第三章 数据缓冲区高速缓冲

硬件缓存（cache）

由一种高速寄存器（register）组成，主要解决CPU与RAM之间的速度差问题。

数据缓冲区高速缓冲（buffer）

由软件实现的解决文件系统和物理硬盘之间的数据同步的一种方法。

数据缓冲区高速缓冲是UNIX特有的对数据并发访问的一种控制机制。

问题的提出：

1、磁盘机械运行速度大大低于处理机的运行速度；

2、多进程并发运行，少量的磁盘（通道）I/O成为瓶颈；

3、数据访问的随机性，磁盘忙闲不均

解决办法：

1、建立一个被称为数据缓冲区高速缓冲（简称高速缓冲）的内部数据缓冲区池（buffer pool）来存放要用的数据；

2、写数据时

把数据尽量多地尽量长时间地保存在缓冲池中

延迟写出到磁盘上

以备后续进程使用

3、读数据时

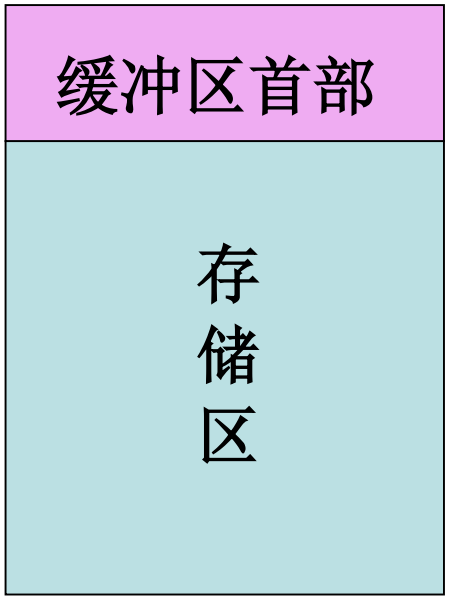
先在缓冲池中查找已有的数据

如没有，再从磁盘读取，并保存在缓冲池中

事先预读数据到缓冲池中

3.1 缓冲区及缓冲区首部

缓冲区池由若干个缓冲区组成，每一个缓冲区又由两部分组成：一个实际存放数据的存储区和一个标识该缓冲区的缓冲区首部。



因为缓冲区首部与数据存储区之间有一一对应的关系，所以通常把两者统称为缓冲区。

缓冲区是缓冲区池中数据存储的基本单位。

缓冲区首部的定义:

```
struct  buf {
        缓冲区标志                          标识缓冲区状态
        缓冲区链接指针                      向前向后串成链表
        空闲缓冲区链表指针                  联结空闲缓冲区
        设备号                              标识缓冲区
        块号
        union{                              缓冲区中的数据类型
                数据块
                超级块
                柱面块
                i节点块
        }b_un
        其它控制信息
}
```

