第17章 缓存处理

在本章我们将详细查看文件“ bio.c ”。其中包含大多数处理缓存头和缓存的基本例程(4535、4720)。

各缓存头都包含设备号“ b_dev ”(4527)和块号“ b_blkno ”(4531)(注意，后者被说明为无符号整型)。

缓存头可同时连接在两个列表中：

• b列表：每个设备控制器1个，它将与该设备类型相关连的缓存连接在一起。

• av列表：该缓存列表中各缓存的当前用法都可改变成另一种用法。

av列表和各个b列表是双向连接的，这样便于在列表中任一点进行插入与删除操作。

17.1 标志

若一个缓存暂时从av列表中抽出(也就是不在av列表中)，则其“ B_BUSY ”标志被设置。

若一缓存的内容正确地反映了存储在或应存储在磁盘上的信息，则其“ B_DONE ”标志被设置。

若“ B_DELWRI ”标志设置，那么缓存中内容新于相应盘块中的内容，因此在重分配该缓存之前，一定要将其内容先写至相应盘块中。

17.2 一个类超高速缓存存储

我们将会看到在UNIX中对大型缓存的处理方法是模拟了对与计算机内存相关连的硬件高速缓存的操作，例如在PDP11/70中所实现的那样。

除了每一次都很短暂的时间外，并不将缓存分配给一特定程序或文件。于是，在大量文件和程序之间就可以有效地共享数量相当少的各个缓存。

在需要将某个缓存分配它用之前，信息一直保留在其中，并不舍弃，也就是说，如果最近只更改了缓存中的一部分内容，那么并不需要将这部分内容立即写到磁盘上。这样一来，读/写记录的长度小于缓存长度的程序就不会增加过多的开销。

最后，当进程终止以及文件被关闭时，保证将进程所占用缓存的内容正常地写到磁盘上的问题基本上被解决了，而这种问题在其他操作系统中往往会造成很多麻烦。

有一个需考虑的实际问题：如果由操作系统单独决定“ 什么时候实施写操作 ”，那么某些缓存的内容在相当长时间内可能都不会被写到磁盘上。为了解决这一问题， UNIX提供了一个公用程序，它每分钟运行2次，并强制将所有这些缓存的内容都无条件地写到磁盘上。这种措施使突然的系统崩溃所造成的损失显著减少。

17.3 clrbuf(5038)

本例程将指定缓存中的前256字(512字节)清0。注意，传送给“ clrbuf ”的参数是一缓存头。

“ clrbuf ”由alloc调用(6982)。

17.4 incore(4899)

此例程搜寻一个已分配给一特定(设备、块号)对的缓存。它循环搜索其列表头是该设备类型“ devtab ”结构的b列表。若在其中找到该缓存，则返回该缓存头的地址。“ incore ”由“ breada ”调用(4780、4788)。

17.5 getblk(4921)

此例程执行与“ incore ”相同的搜索，但是若这种操作未获成功，则从 av列表(可用缓存列表)中分配一个缓存。

调用“ notavail ”(4999)，可将该分配到的缓存从av列表中移出，并被标记为“ B_BUSY ”。

与“ incore ”相比，“ getblk ”对传送给它的参数要进行更多的检查。下列例程都调用“ getblk ”：

exec	(3040)	Writei	(6304)
exit	(3237)	iinit	(6928)
bread	(4758)	alloc	(6981)
breada	(4781、4789)	free	(7016)
smount	(6123)	update	(7216)

4940：到此点时，所要求的缓存已在搜索 b列表中找到。该缓存或者其“ B_BUSY ”标志已设置，在此情况下应调用“ sleep ”(4943)，或者是立即可用的(4948)。

4953：若在b列表中没有找到所要求的缓存，并且假定 av列表为空，则对 av列表设置“ B_WANTED ”标志，然后调用“ sleep ”(4955)。

4960：若av列表非空，则选择该列表中的第 1个成员，若其代表一“ 延迟写 ”缓存，则安排将其以异步写方式写到磁盘上(4962)。

4966：“ B_RELOC ”是在开发UNIX中遗留下来的，现已不再使用(4583)。

4967：从此到4973行的代码无条件地将该缓存从其原设备类型的 b列表中移出，然后插入到新设备类型的b列表中。因为这经常可能是“ no_op ”(“ 无操作 ”)，亦即，新、旧设备类型可能是相同的，所以在执行4967至4974行之前，插入下列测试语句似乎是合适的：

```
if(bp->b_dev==dev)
```

注意：若调用“ getblk ”时，dev参数值为“ NODEV ”(-1)，则“ getblk ”对此有特殊处理(这种调用有时没有提供第2个参数。见3040行)。

“ NODEY ”b列表的“ devtab ”结构使用的是“ bfreelist ”。

17.6 brelse(4869)

此过程将调用参数所指示的缓存连接到 av列表中。然后唤醒等待此特定缓存或任何可用缓存的进程。

注意，如果有两个进程在等待缓存——一个在等待该特定缓存，另一个则在等待任一可用缓存，那么两者在同一优先级上“睡眠”(4943、4955)，于是哪一个进程会获得此缓存是不确定的。

