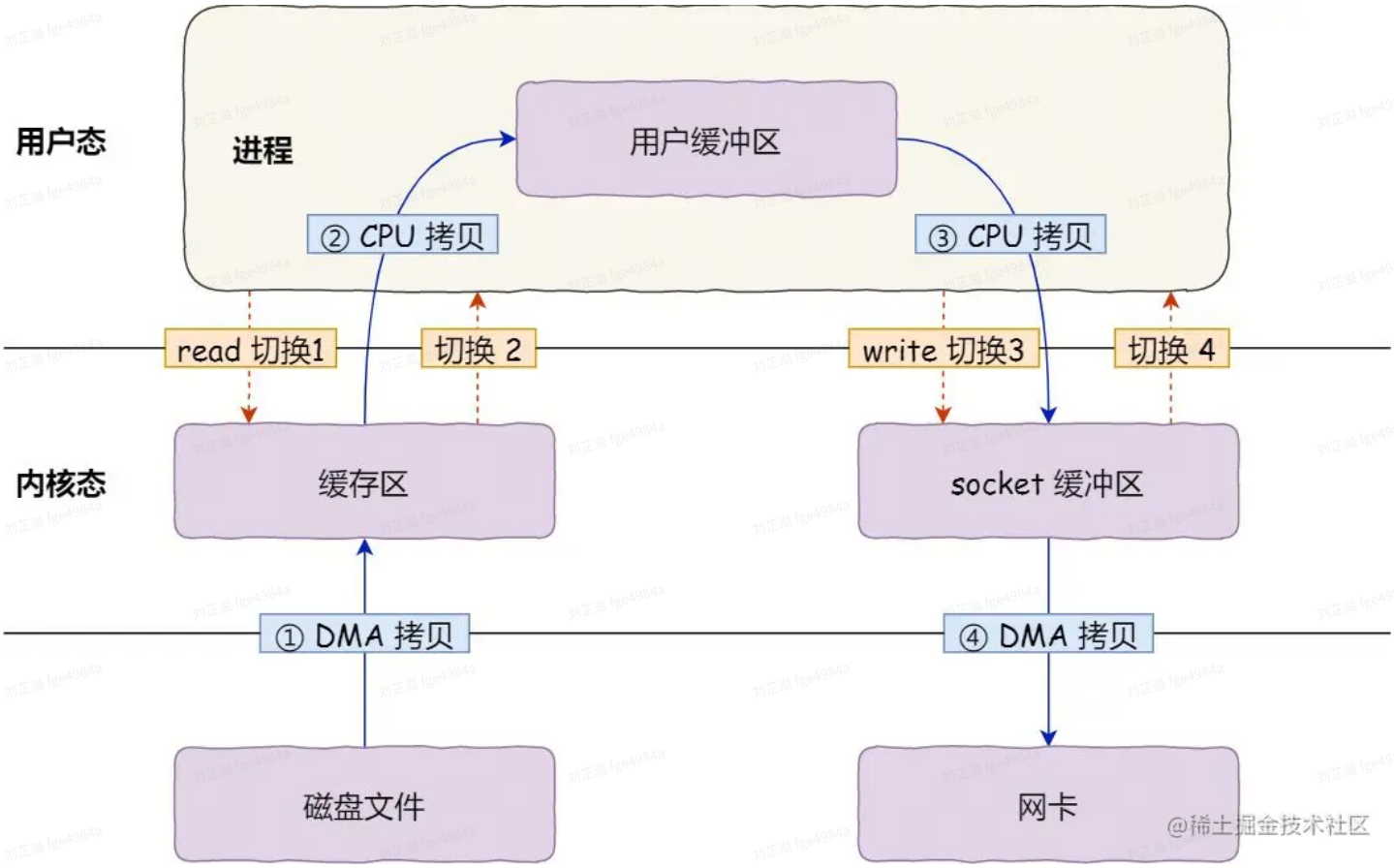
- 零拷贝（Zero-Copy）是一种 `I/O` 操作优化技术，可以快速高效地将**数据从文件系统移动到网络接口**，而不需要将其从内核空间复制到用户空间。其在 `FTP` 或者 `HTTP` 等协议中可以显著地提升性能。
- 因为两次 DMA 都是依赖硬件完成的。所以，**所谓的零拷贝，都是为了减少 CPU copy 及减少了上下文的切换。**
- 但是需要注意的是，并不是所有的操作系统都支持这一特性，目前只有在使用 `NIO` 和 `Epoll` 传输时才可使用该特性。

传统I/O操作

- 传统 I/O 的工作方式是，数据读取和写入是从用户空间到内核空间来回复制，而内核空间的数据是通过操作系统层面的 I/O 接口从磁盘读取或写入。
- 需要两次系统调用 `read` 和 `write`

```
1   read(file, buf, len);
2   write(socket, buf, len);
```