3.2 缓冲池的结构

1、最近最少使用（LRU）算法：

① 程序设计采用模块化和层次化结构，尽量避免使用goto语句，程序跳转少，适应“流水线（pipeline）”体系结构的系统；

② 特定时间段内，程序在一个相对集中空间（代码段）内运行，涉及的数据（广义的：文件名、变量、指针和数组等）的个数相对较少；

③ 当前使用过的数据，马上还要使用的可能性最大，较长时间未用过的数据，即将使用的可能性最小。

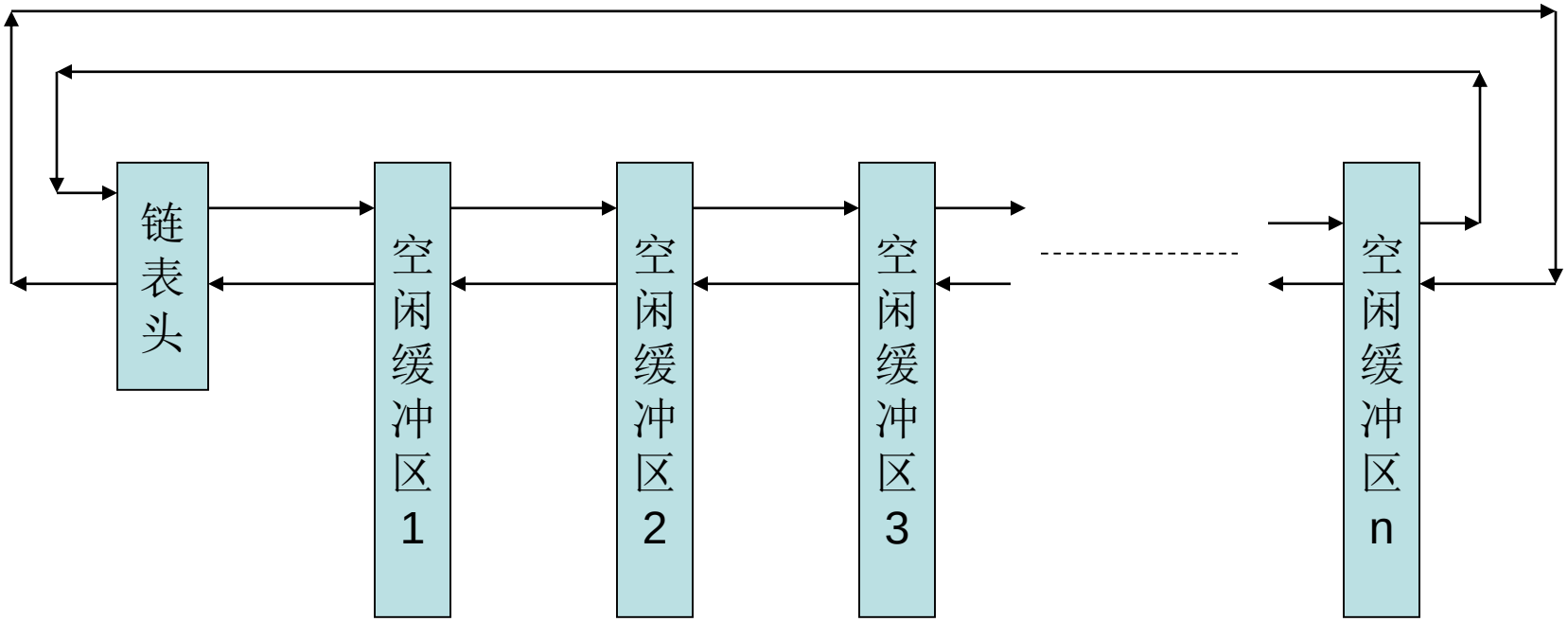
2、缓冲池设计基本原则：

① 存放有刚使用过的数据尽量长时间地保留在内存中，以便马上还要使用时能在内存中找到；

② 需要腾出内存空间时，把很久都未使用过（即最近最少使用）的数据交换到磁盘上去。这些数据马上还要使用的可能性最小。

3、空闲缓冲区链表

核心维护了一个空闲缓冲区链表，它按照最近被使用的先后次序排列。空闲链表是一个以空闲缓冲区链表头开始的“双向循环链表”。链表的开始和结束都以链表头为标志。



4、空闲缓冲区链表操作

① 取用任意空闲缓冲区

从空闲缓冲区链表的表头位置取下一个空闲缓冲区，后面的空闲缓冲区依次向前移动。

② 释放一个空闲缓冲区

把这个装有数据的空闲缓冲区附加到空闲链表的链尾。只有当该空闲缓冲区所装数据出错时才挂到链头。

③ 取用装有指定内容的空闲缓冲区

从链表头开始查找，找到后取下使用，用完后放到链尾。

当系统不断从链头取用空闲缓冲区，又把使用过的（装有数据的）缓冲区挂到链尾，一个装有有效数据的缓冲区就会逐渐向链表头移动。在链表头位置的就是“最近最少使用”的空闲缓冲区。

