第20章 文件系统

在大多数计算机系统中有多个外部存储设备用于存储文件。本章讨论 UNIX对整个文件集合和文件存储设备的管理。首先要介绍几个定义：

文件系统：存放在单个面向块的存储设备上、具有层次结构的目录系统的各文件的整体集合。

存储设备：能够存储信息的设备(特别是磁盘包、DEC磁带等)。

存取设备：为存储设备传输信息的机构。

仅当存储设备插入到一个存取设备中时，它才是可存取的。在这种情况下，对存储设备的引用是通过对存取设备引用而实现的。

如果满足下列条件，则可将一存储设备视为一文件系统卷：

1) 信息记录在可编址的存储块中，每块 512个字符，可对各存储块单独进行读或写操作。(注意，IBM兼容磁带并不满足此条件。)

2) 信息记录在存储设备上，它满足某种一致性标准：

- #1块格式化为“ 超级块 ”(super block)。
- #2块至#(n+1)块(n记录在超级块中)包含一张“ inode ”表，它引用记录在该存储设备上的所有文件，但不引用任一其他文件。
- 记录在该存储设备上的目录文件引用所有在同一存储设备上的文件，亦即，一文件系统卷是由自包含的文件、目录和“ inode ”表集组成的。

若一个存储设备位于存取设备中，而且这种存在已由操作系统正式识别，则称一文件系统卷是安装的。

20.1 超级块(5561)

“ 超级块 ”总是存储设备上的 #1块。(#0块常被忽略，用于与 UNIX不一定有关的其他方面。)

“ 超级块 ”包含用于分配资源的信息，亦即存储块和“ inode ”表项。在文件系统卷已被安装期间，在内存中有一“ 超级块 ”副本，其在那里进行更新。为了防止在存储设备上的副本变得太陈旧，定期将内存副本的信息写至存储设备的超级块中。

20.2 mount表(0272)

对于每一安装的文件系统卷，都使用“ mount ”表中一项。每个“ mount ”表项包含下列信息：

- 文件系统卷所安装的设备。
- 指向一缓存头的指针，在该缓存头所管理的缓存中存放该设备的“ 超级块 ”。

• 一个“ inode ”指针。

“ mount ”表按下列方式引用：

• iinit(6922)由“ main ”调用(1615)，它为根设备构造一“ mount ”项。

• smount(6086)是一个系统调用，它为附加的设备构造“ mount ”项。

• iget(7276)，若其遇到一个带“ IMOUNT ”标志的“ inode ”，则搜索“ mount ”表。

• getfs(7167)搜索“ mount ”表以找到并返回特定设备“ 超级块 ”的指针。

• update(7201)周期地被调用，它搜索“ mount ”表，找到应从内存表写至存放在文件系统卷上表的信息。

• sumount(6144)是一个系统调用，它从“ mount ”表中删除相应项。

20.3 iinit(6922)

此过程被“ main ”(1615)调用，它为根设备初始化“ mount ”表项。

6926：对根设备调用“ open ”例程。注意，“ con5.c ”(4695)中定义了“ rootdev ”。

6931：将根设备“ 超级块 ”的内容复制到一缓存区，该缓存并不与任一特定设备相关连。

6993：将“ mount ”表的＃0项分配给根设备。只对其3个元素中的2个显式地赋初值。决不会引用其第3个元素，“ inode ”指针。

6936：显式清除存放在“ 超级块 ”中的各锁变量 (当上次写该“ 超级块 ”至文件系统卷上时，可能已设置了这些锁)。

6938：将根设备安装为“ 可写 ”状态。

6939：系统按记录在“ 超级块 ”中的时间设置其当前时间和日期。若此系统已停止了相

需重置时间。

统卷的主要过程是：

并使该设备准备就绪。

的命令：

呈执行“ mount ”系统调用，向其传送的参数是分别指向 1

找到一个面向块的存取设备。

好调用“ namei ”以译码第2个文件名。注意，“ trap ”将

(2770)。

牛是否满足下列条件：

文件。

6103：搜索“ mount ”表，查找一空闲项(mp->m_bufp==NULL)，或者已经为该设备分配使用的项(mount表定义在0272行)。

6111：“ smp ”应当指向“ mount ”表中的一合适项。

6113：执行以设备名和读/写标志作为参数的相应“ open ”例程。正如前面已见到的，RK05磁盘的“ open ”例程是空操作。

6116：从设备读＃1块。该块是“ 超级块 ”。

6124：将“ 超级块 ”从与“ d ”相关连的缓存中复制到与“ NODEX ”相关连的缓存中，在该设备被拆卸之前，不释放与“ NODEV ”相关连、存放“ 超级块 ”信息的缓存。

