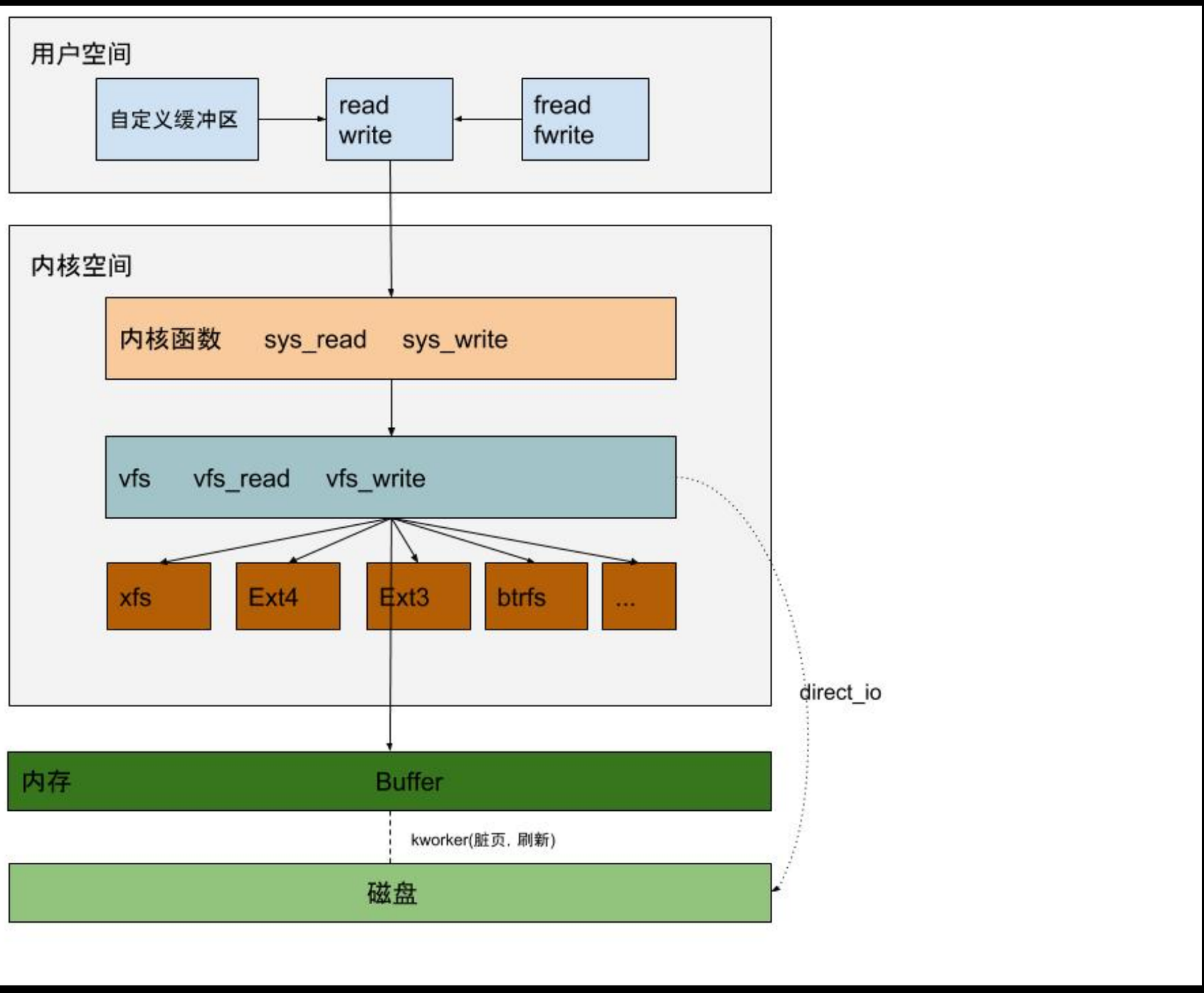
Linux I/O 体系介绍

丁静 dingjingdjdj@gmail.com

Linux 文件系统架构



VFS-虚拟文件交互系统

- VFS 是Linux 内核的子系统，为用户空间程序提供了文件和文件系统相关的接口，负责和实体文件系统打交道。Linux 其他子系统进行文件相关操作时，只与 VFS 交互

