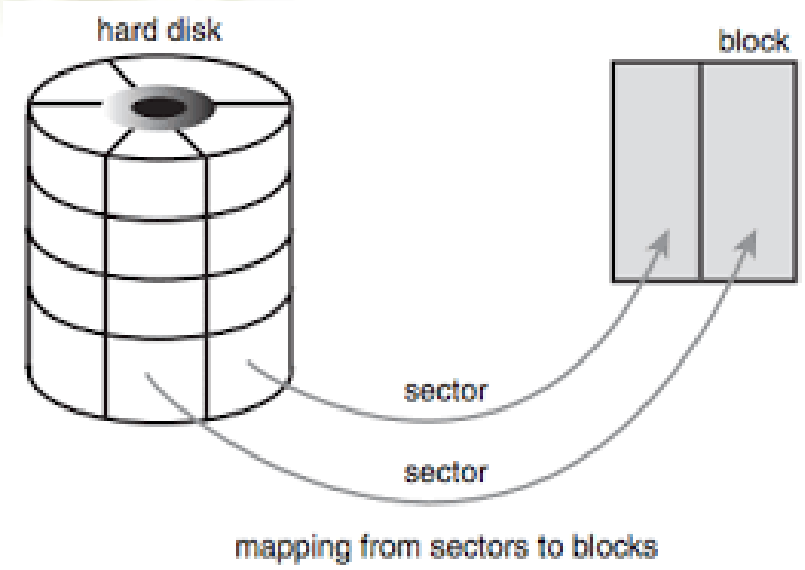
8.4 页高速缓存以及读写

西安邮电大学

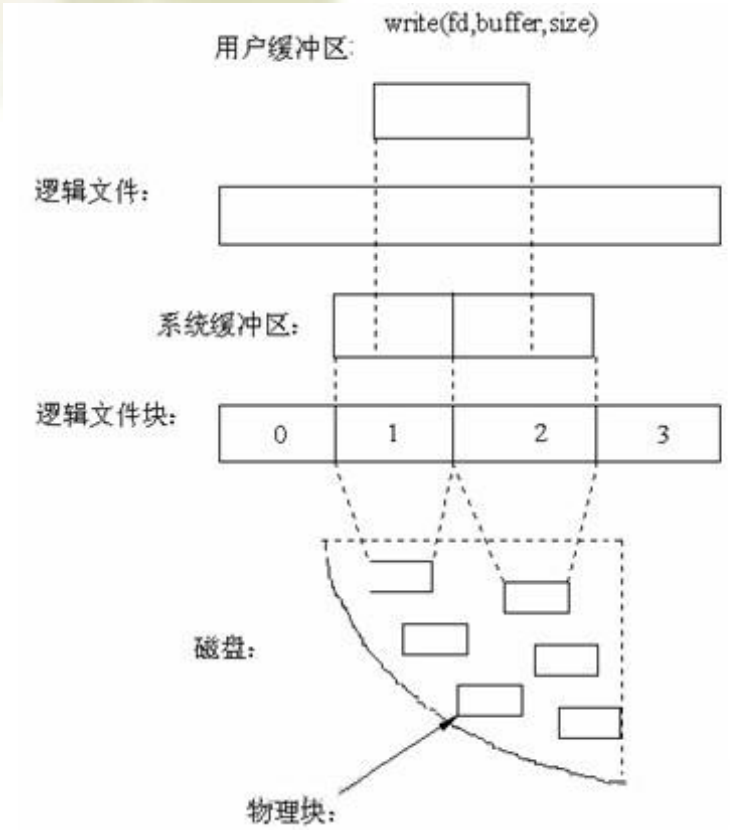
文件系统的读写单位是什么



mapping from sectors to blocks

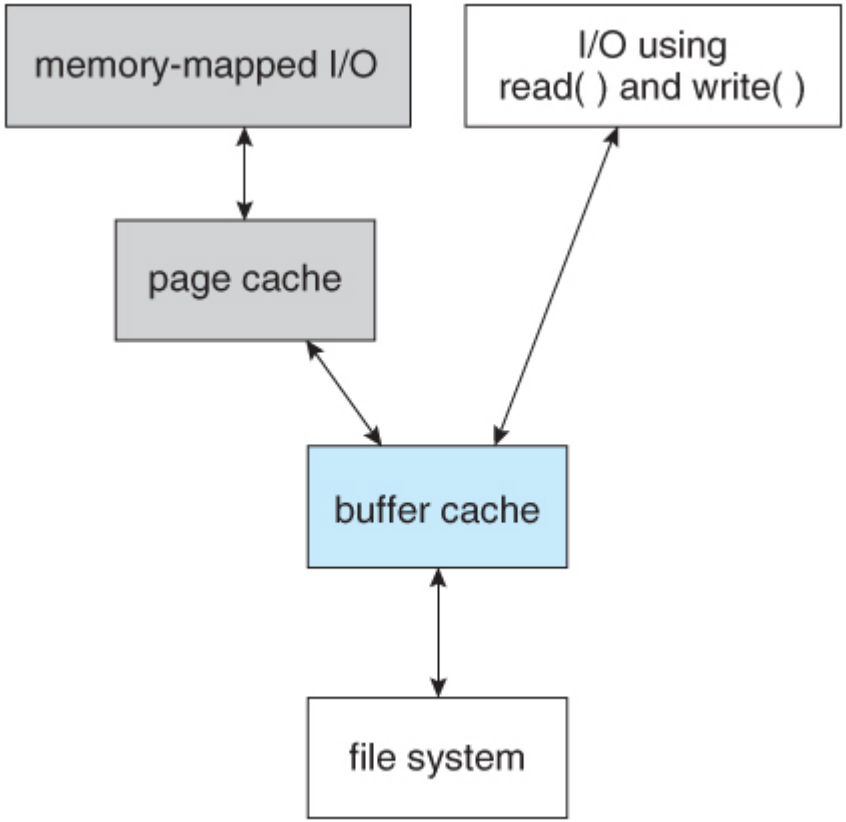
文件系统的读写单位是块，一个块的大小是2的n次方个扇区，比如1k，2k，4k，4M 等

一个进程发出读写请求到读到数据



