Linux 文件 I/O 进化史（二）：mmap

mmap

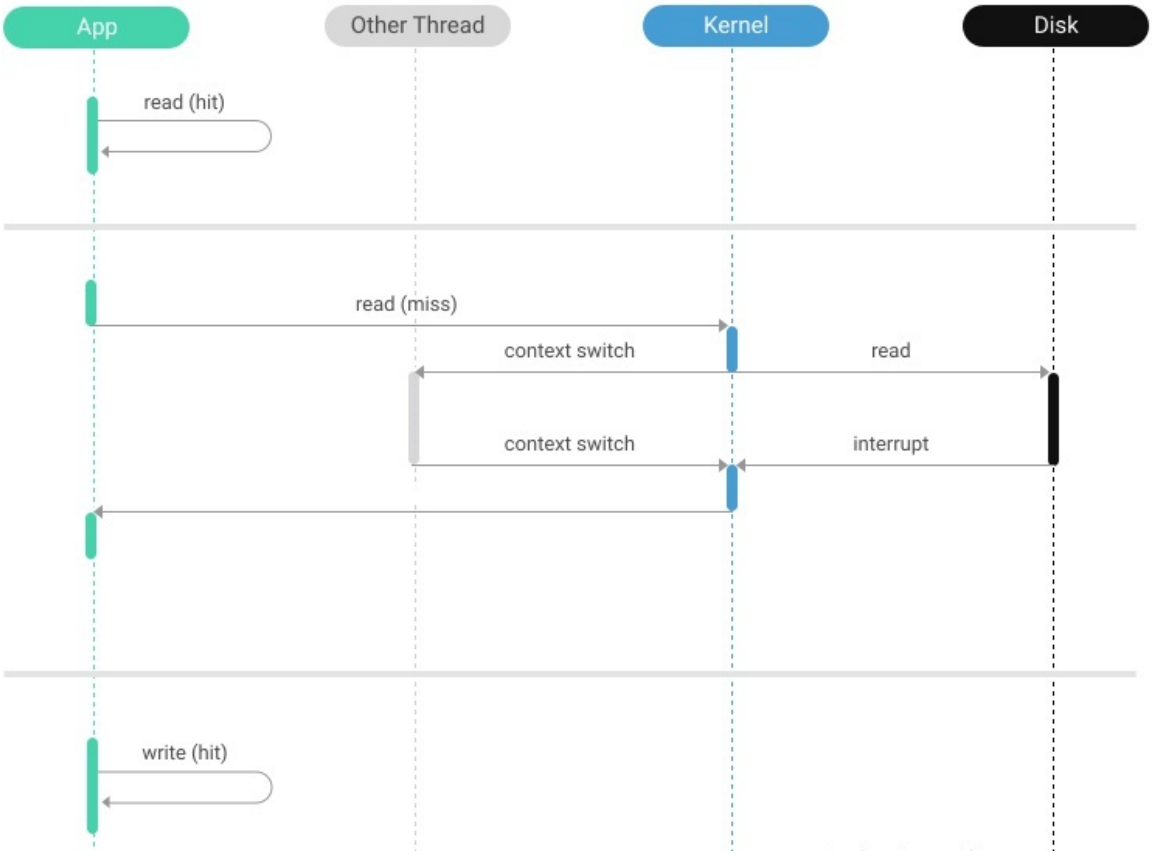
mmap 可以将文件或设备映射到内存中，使应用程序可以像读写内存一样读写文件。

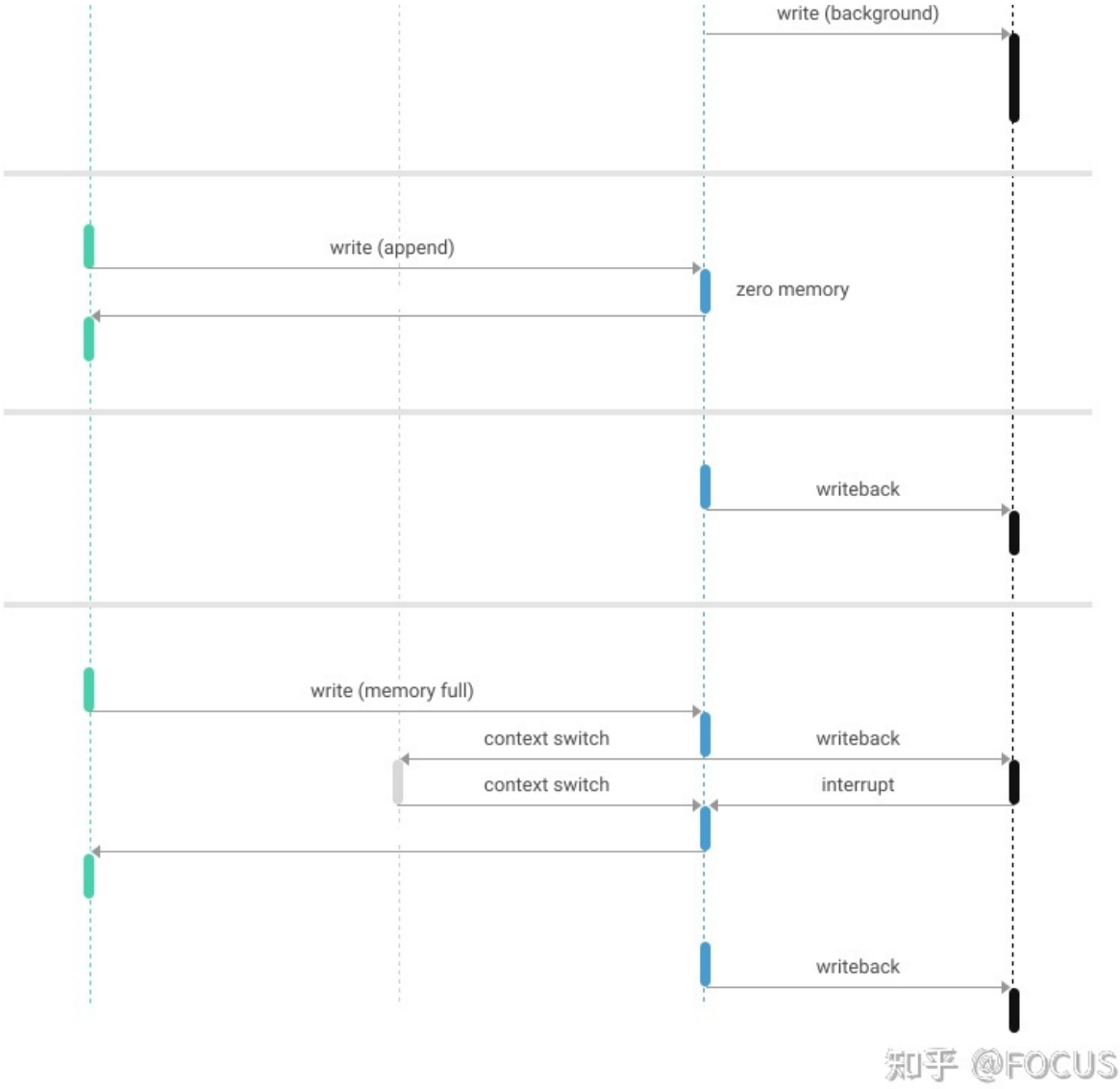
```
#include <sys/mman.h>

void *mmap(void *addr, size_t length, int prot, int flags,
           int fd, off_t offset);
int munmap(void *addr, size_t length);
```

具体参数说明可以参考 Linux Programmer's Manual: mmap(2)。

使用 mmap 之后，如果访问的数据刚好在内存（page cache）中，读写文件和读写内存没什么差别（不需要经过系统调用）。

如果访问的数据不在内存（page cache）中，则会产生一个缺页中断（page fault），更精确一点是 major page fault，此时会产生文件 I/O，线程会被阻塞，发生上下文切换。





（图片来自 Scylla）

优缺点

使用 mmap 访问文件的好处很明显：

1. 读写文件不需要使用 read/write 系统调用。
2. 可以减少用户空间和内核空间的内存拷贝。

同时 mmap 也存在一些缺点：