- VFS 是文件系统的抽象层，提供了一个通用的文件系统模型，定义了数据结构和接口行为

- 一切都是文件(普通文件，目录，软链，socket, 管道)，目录是一种特殊的文件，存储文件名称和索引节点的映射关系

- VFS 相关的数据结构只存在于内存中

VFS 数据结构

- 超级块对象(superblock)

保存文件系统信息，超级块对象一般对应于存放在磁盘上的文件系统控制块，每个文件系统都应该有一个超级块对象

- 索引节点对象(inode)

保存具体文件的元数据信息，保存在磁盘的文件控制块。每个索引节点对象都有一个索引节点号，用于唯一标识某个文件系统的一个具体文件

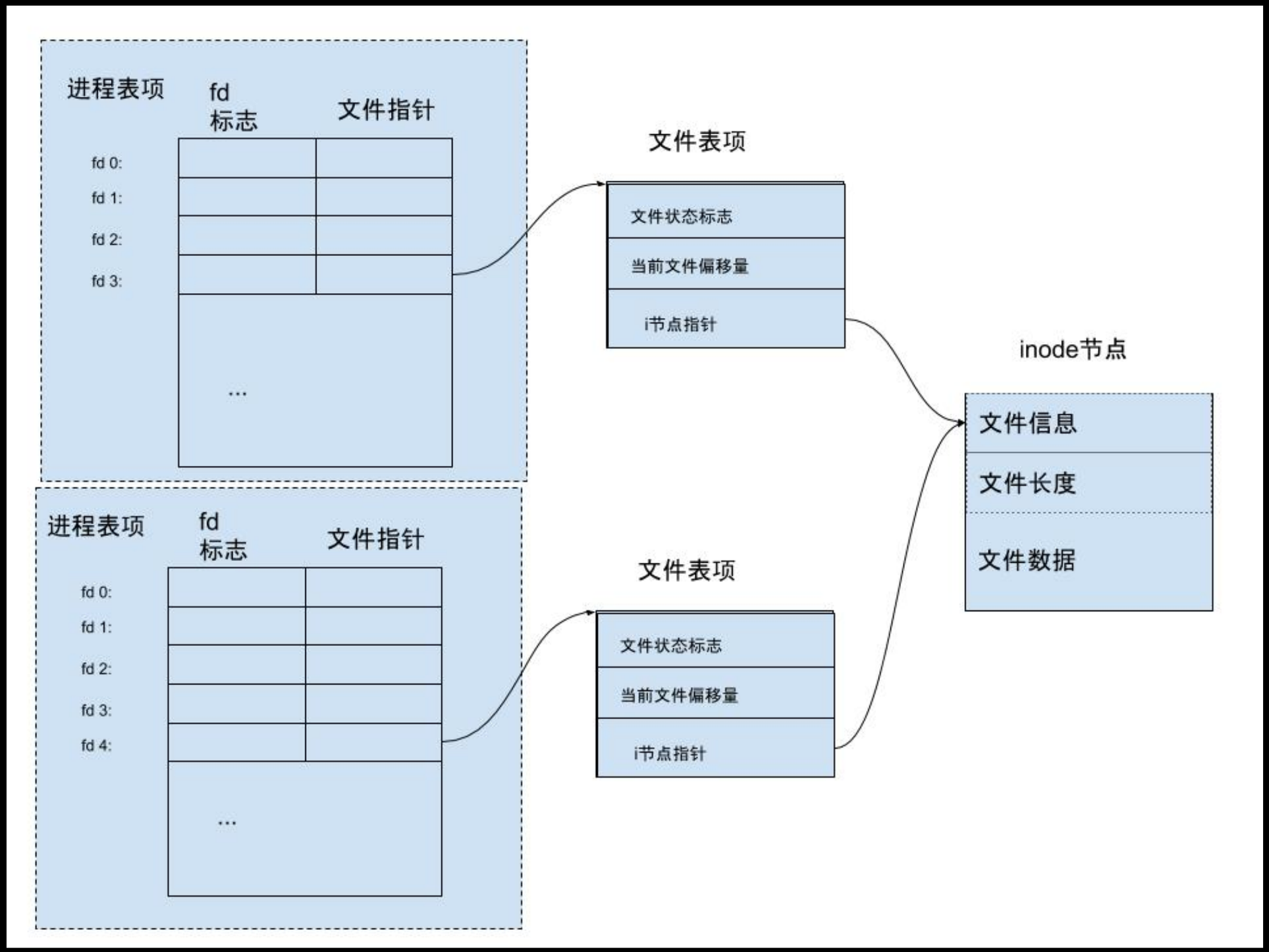
- 目录项对象(dentry)