5、空闲缓冲区分类

系统中共设置了四个空闲缓冲区链表，根据缓冲区的不同用途而把它的放入不同的空闲缓冲区链表中。避免在取用空闲缓冲区时，逐个判断缓冲区中的内容。这四个空闲链表是：

0#空闲缓冲区链表——存放文件系统超级块
1#空闲缓冲区链表——存放通常使用的数据块
2#空闲缓冲区链表——存放延迟写、无效数据或错误内容
3#空闲缓冲区链表——存放没有对应存储空间的缓冲区首部

如果某种类型的空闲缓冲区不够用时，核心也从其它空闲缓冲区链表中取用空闲缓冲区。

6、缓冲区设置

当核心需有一个空闲缓冲区时，它根据要装入的数据类型，从相应的空闲缓冲区链表的表头位置取下一个空闲缓冲区，装入一个磁盘数据块；

根据该数据块所对应的设备号和块号数据对计算其 **hashno**（散列、杂凑）值，根据其 **hashno** 的值放入到相应 **hash** 链表的链头。

hashno = ((diskno + blkno) / RND) % BUFHSZ

diskno： 设备号
blkno： 块号
BUFHSZ： 最大**hash**值，通常为**63**。
RND： 随机数，其值为：**RND=MAXBSIZE / DEV_BSIZE**
MAXBSIZE： 最大文件系统块的大小
DEV_BSIZE： 物理设备块大小

