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

```
{
    union
    {
    unsigned char data_block[BLOCK_SIZE];      /*缓冲区存放的数据块*/
    DIR_ENTRY dir_block[DIR_NUM];              /*缓冲区存放的目录项*/
    }b;

    long b_num;                              /*缓冲区对应的块号*/
    unsigned char b_dev;                        /*块所在的设备 */
    unsigned char b_attrib;                     /*块的属性 */
    struct b_buf *b_prev;                    /*前指针*/
    struct b_buf *b_next;                    /*后指针*/
    struct b_buf *b_hash;                    /*指向 hash 表的指针*/
    unsigned char b_dirty;                      /*块是否修改过标记*/
    unsigned char b_inuse;                      /*块使用次数的计数器*/
    BYTE b_copies;                           /*拷贝个数*/
    BYTE b_offset;                           /*拷贝之间间隔的扇区数 */
}b_buf[NR_BUFS];
```

结构的第一部分是一个用联合体表示的数据区，同一个扇区中存储的数据可能是一般的数据，也可能是目录。实际上后者是前者的一种，这里这样定义该结构主要是便于对目录操作的时候节省一些计算步骤。但是这样也有一个问题，就是定义的 DIR_ENTRY 这个数据结构是从 MS-DOS 的 FAT 文件系统中来的，这样使该缓冲区的数据结构不能不加修改用于别的文件系统。

接下来的部分是缓冲区的控制区，首先是块所在设备的设备号和缓冲区对应的块号，这两个变量一起标识了特定设备上的特定块；块的属性是与文件系统相关的特性，它标识了存储在数据区中的块的种类，可以是数据、目录、FAT 表、超级块等，便于程序采用不同的方法处理；指针域中包含三个变量：双向链表的前后指针以及指向 hash 链的后项指针。它们的作用在图 4-1 中已经清楚的表示了出来；块是否修改过标记是用于是释放该块的时候检查该块是否需要写回设备

的依据；块使用次数的计数器记录该缓冲区的使用次数。

以下的两个域：拷贝个数和拷贝之间间隔的扇区数是专门针对 FAT 文件系统定义的，拷贝个数域的取值为 1 或 2，因为在 FAT 文件系统中很多重要的数据，象引导记录和 FAT 表，都有它的备份，但最多有一个备份。拷贝之间间隔的扇区数就是这些备份之间的距离，这样便于在这些数据修改后及时的更新所有的备份。

对于缓冲区的个数 NR_BUFS，可以用户自己定义，也可以使用系统的缺省定义，在 80386 以下的 CPU 上是 40 个，80486 以上是 512 个。虽然缓冲区越多文件系统的性能会越高，但是考虑到嵌入是系统的特殊性，我们建议用户根据自己的系统定义缓冲区大小。

4.3.2 高速缓存管理的主要算法描述

高速缓存管理模块中主要的函数有：get_buffer（得到一个缓冲区）、release_buffer（释放一个缓冲区）、flushall（支持 SYNC 系统调用，刷新所有缓冲区）、rm_lru（从 LRU 链中获取一个缓冲区）以及 init_buf（初始化整个缓冲区）。以下是这些函数的详细算法描述：

4.3.2.1 get_buffer

算法：get_buffer
功能：得到一个缓冲区
输入参数：设备号 dev
　　块号 block
输出参数：指向空闲缓冲区的指针 B_BUF *bp
算法描述：
{
计算该块在 hash 表中的位置；
将在 hash 表中找到的该块赋给 bp;
while (没有到该 hash 链的结尾)
{