Linux 文件系统数据缓冲区的分析研究

夏　煜[1]　郎荣玲[2]　戴冠中[1]
[1] 西北工业大学自动控制系，西安 710072）
[2] 西北工业大学应用数学系，西安 710072）
E-mail：yuxianwpu@263.net

摘　要　文章深入研究了 Linux 文件系统的数据缓冲区管理，包括数据缓冲区的整体结构、数据缓冲区采用的数据结构和实现方法。

关键词　数据缓冲区　哈希链表　LRU 链表

文章编号 1002-8331-（2001）17-0126-03　**文献标识码** A　**中图分类号** TP316

Research on Buffer Cache in Linux File System

Xia Yu[1]　Lang Rongling[2]　Dai Guanzhong[1]
[1] （Department of Automatic Control，Northwestern Polytechnical University，Xi´an 710072）
[2] （Department of Mathematics，Northwestern Polytechnical University，Xi´an 710072）

Abstract：The paper studies the managerment of buffer cache in Linux file system in detail，including the whole structure of buffer cache，its data structures and realizing methods.

Keywords：buffer cache，hash list，LRU list

1　引言

在操作系统中，当用户进程访问文件系统时，文件系统对外部块设备提出大量的 I/O 请求。由于外部设备速度相对较慢，如果文件系统的所有请求都直接由外部设备直接响应，会产生对外部设备频繁的 I/O 操作，从而降低系统性能，甚至产生一种振荡的现象，使外部设备的 I/O 操作无法进行。为了解决上述问题，Linux 的文件系统采用数据缓冲区的方法。数据缓冲区是一部分内存空间，文件系统将数据块从外部设备读入缓冲区中，并通过一定的方法将数据块有效地组织起来，以便于数据块的查找、更新等操作。通过这种方法，用户进程对数据块的操作就可以在缓冲区中进行了。在完成操作后，如果该数据块不再使用或不经常使用，便可将数据块写回外部设备，同时释放占用的内存空间。

文章研究的 Linux 操作系统内核源程序版本为 2.2.5-15。

2　数据结构

为了方便文件系统的访问，外部设备的数据空间被分成相同大小的数据块，外部设备的数据空间的分配就以数据块为单位。可以说，数据块是块设备的最小分配单位。在 Linux 文件系统中定义了多种数据块大小，如：1K、2K、4K 等，这种机制可以使文件系统支持不同块设备和不同的具体文件系统。

数据缓冲区利用缓冲区头数据结构来管理内存中的数据块，每一个内存中的数据块都用一个缓冲区头来表示。通过缓冲区头数据结构，内存中的数据块用合理的方法组织起来，以有利于数据块的查找、更新和数据缓冲区的分配、回收。

缓冲区头数据结构为：

```
struct buffer_head{
    ...
    struct buffer_head *b_next;
    unsigned long b_blocknr;
    unsigned long b_size;
    kdev_t b_dev;
    kdev_t b_rdev;
    unsigned long b_state;
    struct buffer_head *b_next_free;
    char *b_data;
    unsigned int b_list;
    struct buffer_head **b_pprev;
    struct buffer_head *b_prev_free;
    ...
};
```

该数据结构的主要属性有：

b_next，b_pprev 表示缓冲区头在哈希表中的直接后继和直接前驱的指针。

b_next_free，b_prev_free 表示缓冲区头在空闲链表或 LRU 链表中的直接后继和直接前驱指针。

b_dev，b_rdev 表示所代表的数据块的设备名。

b_blocknr 表示数据块在块设备上的数据块号。

这两个属性可唯一地表示和标识所有外部设备的任何一个数据块。

b_size 表示所代表的数据块的大小。

基金项目：西北工业大学“双新计划”基金资助

作者简介：夏煜（975-），男，博士生，主要研究方向为计算机控制和智能控制、计算机操作系统、计算机网络与安全。郎荣玲（975-），女，硕士生，主要研究方向图论、计算机密码学、网络安全。戴冠中（937-），男，教授，博士生导师，主要研究方向计算机控制和智能控制、控制系统中的并行处理和并行计算机。