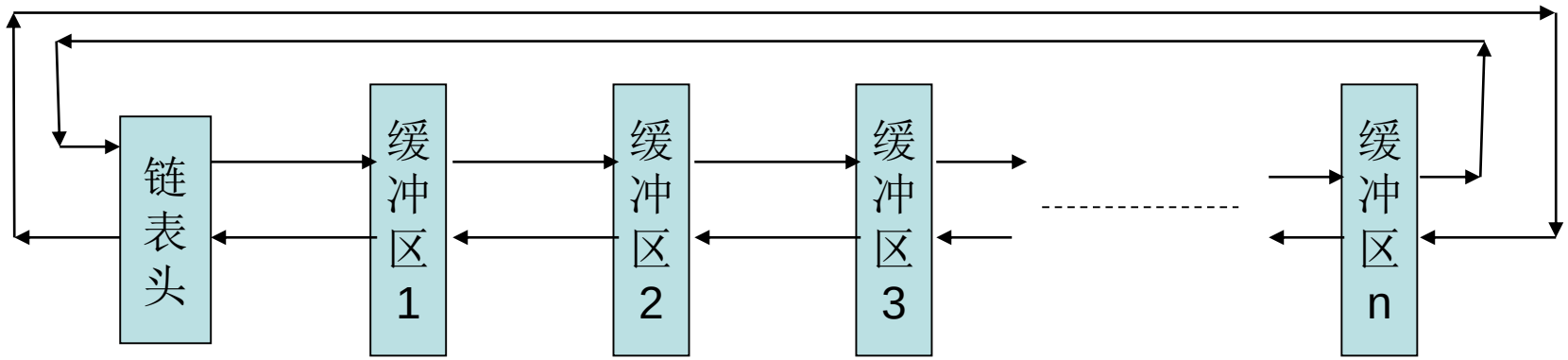
hashno 的取值范围： 0 ~ 62

7、缓冲池的结构

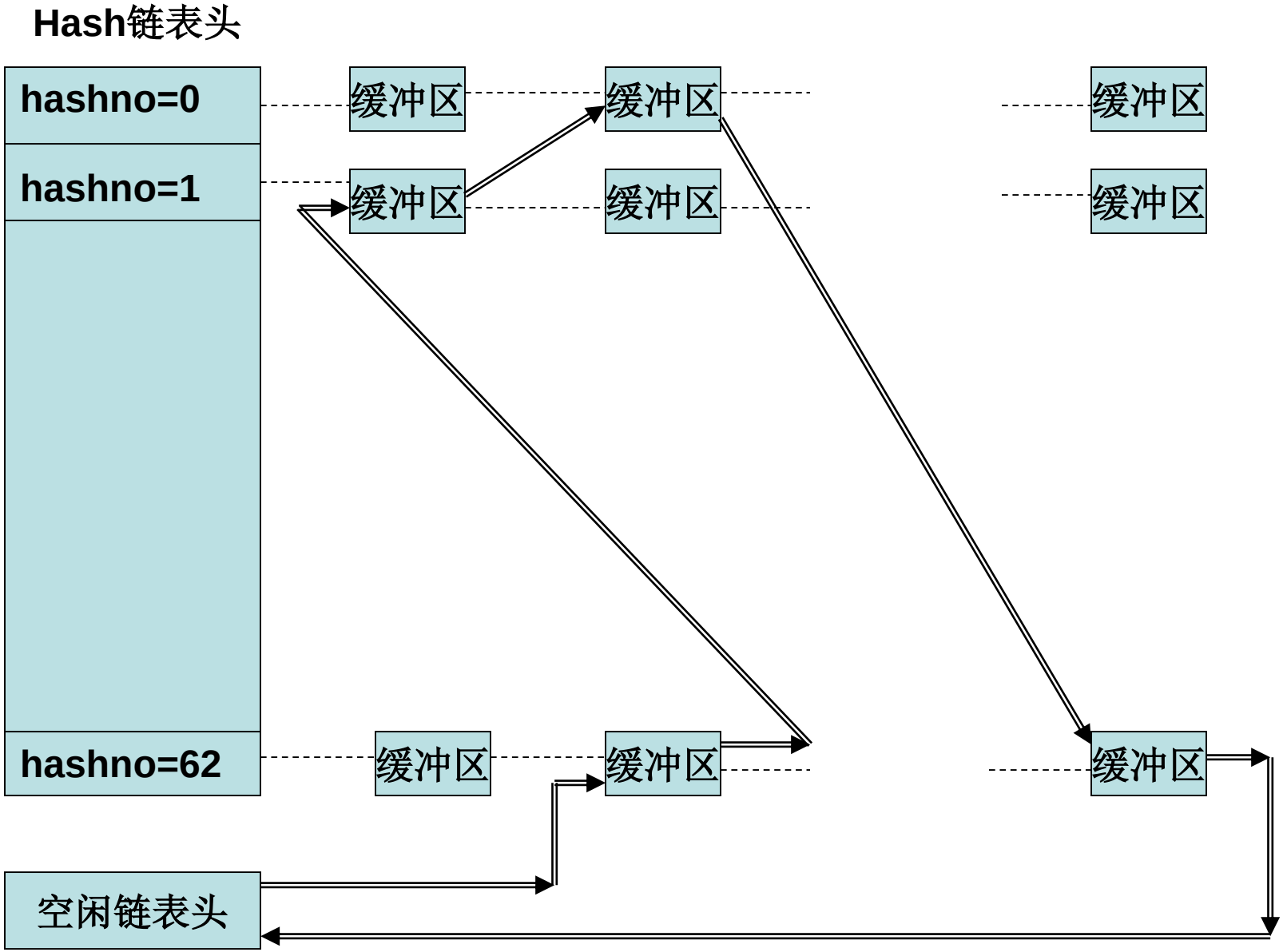
具有相同 **hashno** 的缓冲区链接在同一个**hash**链表中，因此系统中共有 **63** 个**hash** 链表，分别链接 **hashno** 为 **0 ~ 62** 的缓冲区。

每一个 **hash** 链表都是一个由链表头指向的双向循环链表，查找某一个指定 **hashno** 值的缓冲区时，也是从相应的**hash**链表的表头位置开始向表尾方向进行查找。