6130：“ ip ”指向第2个命名文件的“ inode ”。现在，此“ inode ”增加了“ IMOUNT ”标志。其效果是迫使“ iget ”(7292)在该文件系统卷装配期间忽略该文件的正常内容。 (实际上，此文件通常是特地为此目的而创建的空文件。)

20.6 注释

1) 一个装配设备的“ 读/写 ”状态只取决于向“ smount ”提供的参数。并无意图对硬件“ 读/写 ”状态进行检测。于是，一张“ 写保护 ”的盘若并未安装为“ 只读 ”，那么系统就会不断地为此报错。

2)“ mount ”过程对“ 所装配的文件系统卷 ”不执行任何类型的标号检查。在很少对文件系统卷作重新安排情况下，这是合理的。但是，在不断对文件系统卷作装配拆卸、重装配情况下，采取一些方法对正确的文件系统卷是否已装配进行验证似乎是必要的。还有，如果一文件系统卷包含敏感信息，那么最好还应采取某种形式的口令字保护。在“ 超级块 ”(5575)中还有足够空间可以存放一个名字和一个加密口令字。

20.7 iget(7276)

此过程由“ main ”(1616、1618)、“ unlink ”(3519)、“
7664)调用。调用此过程时的两个参数惟一地标识了一个文
文件的“ inode ”编号。“ iget ”返回指向内存“ inode ”表中

在执行“ iget ”过程时，首先搜索内存“ inode ”表，
经有一项与该文件相关连。如果没有，那么“ iget ”创建此

7285：搜索内存“ inode ”表……

7286：如果对于指定的文件已存在相关项……

7287：若该项已上锁，则睡眠。

7290：再试。(注意，因为该项可能已经消失，所以要

7292：如果“ IMOUNT ”标志已设置……。这是非常
论。

7302：如果“ IMOUNT ”标志没有设置，则使该“ inc
标志，返回该“ inode ”的指针。

7306：记住“ inode ”表中的第1个空闲项位置。

7309：如果“ inode ”表中各项都已使用，则向操作员发送一条消息，并以出错返回。

7314：到达此点时，将填写在“ inode ”表中找到的一空闲项。

7319：读包含文件系统卷“ inode ”的块。注意，此处使用“ bread ”而非“ readi ”，并认为从＃２块开始存放“ inode ”信息，而且“ inode ”编号从1(而不是0)开始。

7326：在此点上如因读操作出错，那么并不向系统其他部分报告。

7328：复制相关“ inode ”信息。此段代码隐式地使用了文件“ ino.h ”中的内容。对此文件的内容在整个UNIX源代码中都不作显式引用。

让我们再回到尚未说明清楚的部分：

7292：发现“ IMOUNT ”标志已设置。该标志是在装配一文件卷时由“ smount ”设置的。

7293：搜索“ mount ”表以找到指向当前“ inode ”的项。虽然搜索此表的开销不算太大，但是在“ inode ”中，例如在“ i_lastr ”字段中似乎可以存放一“ 指向 mount表项的指针 ”，这可以减少时间和代码空间。

7396：将“ dev ”和“ ino ”重新设置为装配设备编号和被装配文件系统卷根目录的“ inode ”号。重新开始。

显然，因为“ iget ”是由“ namei ”(7534、7664)调用的，这种技术使得在装配文件系统卷上的整个目录结构被集成到原先已存在的目录结构中。如果我们暂时不考虑目录结构可能会与树结构不同，那么对于装配文件系统卷，我们可以认为是用一棵子树代换了现存树上的一个叶节点。

20.8 getfs(7167)

句以外，几乎不再需要作更多的解释。此过程是由下列各

“ ialloc ”　　(7072)
“ ifree ”　　(7138)
“ iupdat ”　　(7383)

字符指针，亦即无符号整型，此处灵活地使用了“ n1 ”和
行一次测试。

卷中存放的信息尽可能保持为最新的。此过程的注释　(从
能(以相反顺序)。
’(3486)。通过“ shell ”命令“ sync ”调用“ sync ”系统调
续运行，其功能是每隔 30s调用“ sync ”一次。(见UPM

sumount ”先调用“ update ”(6150)。在全部停止系统活动
”以执行系统的最后1个动作。
在执行中，则立即返回。

7210：搜索“ mount ”表。

7211：对于每一个被装配的卷，……

7213：除非最近没有修改过该文件系统，或者“ 超级块 ”已上锁，或者该卷是以“ 只读 ”方式装配的，……

7217：更新“ 超级块 ”，将其复制到一缓存中，然后将该缓存的内容写到相应文件系统卷中。

7223：搜索“ inode ”表，对于每一个非空闲项上锁，若需要，则调用“ iupdat ”更新在文件系统卷上的“ inode ”项。

7229：允许以后再执行“ update ”。

7230：“ bflush ”(5229)写各“ 延迟写 ”块。

20.10 sumount(6144)