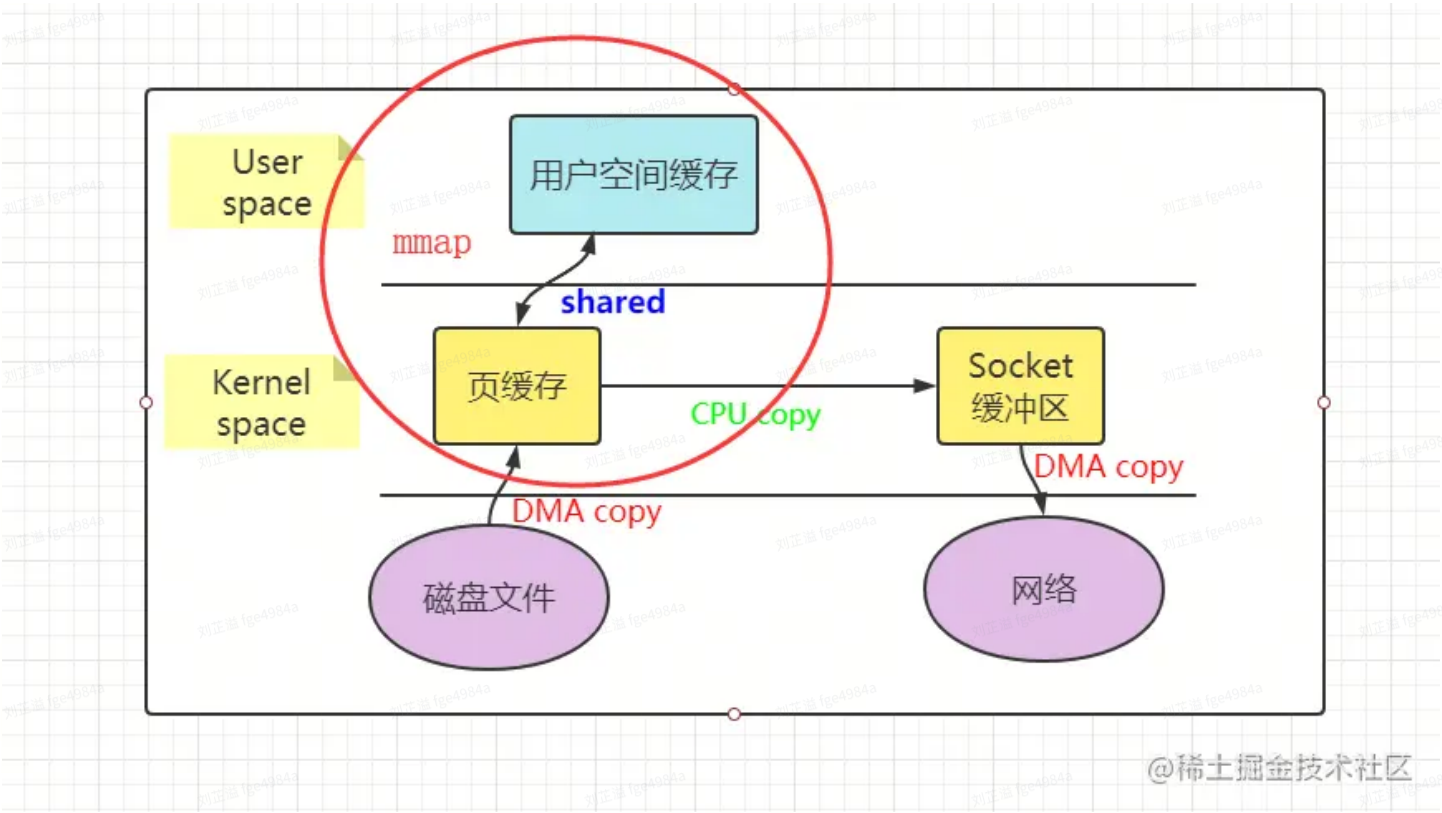
- 发生了四次上下文切换与四次数据拷贝
  - `第一次拷贝`，把磁盘上的数据拷贝到操作系统内核的缓冲区里，这个拷贝的过程是通过 DMA 搬运的。
  - `第二次拷贝`，把内核缓冲区的数据拷贝到用户的缓冲区里，于是我们应用程序就可以使用这部分数据了，这个拷贝到过程是由 **CPU 完成的**。
  - `第三次拷贝`，把刚才拷贝到用户的缓冲区里的数据，再拷贝到内核的 socket 的缓冲区里，这个过程依然还是由 **CPU 搬运**的。
  - `第四次拷贝`，把内核的 socket 缓冲区里的数据，拷贝到网卡的缓冲区里，这个过程又是由 **DMA 搬运**