比如图中，当用户通过write系统调用给文件中写数据时，首先与用户态的I/O缓冲区打交道，然后逻辑文件中的写指针进行移动，真正的写入还没有发生，然后陷入内核态，用户态缓冲区的数据被搬到内核缓冲区，这个缓冲区就是page cache，到最后到底是如何把数据真正写入磁盘，我们继续探究下去。

从Buffer Cache到 Page Cache



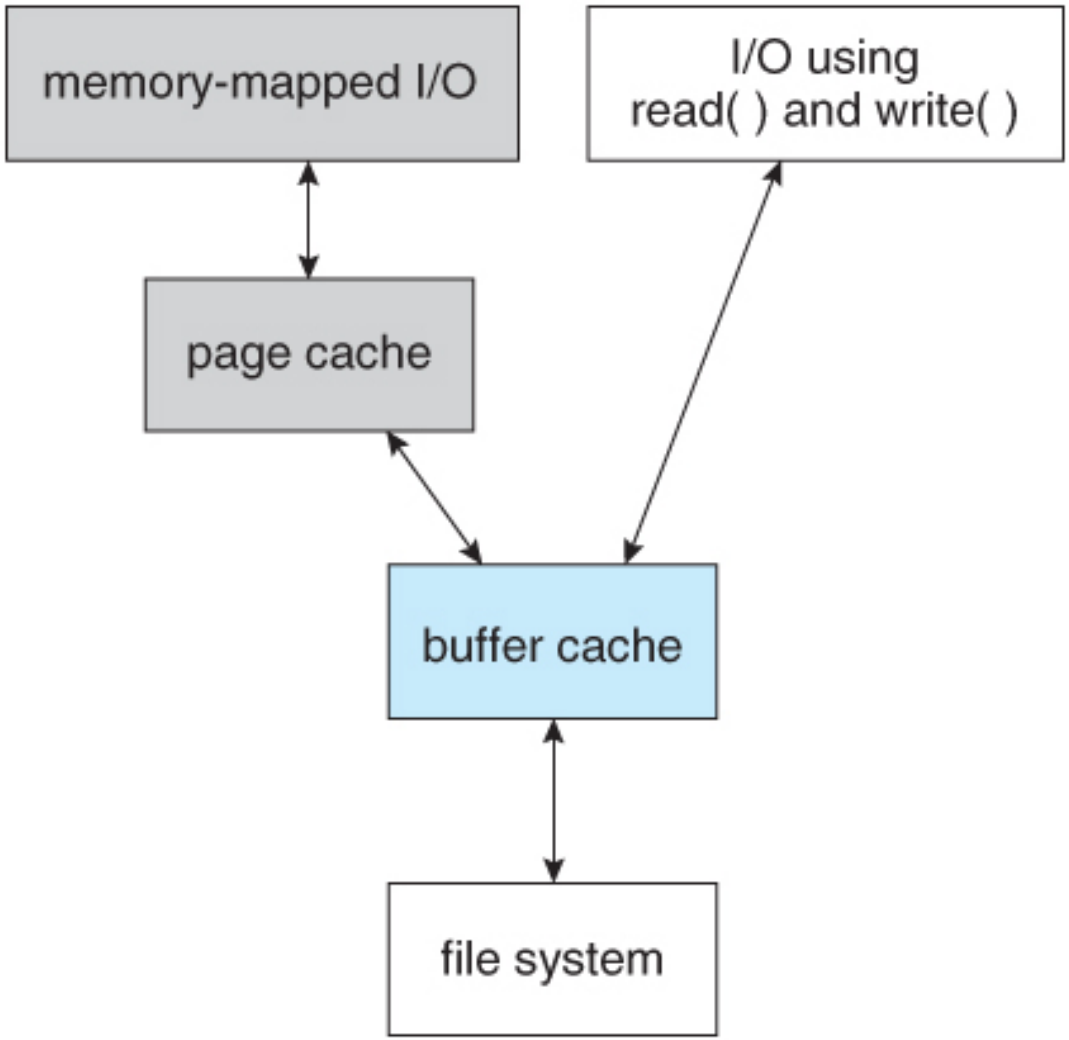
buffer cache是面向底层块设备的，所以它的粒度是文件系统的块，块设备和系统采用块进行交互。块再转换成磁盘的基本物理结构扇区。扇区和块之间是可以快速转换的，这就涉及页高速缓存。