代表一个目录项，是路径的一个组成部分。完全是在内存中构建的，没有对应的磁盘结构

- 文件对象(file)

用于保存已打开文件与进程直接进行交互的信息。也是完全保存在内存中，且仅当进程访问文件期间有效

进程与文件关系


进程表项
fd
标志
文件指针
fd 0:
fd 1:
fd 2:
fd 3:
...
文件表项
文件状态标志
当前文件偏移量
i节点指针
inode节点
文件信息
文件长度
文件数据
进程表项
fd
标志
文件指针
fd 0:
fd 1:
fd 2:
fd 3:
fd 4:
...
文件表项
文件状态标志
当前文件偏移量
i节点指针

inode对象

- 存储文件的元数据信息，包括inode号码，文件大小，文件读、写、执行权限，文件时间戳(变更时间，修改时间，访问时间)，链接数，文件数据block位置

```
➜  ~ stat  xx.sql
  File: "xx.sql"
  Size: 130983          Blocks: 256        IO Block: 4096   普通文件
Device: fc10h/64528d    Inode: 13500994    Links: 1
Access: (0644/-rw-r--r--)  Uid: (  530/dingjing)   Gid: (  530/dingjing)
Access: 2018-04-23 07:21:43.325609800 +0800
Modify: 2018-04-23 07:21:43.325609800 +0800
Change: 2018-09-06 11:38:03.968989911 +0800
➜  ~ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/vda1        20G   14G  5.7G  70% /
tmpfs           2.0G   24K  2.0G   1% /dev/shm
/dev/vdb        985G  700G  235G  75% /mnt
/dev/vdc        985G  775G  160G  83% /data
cm_processes     24G   79M   24G   1% /var/run/cloudera-scm-agent/process
➜  ~ df -i
Filesystem       Inodes   IUsed    IFree IUse% Mounted on
/dev/vda1       1310720  167098  1143622   13% /
tmpfs           6178045      11  6178034    1% /dev/shm
/dev/vdb       65536000 2928658 62607342    5% /mnt
/dev/vdc       65536000 3280355 62255645    6% /data
cm_processes    6178045     583  6177462    1% /var/run/cloudera-scm-agent/process
➜  ~ sudo dumpe2fs -h /dev/vda1 | grep "Inode size"
dumpe2fs 1.41.12 (17-May-2010)
Inode size:               256
➜  ~ sudo dumpe2fs -h /dev/vdb | grep "Inode size"
dumpe2fs 1.41.12 (17-May-2010)
Inode size:               256
```

-

链接

- 硬链接，多个文件指向同一个inode，会把链接数增加(ln 源文件 目标文件)

- 软链接，不会增加源文件的链接数，只是对源文件的文件名做映射(ln -s 源文件 目标文件)