b_state 表示缓冲区头的状态。该状态可为 BH_Uptodate（最新）、BH_Dirty（脏）、BH_Lock（锁定）、BH_Protected（保护）等。其中，最新表示该数据块缓冲区为新，未修改；脏表示该数据块缓冲区修改后还未写回；锁定表示数据块缓冲区等待被写入。

b_data 表示该缓冲区头的数据块指针。

b_list 表示该缓冲区所在的子链表类型。分别有三个子链：未使用的缓冲区链表（clean）、脏的缓冲区链表（dirty）、锁定的缓冲区链表（locked）。

3 数据缓冲区的体系结构

评价一个数据缓冲区结构的优劣依据两个标准：一，数据缓冲区的组织要易于查找。只有实现了内存中的数据缓冲区快速查找和定位，才能充分发挥数据缓冲区的缓冲功能。二，数据缓冲区要很好地解决数据缓冲区重用的问题。随着时间的推移，越来越多的数据缓冲区被占用，但是内存中的数据缓冲区数目是一定的，最终可能会出现数据缓冲区不足的现象。这时，就要采用一定的方法将长时间未被使用的数据缓冲区释放，为其它数据块提供空闲数据缓冲区。而经常被访问的数据块保存在内存中，以继续使用。

在 Linux 文件系统中，数据缓冲区分为两个功能模块：空闲块数据缓冲区和数据块缓冲区。对于任何一个数据块缓冲区元素，要么在空闲数据缓冲区中，通过空闲数据缓冲区链表链接起来；要么在数据块缓冲区中，通过哈希链表实现数据缓冲区的快速查找，通过 LRU 链表实现数据缓冲区高效重用的问题。

3.1 空闲块数据缓冲区

在文件系统初始化时，数据缓冲区未被使用，它们都链接在空闲块数据缓冲区链表中。随着数据缓冲区的使用，该链表中的缓冲区逐渐减少，它们分别被占用后，插入到数据块缓冲区中。当操作系统执行更新操作后，数据块缓冲区中释放的缓冲区又被链接到空闲块数据缓冲区链表中，以备为新的数据块进行分配。

Linux 的空闲块数据缓冲区是由多个双向循环链表组成的，每种文件系统支持的数据块大小形成一个链表。这些链表都使用缓冲区头数据结构的 b_free_next 和 b_free_prev 指针链接。空闲数据块缓冲区的链表定义为：

```
#define NR_SIZES 7
static struct buffer_head *free_list[NR_SIZES]={NULL,};
```

NR_SIZES 表示空闲数据块缓冲区支持多少种块大小，这也决定了空闲块数据缓冲区的链表数目。指针数组 freed_list 的每一个元素指向了一个同样数据块大小的空闲数据缓冲区链表。

可通过下面的算法实现根据数据块大小决定空闲数据块所在的链表，即实现定位 free_list 的数组下标。该算法为：

```
static char buffersize_index[65] ={
-1,0,1,-1,2,-1,-1,-1,3,-1,-1,-1,-1,-1,-1,-1,
4,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,
5,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,
-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,6
};
#define BUFSIZE_INDEX(X) ((int)buffersize_index[(X)>>9])
```

该算法采用了一个数组保存相应块大小对应的链表号。对于大小为 X 的数据块，利用 BUFFER_INDEX(X) 的值作为 free_list 指针数组的下标，因此可支持 7 种数据块大小。这 7 种数据块大小分别为 512、1024、2048、4096、8192、16384、32768 字节，它们各自链接成一个链表。其中对应的 -1 值，表示该数据块大小是非法的。

3.2 数据块缓冲区

当用户进程需要外部设备的某个数据块时，它将该数据块存放到数据块缓冲区中。Linux 文件系统使用两个静态全局变量表示整个缓冲区中的缓冲区头数目和缓冲区数目，定义为：

```
static int nr_buffers=0;
static int nr_buffer_heads=0;
```