这 **63** 个 **hash** 链表就构成了数据缓冲区高速缓冲的缓冲池，所有的缓冲区都存放在缓冲池中的某一个链表中。



hash 链表的结构


Hash链表头
hashno=0
hashno=1
hashno=62
缓冲区
缓冲区
缓冲区
缓冲区
缓冲区
缓冲区
缓冲区
缓冲区
缓冲区
空闲链表头

缓冲池的结构

8、缓冲区的使用

如果要找特定缓冲区，根据**hashno**从相应的**hash**链表的表头处开始逐个向后查找；

如果找到，则直接取用，并将其移动到**hash**链的链头；

如果未找到，则从相应的空闲缓冲区链表的表头处取下一个空闲缓冲区，填入相应数据，重新计算其**hashno**，并放到新的**hash**链表的表头；

释放缓冲区时，将该缓冲区仍保留在原**hash**队列中，同时挂接到空闲缓冲区链表的表尾。（同时在两个队列中）

申请缓冲区的两个途径：

要指定缓冲区 —— 在**hash**链表中查找

要空闲缓冲区 —— 在空闲链表中查找

9、进一步说明

一个缓冲区只有当它是空闲状态时，它才同时处在hash链表和空闲链表中。如果不空闲，则它只能处在某一个hash链表中。

在空闲缓冲区链表中的缓冲区一定在某个hash链表中；在hash链表中的缓冲区不一定在空闲链中。不存在脱离hash链表的另一个空闲的缓冲区链表。

缓冲池中的缓冲区个数是固定不变的，每个缓冲区在不同时刻存放着不同的磁盘数据块，具有不同的hash值，因此处在不同的hash链表中。

缓冲区中的数据与某个磁盘数据块一一对应，这种对应有两个特点：

① 一个磁盘数据块在缓冲池中最多只能有一个副本；

② 缓冲区与数据块的对应是动态的，LRU数据块将被释放。

3.3 缓冲区的检索算法

在UNIX文件系统中的下层，即直接与逻辑存储设备联系的部分，包含如下基本算法：

getblk	申请一个缓冲区
brelse	释放一个缓冲区
bread	读一个磁盘块
breada	读一个磁盘块，预读另一个磁盘块
bwrite	写磁盘块

1、申请一个缓冲区算法 getblk

根据缓冲池的结构，核心申请一个缓冲区分配个磁盘块时，可能出现的五种典型状况：

① 该块已在hash队列中，并且缓冲区是空闲的；

② hash队列中找不到该块，需从空闲链表中分配一个缓冲区；

③ hash队列中找不到该块，在从空闲链表中分配一个缓冲区时，发现该空闲缓冲区标记有“延迟写”，核心必须写出缓冲区内容到磁盘上，再重新分配一个空闲缓冲区；

④ hash队列中找不到该块，并且空闲链表已空；

⑤ 该块已在hash队列中，但该缓冲区目前状态为“忙”。

```
算法  getblk
输入:文件系统号
     块号
输出:现在能被磁盘块使用的上了锁的缓冲区
{
    while  (没找到缓冲区)
    {
        if  (块在散列队列中)
        {
            if  (块忙)     /*   第五种情况   */
            {
                sleep(等候"缓冲区变为空闲"事件);
                continue;     /*   回到 while 循环   */
            }
            为缓冲区标记上"忙";     /*   第一种情况   */
            从空闲表上摘下缓冲区;
            return(缓冲区);
        }
        else     /*   块不在散列队列中   */
        {
            if(空闲表上无缓冲区)     /*   第四种情况   */
            {
                sleep(等候"任何缓冲区变为空闲"事件);
                continue;     /*   回到 while 循环   */
            }
            从空闲表上摘下缓冲区;
            if  (缓冲区标记着延迟写){     /*   第三种情况   */
                把缓冲区异步写到磁盘上;
                continue;     /*   回到 while 循环   */
            }
            /*   第二种情况——找到一个空闲缓冲区   */
            从旧散列队列中摘下缓冲区;
            把缓冲区投入新散列队列;
            return(缓冲区);
        }
    }
}
```