从Buffer Cache到 Page Cache

随着内核的功能越来越完善，块粒度的缓存已经不能满足性能的需要。内核的内存管理组件采用了比文件系统的块更高级别的抽象-页(page)，页的大小一般从4KB到2MB,粒度更大，处理的性能更高。所以缓存组件为了和内存管理组件更好地交互，创建了页缓存page cache来代替原来的buffer cache。

页缓存是面向文件，面向内存的。通过一系列的数据结构，比如inode，address_space，page，将一个文件映射到页的级别，通过page + offset就可以定位到一个文件的具体位置。

从Buffer Cache到 Page Cache



描述页缓存的address_space对象

```
struct address_space {
        struct inode            *host;          /* 所有者：inode，或块设备 */
        struct radix_tree_root  page_tree;      /* 所有页的基数树 */
        unsigned int            i_mmap_writable;/* VM_SHARED映射的计数 */
        struct prio_tree_root   i_mmap;         /* 私有和共享映射的树 */
        struct list_head        i_mmap_nonlinear;/* VM_NONLINEAR映射的链表元素 */
        unsigned long     wbri_banrpages;       /* 页的总数 */
        pgoff_t                 writeback_index;/* 回写由此开始 */
        struct address_space_operations *a_ops; /* 方法，即地址空间操作 */
        unsigned long           flags;          /* 错误标志位/gfp掩码 */
        struct backing_dev_info *backing_dev_info; /* 设备预读 */
        struct list_head        private_list;
        }
```

其中，第一个字段是host，每一个所有者可以理解为一个具体的文件，也就是一个inode指向的文件，它对应着一个address_space对象，页高速缓存的多个页可能属于一个所有者，从而可以链接到一个address_space对象。那么一个页（page）怎么和一个address_space产生关联的呢？

page对象

```
struct page {
        unsigned long          flags;
        atomic_t               _count;
        atomic_t               _mapcount;
        unsigned long          private;
        struct address_space   *mapping;
        pgoff_t                index;
        struct list_head       lru;
        void                   *virtual;
};
```

page中有两个字段：mapping和index。其中mapping指向该页所有者的address_space，index字段表示所有者地址空间中以页大小为单位的偏移量。用这两个字段就能在页高速缓存中查找。