Linux 的数据块缓冲区是由两种链表共同实现的：哈希链表和 LRU 链表，它们分别解决了数据块的快速定位和缓冲区重用问题。

(1)哈希链表

哈希链表是为了实现缓冲区中数据块的快速查找而形成的数据结构。哈希链表利用了缓冲区头的两个属性链接成链表，它们是 b_pprev 表示直接前驱的一重间接指针，b_next 表示直接后继的指针。在 Linux 文件系统中哈希链表的定义为：

```
static struct buffer_head **hash_table;
static int nr_hashed_buffers=0;
```

二重指针 hash_table 指向了哈希链表，nr_hashed_buffers 表示了在哈希链表中的缓冲区数目。

哈希链表所采用的哈希算法为：

```
#define _hashfn(dev,block) (((unsigned)(HASHDEV(dev)^block))& bh_hash_mask)
```

相应的哈希函数 hash() 为：

```
#define hash(dev,block)hash_table[_hashfn(dev,block)]
```

相关的常量和变量定义为：

```
order=5;
nr_hash = (UL <<order)*PAGE_SIZE/sizeof(struct buffer_head*);
bh_hash_mask=nr_hash-1;
#define HASHDEV(dev) ((unsigned int) (dev))
#define PAGE_SHIFT 12
#define PAGE_SIZE (UL<<PAGE_SHIFT)
```

在哈希算法中，首先将该数据块所在设备号和所在块号进行异或，再与哈希掩模值相与，从而得到了该数据块在哈希链表中的键值。哈希掩模值是根据缓冲区头大小、内存页大小、相关系数计算出来的，它决定了哈希链表中键值的数目。

下面分析一下一个数据块缓冲区的查找过程。首先根据数据块所在的设备号 dev 和数据块号 block，通过哈希算法获取哈希键值，根据键值得到哈希链表相应条目的入口指针；然后通过匹配设备号和数据块号的方法依次查找该链表上的每一个数据块缓冲区，直到找到该数据块缓冲区，这个数据块在整个文件系统中是唯一的。数据块缓冲区的插入和删除过程与查找过程相类似。

(2)LRU 链表

LRU 链表，即最近最少使用链表，实现了经常使用的数据块缓冲区尽量长时间地保留在缓冲区中，很少使用的数据块缓冲区在有新的数据块请求且没有空闲缓冲区时将被重用，将缓冲区分配给新的数据块的机制。这种方法的实现大大地减少了

文件系统对外部块设备的读写操作,提高了数据缓冲区的效率。

LRU 链表根据数据块缓冲区使用的频度链接成链表,不经常使用的数据块缓冲区链接到链表的头部,经常使用的数据块缓冲区链接到链表的尾部。文件系统将 LRU 链表的头部数据块缓冲区选择重用,而 LRU 尾部的数据块缓冲区尽量长时间地保存在缓冲区中。

LRU 链表也是双向链表,它使用缓冲区头的两个属性链接成链表,它们是 b_prev_free 和 b_next_free,分别表示该数据块缓冲区的直接前驱和直接后继。Linux 文件系统根据缓冲区类型形成三个 LRU 链表,它们分别是:

未用的数据块缓冲区(clean):表示空闲的数据块缓冲区。

锁定的数据块缓冲区(locked):表示等待数据被写入。

脏的数据块缓冲区(Dirty):表示等待数据被写回设备。

在 Linux 文件系统内核中,LRU 链表定义为:

```
static struct buffer_head *lru_list[NR_LIST]={NULL,};
```

其中,lru_list 是一个指针数组,记录了三个链表的头指针。相关变量和常量定义为:

```
#define BUF_CLEAN 0
#define BUF_LOCKED 1
#define BUF_DIRTY 2
#define NR_LIST 3
```

NR_LIST 定义了 LRU 链表的数目,共有三个链表,依次为 BUF_CLEAN、BUF_LOCKED、BUF_DIRTY 链表。

对于每一个 LRU 链表,都定义了相应的变量记录各自链表中的数据块缓冲区数目,缓冲区数目变量的定义为:

```
static int nr_buffer_type[NR_LIST]={0,}
```