2、 释放一个缓冲区算法 brelse

- 唤醒等待缓冲区的所有进程
- 提高处理机的执行级别以封锁同级或低级的中断
- 将该缓冲区放到空闲队列的尾部（缓冲区有效）或头部（缓冲区无效）
- 降低处理机的执行级别以开放中断

算法　brelse

输入：上锁状态的缓冲区

输出：无

{

唤醒正在等待“无论哪个缓冲区变为空闲”这一事件的所有进程；

唤醒正在等待“这个缓冲区变为空闲”这一事件的所有进程；

提高处理机执行级别以封锁中断；

if （缓冲区内容有效且缓冲区非“旧”）

将缓冲区送入空闲链表尾部；

else

将缓冲区送入空闲链表头部；

降低处理机执行级别以允许中断；

给缓冲区解锁；

}

3、读一个磁盘块 bread

- 由 **getblk** 算法申请一个可用的缓冲区
- 如果缓冲区中的内容有效，则直接返回该缓冲区
- 如果缓冲区中的内容无效，则启动磁盘去读所需的数据块
- 等待磁盘操作完成后返回

算法 **bread**

输入：文件系统号

输出：含有数据的缓冲区

{

得到该块的缓冲区（算法**getblk**）；

if（缓冲区数据有效）

return（缓冲区）；

启动磁盘读；

sleep（等待“读盘完成”事件）；

return（缓冲区）；

}

4、 读一个磁盘块并预读另一个磁盘块 breada

预读的前提：

程序是在一个有限的空间内运行，程序对数据的访问是可预见的。

预读的命中率：

不一定达到100%，但良好的系统结构和算法可使命中率达到较高的水平。

预读的结果：

放在缓冲池内，以免需要的时候再去启动磁盘读数据块。

```
算法 breada
输入：（1）立即读的文件系统块号
      （2）异步读的文件系统块号
输出：含有立即读的数据的缓冲区
{
        if（第一块不在高速缓冲中）
        {
                为第一块获得缓冲区（getblk）；
                if（缓冲区内容无效）
                        启动磁盘读；
        }
        if（第二块不在高速缓冲中）
        {
                为第二块获得缓冲区（getblk）；
                if（缓冲区数据有效）
                        释放缓冲区（brelse）；
                else
                        启动磁盘读；
        }
        if（第一块本来就在高速缓冲中）
        {
                读第一块（bread）；
                return（缓冲区）；
        }
        sleep（第一个缓冲区包含有效数据的事件）；
        return（缓冲区）；
}
```

5、写磁盘块 bwrite

启动磁盘驱动程序的写操作

如果是“同步写”，则本进程睡眠等待磁盘写操作的完成，磁盘写操作完成后，中断唤醒本进程，本进程释放该缓冲区并返回；

如果是“异步写”，则无需等待磁盘写操作的完成，将缓冲区放到空闲链表的表头，以便随后某个进程申请空闲缓冲区时，将其写到磁盘上去。本进程不再关心该缓冲区实际被写出的时间和结果，而直接返回去作其它事情。

事实上无论是同步写还是异步写，其根本区别在于本进程是否等待磁盘驱动程序完成操作后所发出的中断信号。

算法 **bwrite**

输入：缓冲区

输出：无

```
{
        启动磁盘写;
        if（I/O同步）
        {
                sleep（等待“I/O完成”事件）；
                释放缓冲区（brelse）；
        }
        else  if （缓冲区标记着延迟写）
                为缓冲区做标记以放到空闲缓冲区链表头部;
}
```

3.3 数据缓冲区高速缓冲的优缺点

优点：

- ✓ 提供了对磁盘块的统一的存取方法
- ✓ 消除了用户对用户缓冲区中数据的特殊对齐需要
- ✓ 减少了磁盘访问的次数，提高了系统的整体I/O效率
- ✓ 有助于保持文件系统的完整性

缺点：

- ✓ 数据未及时写盘而带来的风险
- ✓ 额外的数据拷贝过程，大量数据传输时影响性能