索引节点、页和页缓存之间的关系

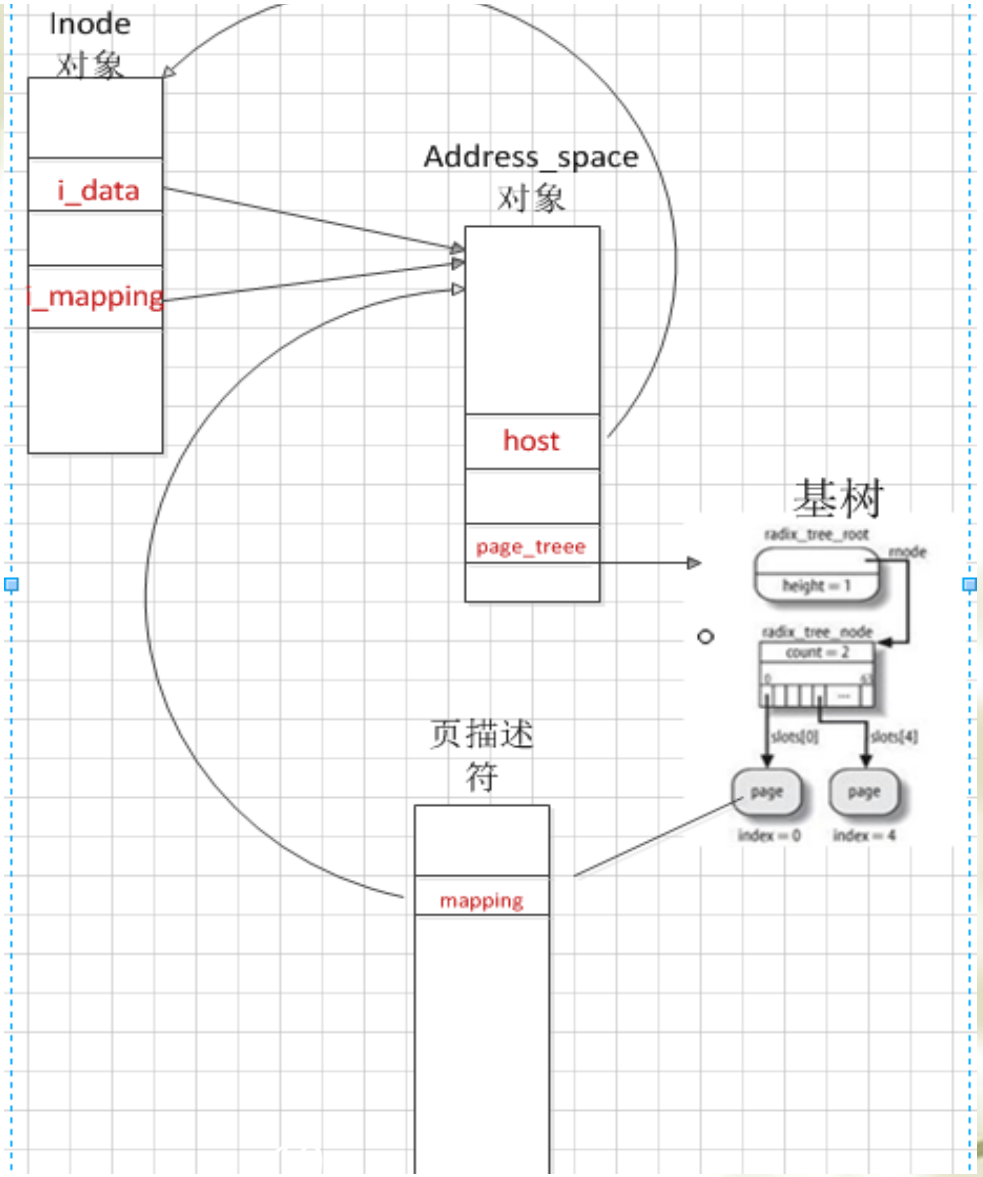
（1）一个inode节点对象对应一个address_space对象。其中inode节点对象的i_mapping和i_data字段指向相应的address_space对象，而address_space对象的host字段指向对应的inode节点对象。

（2）每个address_space对象对应一颗基树。他们之间的联系是通过address_space对象中的page_tree字段指向该address_space对象对应的基树。

（3）一般情况下一个inode节点对象对应的文件或者是块设备都会包含多个页面的内容，所以一个inode对象对应多个page描述符。同一个文件拥有的所有page描述符都可以在该文件对应的基树中找到。

