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