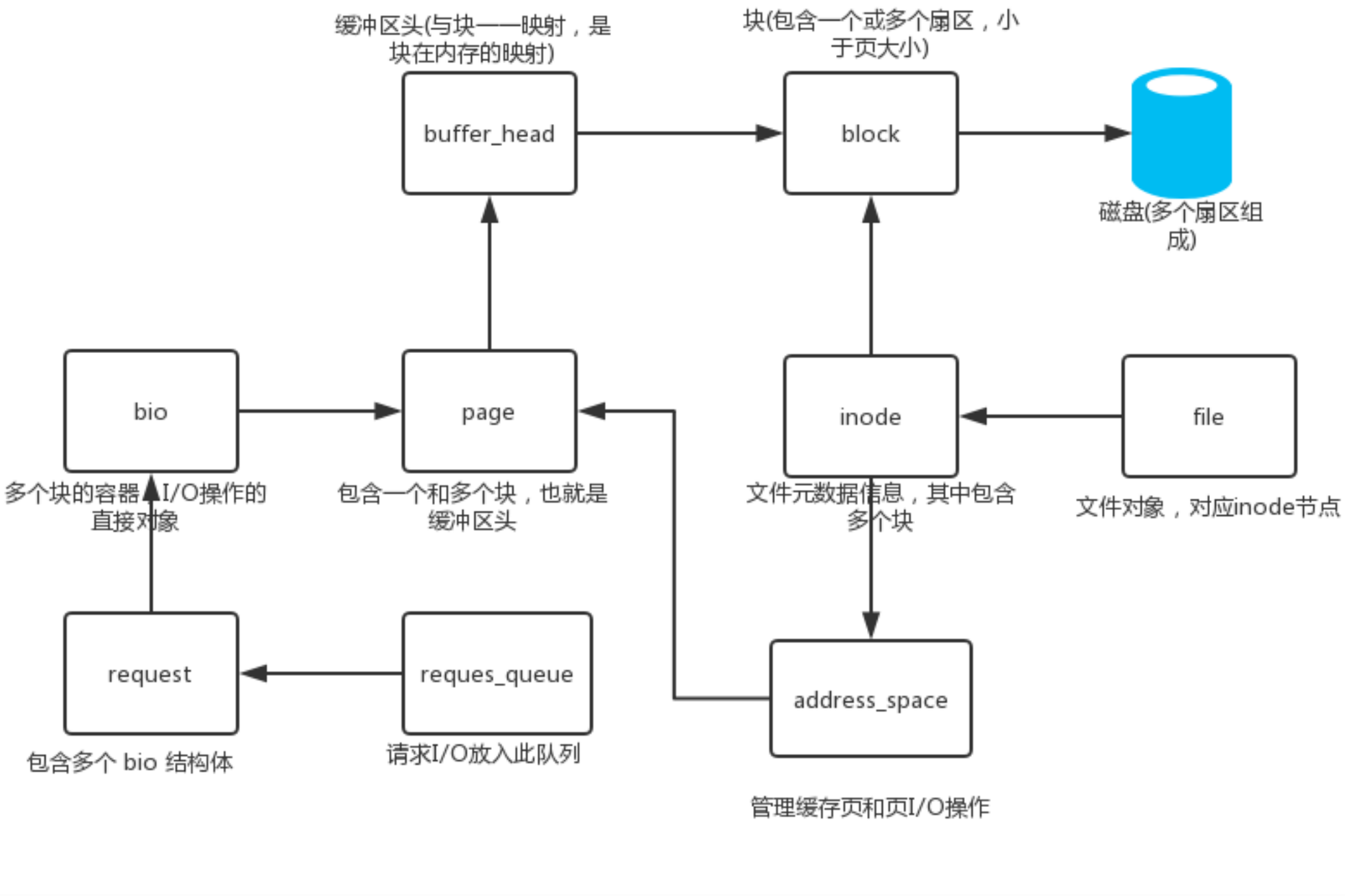
```
→  ~ ln xx.sql xx.sql.bak
→  ~ stat xx.sql
  File: "xx.sql"
  Size: 130983          Blocks: 256        IO Block: 4096   普通文件
Device: fc10h/64528d    Inode: 13500994    Links: 2
Access: (0644/-rw-r--r--)  Uid: (  530/dingjing)   Gid: (  530/dingjing)
Access: 2018-04-23 07:21:43.325609800 +0800
Modify: 2018-04-23 07:21:43.325609800 +0800
Change: 2019-01-21 07:41:01.509698352 +0800
→  ~ stat xx.sql.bak
  File: "xx.sql.bak"
  Size: 130983          Blocks: 256        IO Block: 4096   普通文件
Device: fc10h/64528d    Inode: 13500994    Links: 2
Access: (0644/-rw-r--r--)  Uid: (  530/dingjing)   Gid: (  530/dingjing)
Access: 2018-04-23 07:21:43.325609800 +0800
Modify: 2018-04-23 07:21:43.325609800 +0800
Change: 2019-01-21 07:41:01.509698352 +0800
```

```
→  ~ ln -s xx.sql xx.sql.soft
→  ~ stat xx.sql
  File: "xx.sql"
  Size: 130983          Blocks: 256        IO Block: 4096   普通文件
Device: fc10h/64528d    Inode: 13500994    Links: 2
Access: (0644/-rw-r--r--)  Uid: (  530/dingjing)   Gid: (  530/dingjing)
Access: 2018-04-23 07:21:43.325609800 +0800
Modify: 2018-04-23 07:21:43.325609800 +0800
Change: 2019-01-21 07:41:01.509698352 +0800
→  ~ stat xx.sql.soft
  File: "xx.sql.soft" -> "xx.sql"
  Size: 6               Blocks: 0          IO Block: 4096   符号链接
Device: fc10h/64528d    Inode: 13501562    Links: 1
Access: (0777/lrwxrwxrwx)  Uid: (  530/dingjing)   Gid: (  530/dingjing)
Access: 2019-01-21 07:47:09.762781442 +0800
Modify: 2019-01-21 07:47:09.762781442 +0800
Change: 2019-01-21 07:47:09.762781442 +0800
```