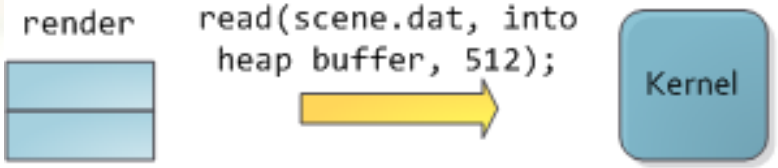
它们之间的关系可以用如图来简单的进行描述。

索引节点、页和页缓存之间的关系

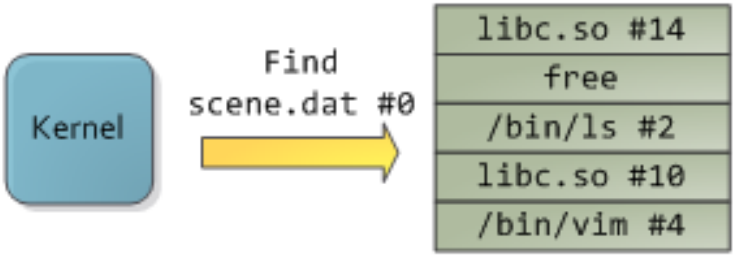


如何读取一个文件？

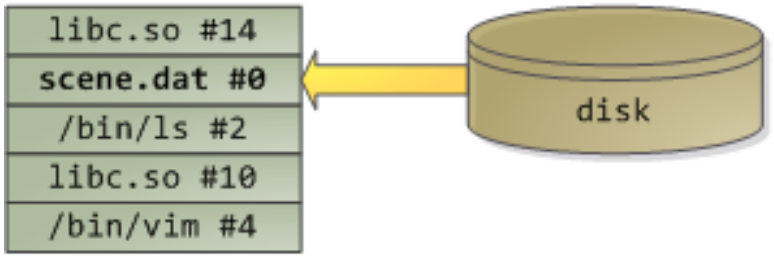
1. Render asks for 512 bytes of scene.dat starting at offset 0.



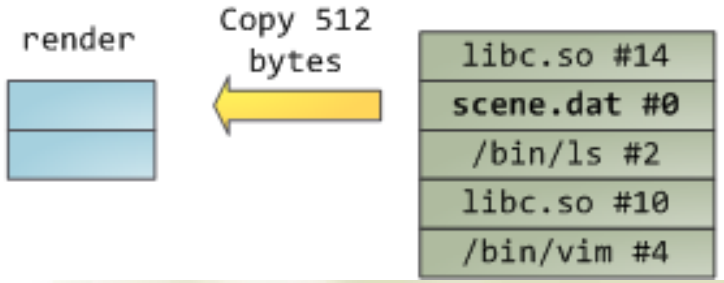
2. Kernel searches the page cache for the 4KB chunk of scene.dat satisfying the request. Suppose the data is not cached.



3. Kernel allocates page frame, initiates I/O requests for 4KB of scene.dat starting at offset 0 to be copied to allocated page frame