零拷贝实现方式

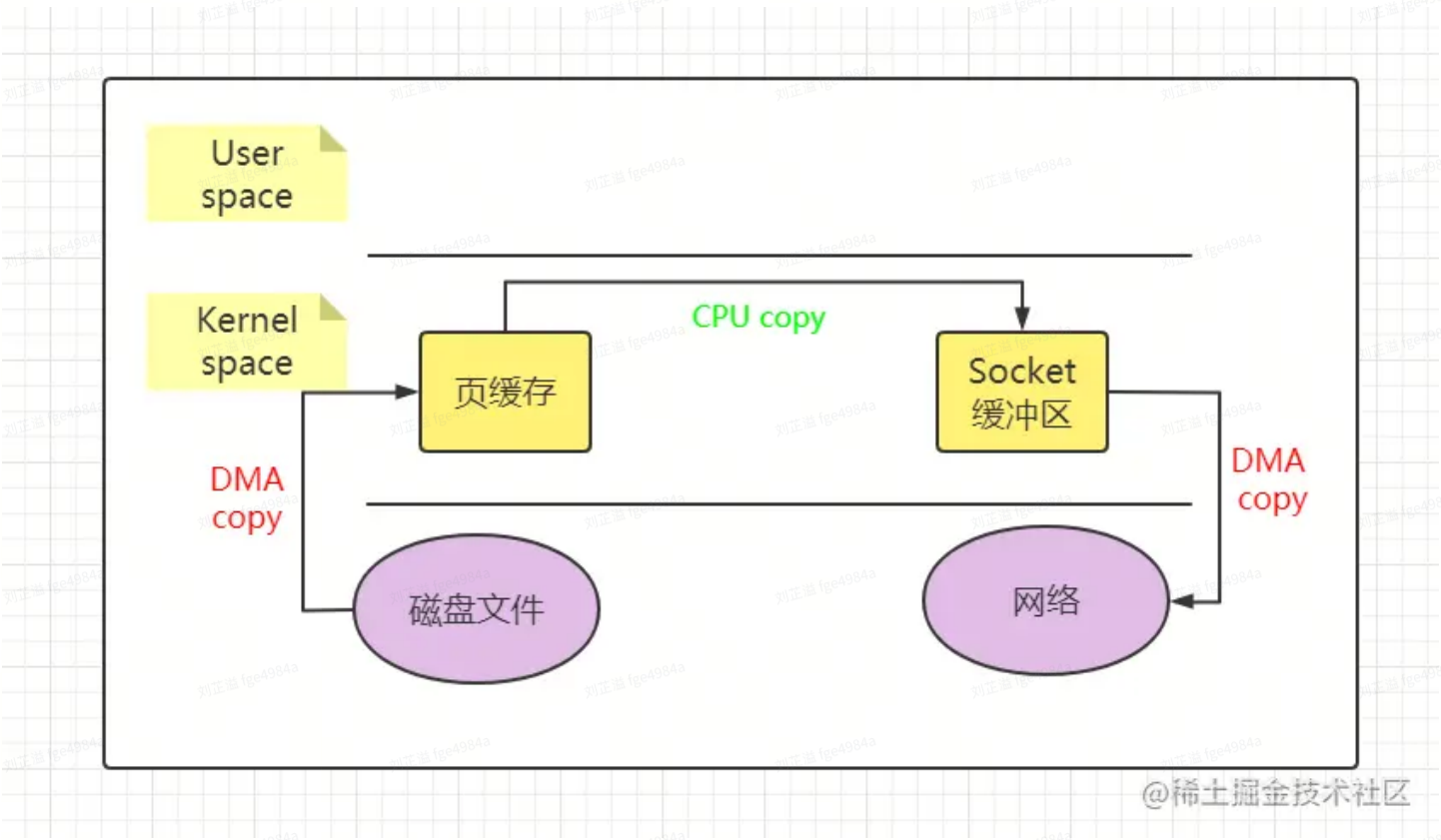
mmap + write

- `mmap` 把内核空间和用户空间的虚拟地址映射到同一个物理地址；只需要操作内核缓冲区就可以了
- 把数据读取到内核缓冲区后，应用程序进行写入操作时，直接把内核的 `Read Buffer` 的数据复制到 `Socket Buffer` 以便写入，这次内核之间的复制也是需要CPU的参与的。
- 这种方式减少了两次数据拷贝，但是还需要进行4次上下文切换