- 删除inode节点，就可以删除文件（特殊文件删除）

- 移动文件，或者重命名文件，只是更改文件名，不影响inode号码(执行速度快)

块 I/O 层


缓冲区头(与块一一映射，是块在内存的映射)
buffer_head
块(包含一个或多个扇区，小于页大小)
block
磁盘(多个扇区组成)
bio
多个块的容器I/O操作的直接对象
page
包含一个和多个块，也就是缓冲区头
inode
文件元数据信息，其中包含多个块
file
文件对象，对应inode节点
request
包含多个 bio 结构体
reques_queue
请求I/O放入此队列
address_space
管理缓存页和页I/O操作

I/O 调度

- 电梯算法 最短寻址
- deadline 照顾读请求，分为写、读、派发队列
- 预测-as
- cfq 完全公平，每个进程维护一个队列，时间轮询
- noop 空操作

脏页

- /proc/sys/vm/dirty_background_ratio 脏数据达到系统内存的百分比，触发后台进程刷新数据到磁盘。缺省值 10。write 系统调用不阻塞
- /proc/sys/vm/dirty_background_bytes 脏数据占用的字节大小，bytes 与 ratio 只能设置一个
- /proc/sys/vm/dirty_ratio 进程中的脏数据占用系统的百分比，触发会刷数据,这是 write系统调用会被阻塞，直到小于这个阈值。
- /proc/sys/vm/dirty_bytes 进程中脏数据的数量，至少两个页面大小
- /proc/sys/vm/dirty_expire_centisecs 脏数据在内存的驻留时间，单位是 1/100 秒，缺省值 3000
- /proc/sys/vm/dirty_writeback_centisecs 刷新进程的唤醒间隔。单位是 1/100 秒，缺省值 500

文件系统的缓存

- echo 3 > /proc/sys/vm/drop_caches, 执行前执行 sync
- slabtop 查看缓存对象
- cat /proc/meminfo
- cat /etc/fstab 查看挂载点，及文件系统信息
-

I/O中的阻塞与非阻塞

- read/write 默认阻塞
- 非阻塞设置标记，轮询
- 轮询耗CPU

```c
#include "apue.h"
#include <errno.h>
#include <fcntl.h>

char    buf[500000];

int
main(void)
{
    int     ntowrite, nwrite;
    char    *ptr;

    ntowrite = read(STDIN_FILENO, buf, sizeof(buf));
    fprintf(stderr, "read %d bytes\n", ntowrite);

    set_fl(STDOUT_FILENO, O_NONBLOCK);  /* set nonblocking */

    ptr = buf;
    while (ntowrite > 0) {
        errno = 0;
        nwrite = write(STDOUT_FILENO, ptr, ntowrite);
        fprintf(stderr, "nwrite = %d, errno = %d\n", nwrite, errno);

        if (nwrite > 0) {
            ptr += nwrite;
            ntowrite -= nwrite;
        }
    }

    clr_fl(STDOUT_FILENO, O_NONBLOCK);  /* clear nonblocking */

    exit(0);
}
```

I/O中的异步形式

- select 阻塞，描述符数量有限制
- poll 阻塞，性能更高
- epoll 回调函数

图 1. 基本 Linux I/O 模型的简单矩阵

	Blocking	Non-blocking
Synchronous	Read/write	Read/wirte (O_NONBLOCK)
Asynchronous	i/O multiplexing (select/poll)	AIO