1. 只支持定长文件（为了支持变长文件，可能可以使用 mremap，但是我几乎没见过有人这样做）。
2. 映射大量文件或大文件可能使页表的开销变大。关于页表介绍，可以参考我之前写的一篇文章：Linux 内存管理。

其它

内核为每个进程维护一个任务结构 task_struct。task_struct 中的 mm_struct 描述了虚拟内存的信息。mm_struct 中的 mmap 字段是一个 vm_area_struct 指针。内核中的 vm_area_struct 对象被组织成一个链

表 + 红黑树的结构。理论上，进程调用一次 mmap 就会产生一个 vm_area_struct 对象（不考虑内核自动合并相邻且符合条件的内存区域）。

实际上，Linux 把二进制程序加载到内存中采用的就是 mmap。因为二进制程序是区分代码、数据段的，这里的就不是简单的整个文件的映射了，需要将文件的分段区域映射到内存的不同位置。mmap 让虚拟空间和文件内容对应起来，具体的进程内存布局可以参考我之前写的一篇文章：Linux 内存管理。

可以通过 `cat /proc/<pid>/maps` 看到某个进程的 mmap 情况。

参考资料

1. Different I/O Access Methods for Linux, What We Chose for Scylla, and Why
2. Linux Programmer's Manual: mmap(2)