第四章 高速缓存管理

4.1 高速缓存概述

通常文件 I/O 操作使用高速缓存来作为用户地址空间和设备之间的中间存储区。例如，写一个文件时，数据实际上是写到高速缓存中去。操作系统周期的清理高速缓存的内容，把它放入磁盘。每一个高速缓存块都有一个缓冲块头部，它保存着这个缓冲块的控制信息，例如该块所属的文件、物理块号等。

使用高速缓存的优点如下：

- 高速缓冲数据 数据尽可能长时间地留在主存当中，使用户在存取若干次同样的数据时，直接存取内存即可，不必每次进行物理 I/O 操作，因而减少了用户进程被阻塞（等待物理 I/O）的时间。
- 一致性 当多个用户进程进行 I/O 操作时，操作系统使用同一个高速缓存。这样就只有一个观察点来观察文件的内容，任何进程在存取文件时不必为定时担心。使用数据缓冲区的算法也有助于保证文件系统的完整性，因为他们处理的是高速缓存中共同的、单一的磁盘块映像。如果两个进程试图同时操作一个磁盘块，缓冲算法就使它们串行工作，这就避免了数据的崩溃。
- 可移植性 由于 I/O 操作通过高速缓存来进行，用户进程就不必操心数据对齐等与硬件设备密切相关的问题，从而提高代码的可移植性。
- 提高系统性能 使用高速缓存可以减少磁盘的存取次数，提高系统的通过能力并降低平均响应时间。特别时在作少量数据传输时，数据留存在高速缓存中，直到系统认为适当的时候才进行数据传送，这样就提高了系统性能。

4.2 高速缓存管理的基本思想

高速缓存管理是文件系统中的一个重要部分，好的高速缓存管理算法能够大幅度提高文件系统的读写文件效率。在高速缓存中，我们采用了 NIX 类系统普遍采用的 LRU 算法，该算法是基于这样的想法：在以前被频繁用到的块有可能在以后也频繁用到；反过来说，已经很久没有使用的块可能在未来较长一段时间

内也不会使用。其实在程序设计中，采用这种思想的地方很多，比如数据压缩，只是它用了不同的表达方式：在以前没有出现的数据序列在以后较长一段时间内也不会出现。

很多内存管理都涉及到该算法，但是它们的实现代价较高，主要是因为每次访问内存时都要去修改 LRU 链表，这是一个很耗费系统资源的工作。但是在高速缓存管理中不存在这种问题，我们只不过需要在每次访问缓冲区时作这项工作。它的主要思想如下图所示：

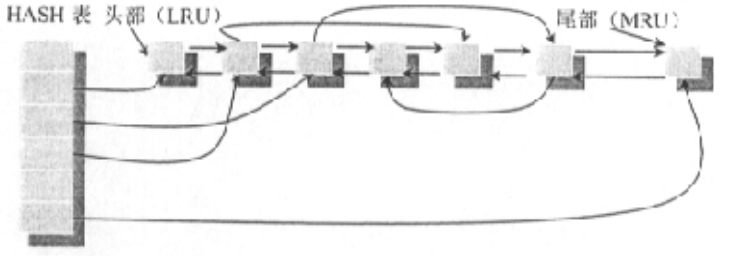

图 4-1 块高速缓存

高速缓存采用一个数组来实现。其中每个缓冲区由包含指针、计数器和标志的头以及用于存放磁盘扇区的体组成。所有未使用的缓冲区均使用双向链表，按最近一次使用时间从近到远的顺序排列起来。

为了迅速判断某一块是否在高速缓存中，我们使用了哈希（hash）表。所有缓冲区，如果它包含的块的哈希代码为 k，则在哈希表中用第 k 项指向的单链表链接在一起。哈希表提取块号低 n 位作为哈希代码，因此不同设备的块可能出现在同一哈希链中。每个缓冲区都在其中某个链中。当系统启动的时候，所有缓冲区均未使用，并且全部在哈西表的第 0 项指向的单链表中，这时，哈西表的其它项均为空指针。一旦有缓冲区的使用请求被提出时，缓冲区将从 0 号链中删除，放到其它链中。

高速缓存管理的另一重要作用是对文件系统屏蔽具体硬件接口，使得文件系统都可以用如下调用读写所有设备：

```
buf_ptr=get_buffer(dev,sector_num);
```

当文件系统需要一块时，它就会调用过程 get_buffer，计算该块的 hash 代码，搜索相应链。get_buffer 被调用时，有两个参数：设备号和扇区号，只要

给出设备号和所要得到的块在设备上的相对扇区号即可得到不同硬件设备上的指定扇区/块。这两个值与缓冲区链中对应域相比较，如果找到包含这一块的缓冲区，这缓冲区头中标志块使用次数的计数器加 1，并返回指向该缓冲区的指针。如果在 hash 表中未找到这样的块，我们可以使用 LRU 链中的第一个缓冲区。由于 LRU 链中的缓冲区必然不在使用中，因而它包含的块可以被换出内存，以释放这个缓冲区。

如果选定的某一块要调出内存，这是需要检查块头中另一个标志，看它在上次读入内存之后是否修改过。若修改过，就重新写回硬盘，然后读入新块。这时请求该块的进程被挂起，直到块被读入之后才重新运行。

最近不可能用到的块，比如引导纪录，放在 LRU 的头部。这样，如果下次需要一个空缓冲区，就可以立即使用。所有其它的块都按真正的 LRU 方式放在 LRU 链的尾部。

修改的块只有在以下两种情况写回磁盘：

（1） 到达 LRU 的头部并被换出。

（2） 执行了 SYNC 系统调用。SYNC 并不是通过 LRU 查找缓冲区，只要缓冲区在内存中并被修改过，它都将修改缓冲区在磁盘上的副本。

但是有一种情况很特殊，就是有一些重要数据在修改后要立即写回磁盘，比如引导记录和 FAT 表，这样可以减少系统在崩溃时文件系统被破坏的可能性。

到这里我们可以给出的结论是：整个高速缓存管理只关心对链表的操作，而不知道文件系统那个过程需要该块，以及为什么需要该块。这样做有利于程序的层次化和模块化。下一小节中，我们详细介绍了高速缓存管理的有关数据结构和实现方法。

4.3 主要数据结构和算法

4.3.1 高速缓冲区数据结构

高速缓冲区要解决的问题是与设备的对应关系，并且要存储物理设备上一个扇区的数据，配合各种管理算法的实现。综合这些因素，它的数据结构定义如下：

struct b_buf