IO 刷新缓冲区

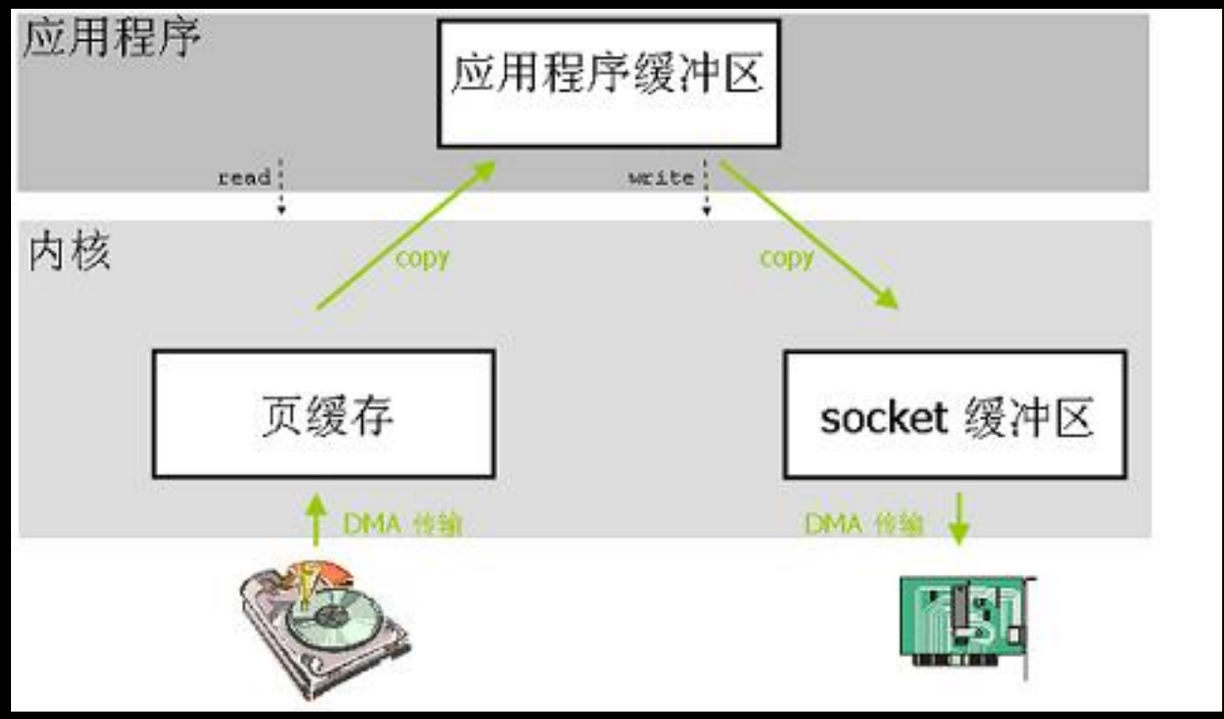
- sync 将所有修改的块缓冲区排入写队列，然后返回，并不等待实际写磁盘结束

- fsync 只对特定的文件描述符fd起作用，等待写磁盘操作结束才返回，元数据和数据内容都更新

- fdatasync 类似于fsync，但只影响文件的数据部分

零拷贝技术

- direct io, 裸IO操作(数据库)
- mmap 内存映射
- sendfile



I/O 性能关注点

- IOPS 每秒的读写次数
- 吞吐量 当前数据传输速率， 单位 B/s
- 读/写比
- IO 大小
- 读写模式 （随机IO优化）
- 磁盘控制器
- 延时离群点

I/O 性能监控

- fio 基准测试
- sar -b (sar -b 1)
- iostat -xzdk 1
- pidstat -d 1
- iotop
- strace -Tfp PID, strace -cfp PID
- lsof -p PID

总结

- 读性能，要关注 cache 命中率，尽量减少冷数据对 cache 的加载，可以通过 DIRECT_IO 的方式避免

- 写性能，要识别出随机IO，有可能利用率 100%, 但是饱和度不够

- 对于同一个文件，inode 对象只有一个，inode 对象通过 address_space 管理 pagecache，如果通过 cache 写，读也会通过 cache 读

- 上面两个图的映射关系很重要，同一个文件，多个进程打开会有多个 file 对象，但是共同引用同一个 inode 对象，inode 对应的 pagecache 是共用的