4. Kernel copies the requested 512 bytes from page cache to user buffer, read() system call ends.



如何读取一个文件

假设一个进程reader要读取一个my.dat文件，实际发生的步骤如下：

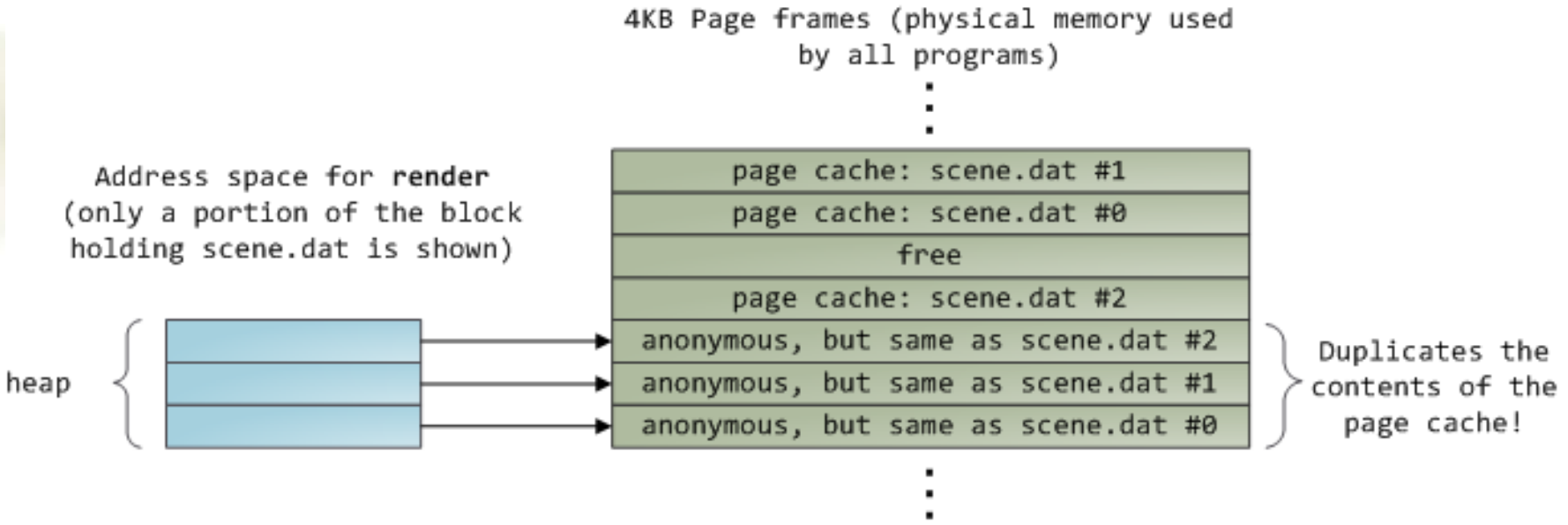
1. reader进程向内核发起读my.dat文件的请求

2. 内核根据my.dat的inode找到对应的address_space，在address_space中查找页缓存，如果没有找到，那么分配一个内存页page加入到页缓存

