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)
{

```
    if(设备号与块号均与参数吻合)
    {   /*所需要的块正好在缓冲区中*/
        if(该块当前未被使用)rm_lru（bp）;/*从空闲表上摘下缓冲块*/
        该缓冲块使用计数器加 1;
        return(bp);
    }
    else     /*块不在 hash 表中*/
    获得该 hash 链上的下一缓冲区;
}

/*在 hash 表中没有找到可用的缓冲区*/
rm_lru(bp);  /*从 LRU 链上摘下 bp 所指向的缓冲块*/
将该块从以前所在的 hash 链上释放出来;

if (该块被修改过)flushall（dev）; /*刷新该设备使用的所有块，将修改过的块回写*/

填充该缓冲区块的各个参数，包括块号、设备号、使用计数;
将该块链接进 hash 链中;
rw_block（bp,opcode = DFSFREADS）;  /*读写磁盘块，后面的 DFSFREADS 表示读设备*/
return(bp);
}
```

4.3.2.2 release_buffer

算法：release_buffer

功能：释放缓冲区

输入参数：待释放的缓冲区指针 bp

输出参数：无

算法描述：

{

if（块为空）/*已经被释放了或者根本就没有使用*/

　　returen；/*将检查放在这里要比放在调用它的函数中实现起来简单*/

/*块仍然在使用中*/

块的使用次数计数器减 1；

if (块的使用次数不为 0)

　　return;　/* 块仍在使用中*/

/*真正释放该块*/

将该块加入到 LRU 头部；

}

4.3.2.3 flushall

算法：flushall

功能：刷新设备中所有修改过的块

输入参数：待刷新的设备 dev

输出参数：无

算法描述：

{

　　初始化当前使用的设备；；

　　for(缓冲区中的所有缓冲块)

　　{

　　　　if (使用该块的设备为待刷新设备 and 缓冲区被修改过)

　　　　{

　　　　　　rw_block(bp,opcode = DFSFWRITES); /*向磁盘写该缓冲块*/

　　　　　　if (该缓冲块在设备上有多个拷贝)

　　　　　　　　按该缓冲块的拷贝个数重复写入磁盘不同区域;

　　　　}

　　清除缓冲块修改位；

}

}

4.3.2.4 rm_lru

算法：rm_lru

功能：从 LRU 链中摘下一个缓冲区

输入参数：指向待摘下缓冲区的指针

输出参数：无

算法描述：

{

缓冲区使用计数器加 1；

记录该缓冲区的前驱和后继；

if (前驱不为空)置前驱的后继指针指向后继；

else/*前驱为空*/

将后继缓冲区置于 LRU 的链头；

if (后继不为空)置后继的前驱指针指向前驱；

else/*后继为空*/

将前驱缓冲区置于 LRU 的链头；

}

4.2.2.5 init_buf

算法：init_buf

功能：初始化高速缓冲区

输入参数：无

输出参数：无

算法描述：

{

```
    置双向链表头指针指向缓冲区数组的第一个元素;
    置双向链表尾指针指向缓冲区数组的最后一个元素;

    for (bp = 第一个缓冲区; bp < 最后一个缓冲区;bp++)
    {
    /*对整个数组从头至尾的初始化*/
        置块号为空;
        置设备号为空;
        后继指针指向数组中下一个元素;
        前驱指针指向前一个数组元素;
        缓冲区状态为“未修改”;
        缓冲区使用计数为 0;

       hash 指针指向后继;
        }
    链表头的前驱为空;
    链表为的后继为空;
    hash 表第 0 项指向双向链表头;
}
```