如果使前者的优先级稍高于后者 (例如，加一个偏置值 1)，那么就能较满意地解决此竞态问题。这种更改的不足之处是在某些相当奇特的情况下可能导致死锁。

如果出了一个错，例如将信息读入该缓存中时出了错，那么在该缓存中的信息可能是不正确的。4883行中的赋值语句保证了在这种缓存中的信息以后不会被错误地引用。“rkstrategy”(5403)、“rkintr”(5467)等可能设置“B_ERROR”标志。

让我们观察一下当完成一磁盘i/o操作时，对于所使用的缓存会发生些什么：

5471：“rkintr”调用“iodone”。

5026：“iodone”设置“B_DONE”标志。

5028：“iodone”调用“brelse”。

4887：“brelse”清“B_WANTED”、“B_BUSY”和“B_ASYNC”标志，但是不清“B_DONE”标志。

……

4948：“getblk”查到该缓存，然后调用“notavail”。

5010：“notavail”设置“B_BUSY”标志。

4759：“bread”(它调用了getblk)发现“B_DONE”标志已设置，然后退出。

注意，“notavail”将缓存头从av列表中移出，而“brelse”则将其返回av列表。“getblk”将缓存头从一个b列表移至另一个b列表。

17.7 binit(5055)

“main”调用此过程(1614)以初始化缓存池，它建立空[illegible]

- av列表(bfreelist是列表头)。
- 针对null设备的b列表(dev==NODEV)(bfreelist再次是[illegible]
- 针对每一主设备类型的b列表。

对于每个缓存：

- 其缓存头连接到设备“NODEV”(-1)的b列表中。
- 其地址设置到它的缓存头中(5067)。
- 其缓存标志设置为“B_BUSY”(这一操作似乎并无实[illegible]
- 调用“brelse”，将其缓存头连接到av列表中(5073)。

块设备数记录在变量“nblkdev”中。在“getblk”(49[illegible](6720)中用其检查“dev”的值。查看“bdevsw”(4656)可[illegible]么“nblkdev”也将被设置为8。

按下列方式进行编辑，可获得这一结果：

```
5084/m/5081/                    "nblkdev=i";
5084/m/5077/                    "i++"
```

17.8 bread(4754)

这是读块设备的标准过程。它由下列各过程调用：

wait	(3282)	iinit	(6927)
breada	(4799)	alloc	(6973)
statl	(6051)	ialloc	(7097)
smount	(6116)	iget	(7319)
readi	(6258)	iupdat	(7386)
writei	(6305)	itrunc	(7426、7431)
bmap	(6472、6488)	namei	(7625)

“ getblk ”找到一缓存。若其“ B_DONE ”标志已设置，则无需进行i/o操作。

17.9 breada(4773)

与“ bread ”相比，“ bread ”增加1个参数。此过程仅由“ readi ”调用(6256)。

4780：检查是否为所要求的块已分配了一缓存。 (该缓存可能现在还不能使用，但至少它已经存在)。

4781：如果还没有为所要求的块分配一缓存，则启动所需的读操作，但是不等待该操作结束。

4788：查看“ 预读 ”(read ahead)块。如果并无相应缓存，则先为其分配一缓存 (4789)，若所需信息已在该缓存中，则释放它 (4791)。

缓存中，因此以异步方式启动一读操作。

块，则调用“ bread ”。

作结束。

下列各过程调用：

free	(7021)
update	(7221)
iupdat	(7400)

(6310)。

C ”标志，则相应i/o操作结束后此过程再返回。

志已设置，但是“ B_DELWRI ”标志没有设置 (注意，在“ geterror ”(5336)以检查出错标志。如果“ B_DELWRI ”此出错指示传送给相关进程是很难处理的。对“ geterror ”出此写操作的缓存头。

17.11 bawrite(4856)

此过程由“ writei ”(6310)和“ bdwrite ”(4845)调用。取决于对要写的块是否全部或部分填写好对应的缓存，“ writei ”调用“ bawrite ”或者“ bdwrite ”。

17.12 bdwrite(4836)

此过程由“ writei ”(6311)以及“ bmap ”(6443、6449、6485、6500和6501)调用。

4844：若此设备是磁带驱动器，则不延迟进行写操作，使各个写磁带操作按序进行。

4847：设置“ B_DONE ”、“ B_DELWRI ”标志，然后调用“ brelse ”将此缓存连接到av列表中。

17.13 bflush(5229)

此过程由“ update ”调用(7201)，而“ update ”则由“ panic ”(2420)、“ sync ”(3489)和“ sumount ”(6150)调用。

“ bflush ”搜索av列表，从中找出“ 延迟写 ”的各块，并强制将它们以异步方式写出。

注意，因为“ notavail ”调整av列表的连接，所以在每次找到“ 延迟写 ”块后，重新启动该搜索过程(它在处理机优先级6级运行)。

也请注意，只有“ update ”以“ dev ”为“ NODEV ”调用“ bflush ”，所以可以简化5238行。

17.14 physio(5259)

调用此例程以处理“ 原始 ” (raw)输入/输出操作，这长。

“ physio ”由“ rkread ”(5476)和“ rkwrite ”(5483)调用组“ cdevsw ”中的项，也就是它们是字符设备项。

“ 原始i/o ”不是UNIX的关键特性。对于磁盘设备，这文件系统作为一个整体的完整性 (请参见UPM的ICHECK(\完整的若干磁道而不是一次一块。

注意对“ start ”的说明(5261)。因为所使用的实际参数任何值，所以这种形式的说明是否确实需要呢？