3. 从磁盘中读取my.dat文件相应的页填充页缓存中的页，也就是第一次复制

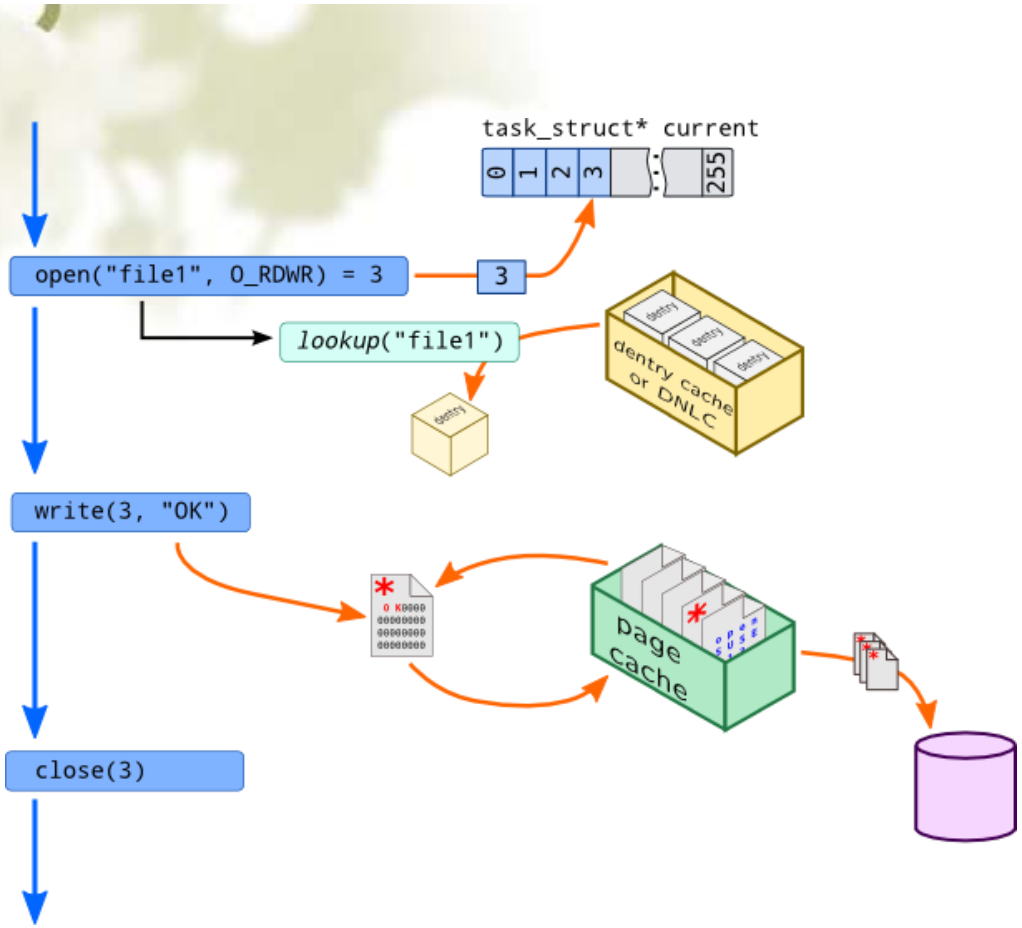
4. 从页缓存的页复制内容到reader进程的堆空间的内存中，也就是第二次复制

如何读取一个文件？



最后物理内存的内容是这样的，同一个文件my.dat的内容存在了两份拷贝，一份是页缓存，一份是用户进程的堆空间对应的物理内存空间。

如何写入文件



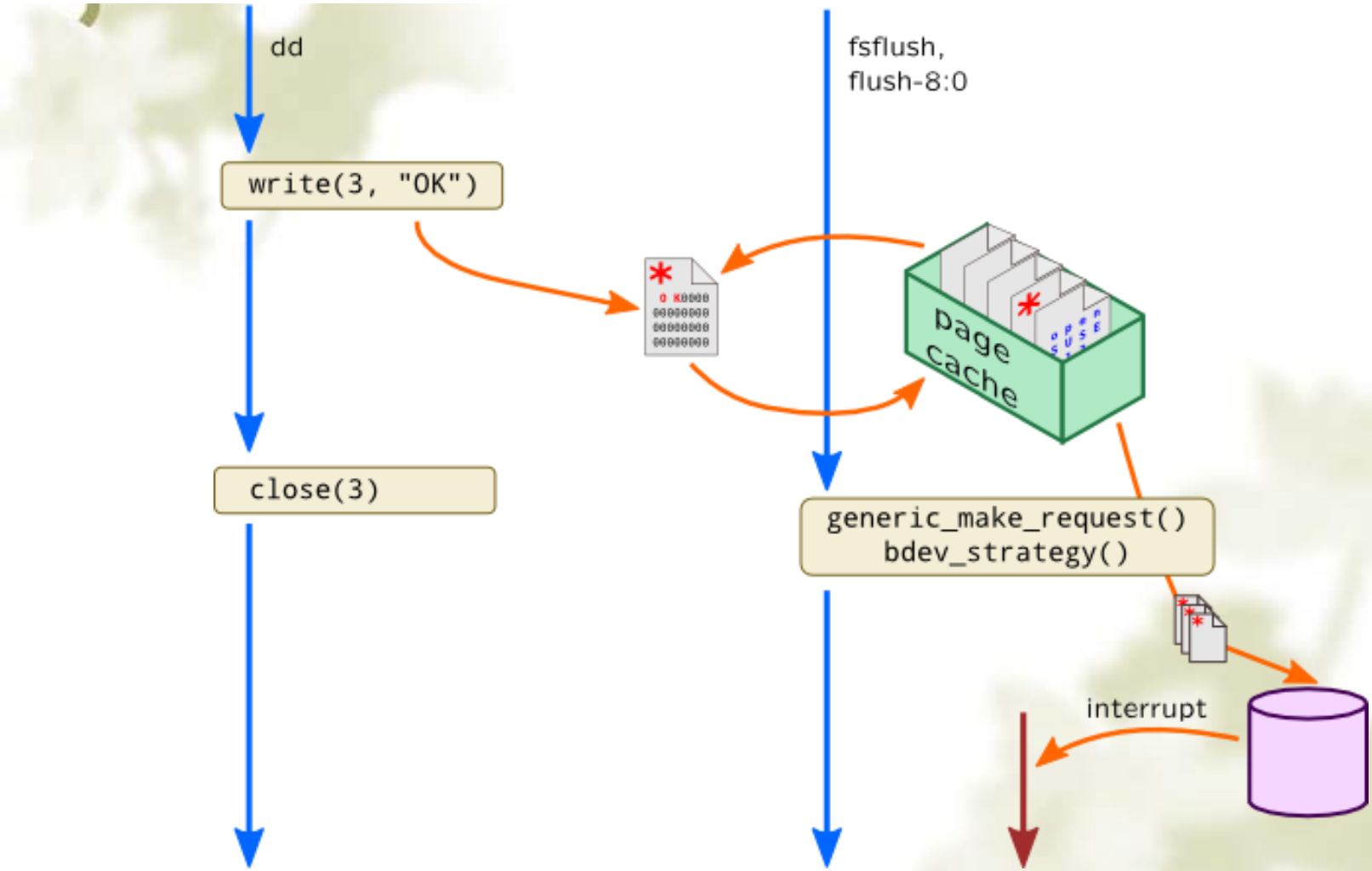
文件的读过程了解后，我们继续来看文件的写入。从用户态来看，文件的写入与读取操作函数原型非常相似，只是数据的流向不一样。