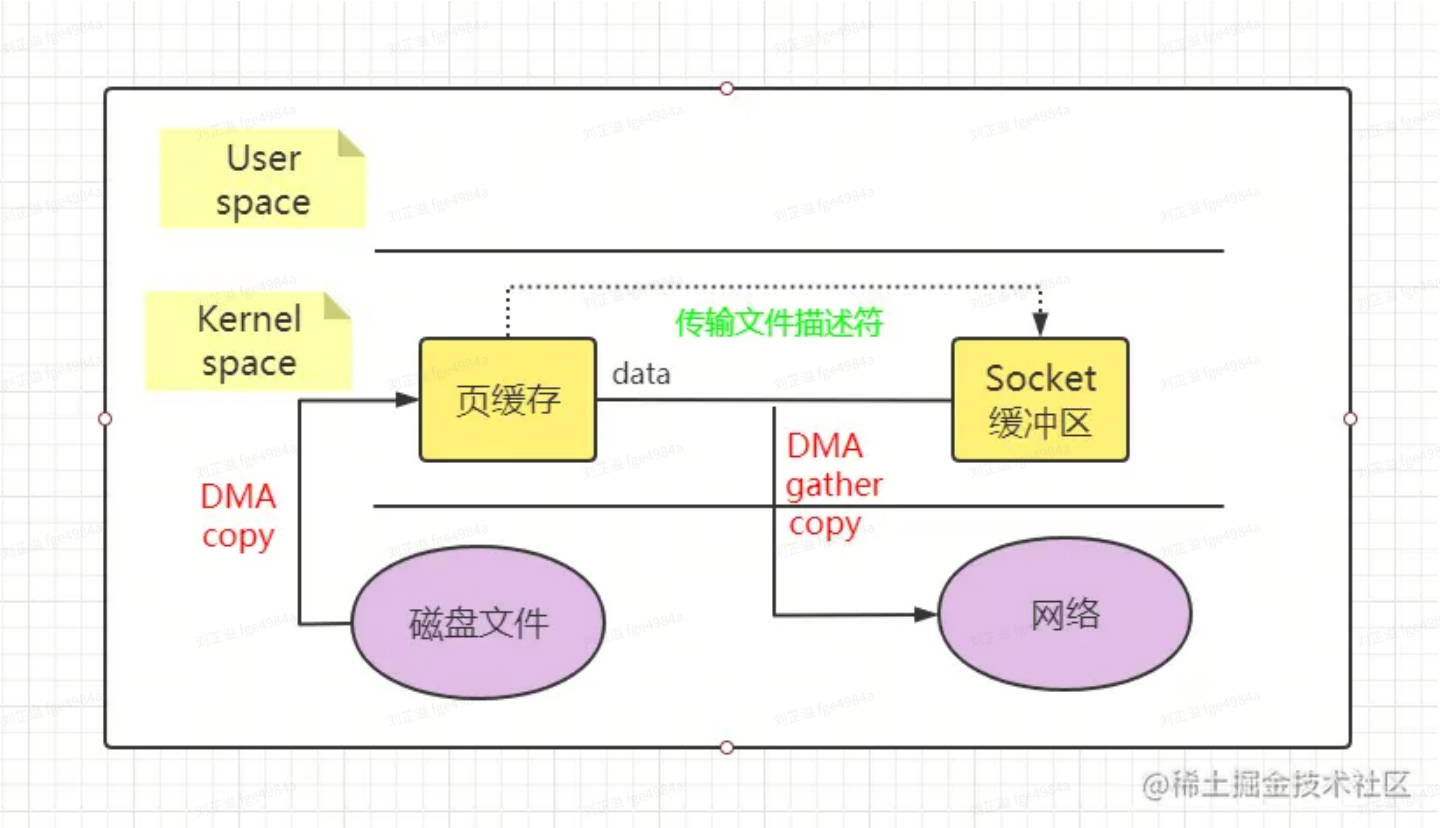
sendfile

- 将 `mmap` + `write` 优化为 `sendfile`；减少了上下文切换，因为少了一个应用程序发起 `write` 操作，直接发起 `sendfile` 操作。
- `sendfile` 方式只有三次数据复制（其中只有一次 CPU COPY）以及2次上下文切换。



带有 scatter/gather 的 sendfile方式

- 这个操作可以把最后一次 `CPU COPY` 去除
- 其原理就是在内核空间 Read BUffer 和 Socket Buffer 不做数据复制，而是将 Read Buffer 的内存地址、偏移量记录到相应的 Socket Buffer 中，这样就不需要复制。
- scatter/gather 的 sendfile 只有两次数据复制（都是 DMA COPY）及 2 次上下文切换。CPU COPY 已经完全没有。不过这一种收集复制功能是需要硬件及驱动程序支持的。



splice

- 用户应用程序必须拥有两个已经打开的文件描述符，一个表示输入设备，一个表示输出设备。
- 与 `sendfile` 不同的是， `splice` 允许任意两个文件互相连接
- 使用 `splice()` 发送文件时，我们并不需要将文件内容读取到用户态缓存中，但需要**使用管道(*内部使用环形缓冲区进行读写，初始时不分配内存，有进程写再进行分配*)作为中转**
- 其实这里的管道只是作为一个通道，并不会产生数据拷贝的