Linux 数据缓冲区的体系结构可由图 1 所示。

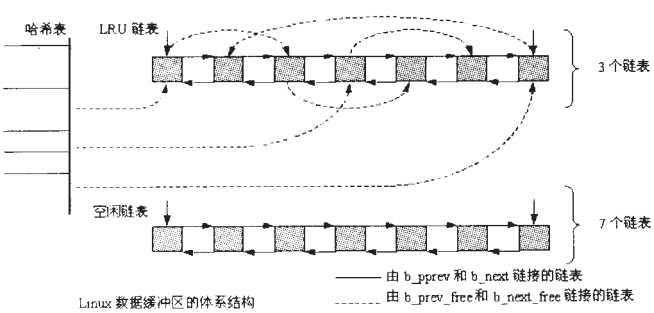

图 1

4 数据缓冲区的实现方法

4.1 数据缓冲区的删除方法

该方法将数据块缓冲区 bh 在数据块缓冲区链表中删除,包括三种链表:哈希链表、LRU 链表、空闲链表。

该方法的原理为:首先检查该数据块的设备号是否存在,如果存在,说明它位于数据块缓冲区,同时在哈希链表和 LRU 链表上删除;如果不存在,说明位于空闲数据缓冲区,在空闲链表上删除。

4.2 数据缓冲区的插入方法

该方法将数据缓冲区插入到适当的数据缓冲区链表中,包括三种链表:哈希链表、LRU 链表、空闲链表。

该方法的原理为:首先检查该数据块的设备号是否存在,如果存在,说明应插入到数据块缓冲区中,同时插入到哈希链表和 LRU 链表的适当位置;如果不存在,说明应插入空闲数据缓冲区,因此插入到空闲链表中。

4.3 获取数据块缓冲区方法

该方法是根据用户进程的请求,获取一个在数据块缓冲区中的数据块。

该方法的原理为:首先利用哈希表查找数据块缓冲区,是否已经存在于该数据块的缓冲区,如果存在则返回该数据块缓冲区的指针,如果不存在,根据数据块的大小确定使用哪一个空闲链表。将该空闲数据块缓冲区从空闲链表中删除,新建一个缓冲区头,调用数据缓冲区的插入方法将其插入到数据块缓冲区链表中。

4.4 释放数据块缓冲区方法

该方法实现缓冲区的释放。Linux 文件系统有两种释放方法:方法一,只是将该数据块缓冲区的使用数减 1。方法二,不仅该数据块缓冲区的使用数减 1,而且释放数据块缓冲区,并回收到空闲数据缓冲区中。

4.5 数据缓冲区的更新方法

在数据缓冲区使用过程中,每个数据块缓冲区的状态都会发生变化。这样在同一个链表上将会有不同状态的数据块缓冲区。该方法实现了相应的数据块缓冲区调整到适当的 LRU 链表上。

该方法原理为:首先,依次检查每个数据缓冲区是否存在设备号,如果不存在则返回,如果存在且状态与所在 LRU 链表不一致,则根据状态将其从当前 LRU 链表删除,插入到相应的 LRU 链表中去,同时修改 b_list 属性值。最后,根据脏缓冲数目激活更新程序,将数据块写回外部设备。

5 结束语

数据缓冲区管理是 Linux 文件系统的重要组成部分,它有着很多优秀的特性。文章主要研究了 Linux 文件系统的数据缓冲区的体系结构、数据结构和实现方法。也正是由于 Linux 操作系统自由、公开、免费的特性,提供给人们一个研究国外优秀操作系统的设计思想和实现方法的机会,对于发展国产操作系统有着很重要的意义。(收稿日期:2000 年 7 月)

参考文献

1.黄祥喜.计算机操作系统实验教程—MINIX 操作系统、分析和实现[M].中山大学出版社,1994

2.Andrews Tanenbaum.操作系统设计与实现[M].清华大学出版社,1997

3.夏煜.Linux 操作系统的文件系统研究[M].硕士学位论文.西安:西北工业大学,2000

4.Michael K Johnson.Linux Kernel Hackers´ Guide[M].1995

5.David A Rusling.The Linux Kernel[M].1996