如何写入文件

从内核的角度来看，写文件对应的系统调用服务例程为sys_write。其对应的与读取的函数处理流程上面也非常相似，只是将相应函数中的read改为了write。

对于写操作，Linux同样采用了缓存技术。但这里的缓存跟文件读取用到的有些不一样。对于文件的写入，考虑到一页数据页中，可能只是某一块数据发生改变，这里写的过程结合了块缓冲技术，也就是如果发现了脏页，只是发生写入的脏块回写到磁盘即可。

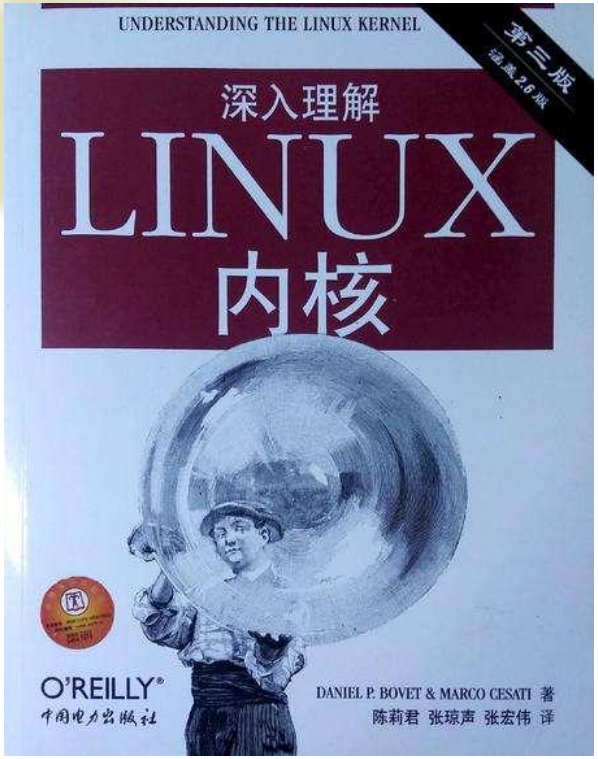
脏页写回

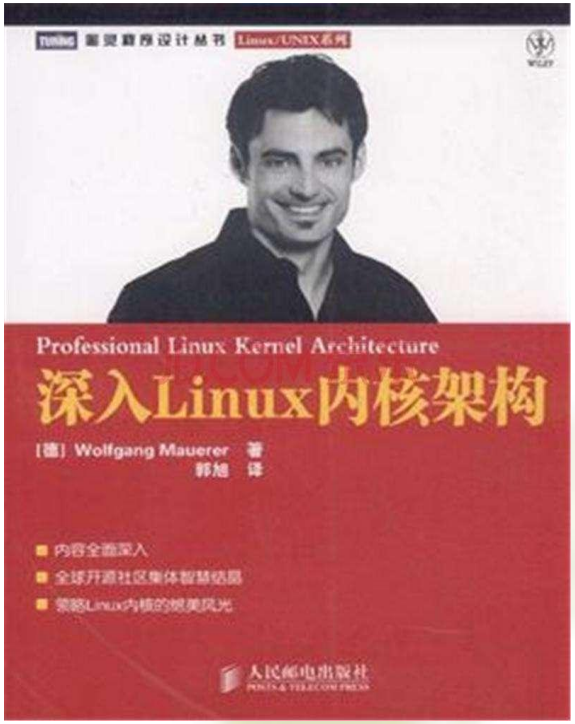

dd
write(3, "OK")
close(3)
fsflush,
flush-8:0
page cache
generic_make_request()
bdev_strategy()
interrupt

脏页回写

Write系统调用并不等数据写入磁盘后才返回。这里Linux采用的是延迟写技术。也就是当write把数据写入到页高速缓冲时，write系统调用将此页标识为脏后，就返回到用户态。而具体的脏页何时写入磁盘，内核有其他的机制负责写入。利用内核线程pdflush，更新页缓冲区中的数据到磁盘提供相应的系统调用如sync，由用户显式地通知内核写回未同步的数据。

参考资料





深入理解Linux内核 第三版第十二章，第十八章

深入Linux内核架构第八章

带着疑问上路

请分析从一个文件读取数据的过程，page cache起什么作用？

谢谢大家！

THANK YOU