此系统调用删除“ mount ”表中一装配设备项。其功能是：从存取设备取出存储设备之前，先正常终止针对该设备的数据传送。

6154：在“ mount ”表中搜寻相应项。

6161：搜索“ inode ”表，若找到该设备上的任一文件未决项，则出错返回，并且不改变“ mount ”表项。

6168：清“ IMOUNT ”标志。

20.11 资源分配

现在我们集中注意力于单个FSV(文件系统卷)的资源管理。

在“ bmap ”要求下，“ alloc ”从自由列表中分配存储块。“ itrunc ”(由core、openl和iput调用)调用“ free ”，它将存储块返回至自由列表。

“ ialloc ”填写FSV“ inode ”表项，而“ ialloc ”则由“ maknode ”和“ pipe ”调用。FSV“ inode ”表中的项由“ ifree ”删除，而“ ifree ”则由“ iput ”调用。

FSV的“ 超级块 ”是各资源管理过程的中心。“ 超级块 ”包含下列各种信息项：

- 长度信息(可用的总资源)。
- 最多包含100可用存储块的列表。
- 最多包含100可用“ inode ”项的列表。
- 为控制对上述列表的外理而需要的锁。
- 标志。
- 最近一次更新的时间。

如果该文件系统卷可用“ inode ”项内存列表已空，则读在FSV上的整个表，并搜索寻找空闲项以重构该表。相反，若可用“ inode ”内存列表溢出。则简单地忽略以后释放的项，在必要时可以重新找到这些项。

对可用存储块，则使用不同的策略。这些块被安排组织成若干组，每组最多 100块。每组中的第1块(除第1组外)用于存放前一组100块的地址。最后一组往往是不完整的，也就是其中

包含的块数一般少于100，这些块的地址存放在“超级块”中。

在第一个块号列表中，其第1项的值是0，它起哨兵的作用。固为整个列表是按LIFO(后进先出)算法管理的，在该列表中若取出的块号为0，则说明该列表事实上是空的。

20.12 alloc (6956)

任何时候若需要一新存储块以存放一文件的某个部分，则“ bmap”调用此过程(6435、6448、6468、6480、6497)。

6961：将设备名变换成“超级块”指针。

6962：若“s_flock”已设置，则表示该可用块列表当前正被另一个进程更新。

6967：获得下一个可用存储块块号。

6968：若该列表中最近取出的一个块号是0，则整个列表现在为空。

6970：调用“badblock”(7040)，对从列表中取出的块号进行检验。

6971：若在“超级块”中的可用块列表现在为空，则刚找到的块将包含下一组100自由块的地址。

6972：设置“s_flock”以便使“超级块”中的可用块列表得到补充，以免其他进程得到“无空间”(nospace)指示。

6975：决定要被复制的列表中有效项项数。

6978：清“s_flock”，唤醒等待该锁的任何进程。

6982：清该缓存，使写入文件的信息在缺省情况下全部是0。

6983：设置“已修改标志”以保证“超级块”会由“ update”(7213)写到相应存储设备的

nl”(5825)和“iput”(7353)调用。在前2种情况下，“file”
该文件将被丢弃
特殊文件，那么不进行任何操作。
”中的块号列表。
则需间接存取。对块号大于7的块需要1次甚至2次间接存

存的257个元素。注意，在这里引用了缓存中的＃512和＃
的惟一场合。因为它们应当包含0，所以对计算并不起任何
并且在7432行也这样做，那么是否会对代码有所改善呢？
列表。
or”语句的结束部分。(同样，在7432行开始的for语句在

项。
ode”标记为已更新的。

20.14 free(7000)

此过程由“ itrunc ”调用(7435、7438、7442)，它将一个存储块插入至一设备的可用列表中。

7005：此行设置“ s_mod ”标志，但是在本过程结束处(7026行)又设置“ s_mod ”，对这种处理的原因不清楚。你对此有什么建议吗？

7006：遵守上锁协议。

7010：若对于该设备以前不存在自由块，则为其建立了一个只包含＃ 0块的列表。此值以后将被解释为列表尾端哨兵。

7014：若在“ 超级块 ”中的可用列表已满，则现在应将其写到 FSV上，设置“ s_flock ”。

7016：获得一缓存，它与正被插入至自由列表的块相关连。

7019：将该超级块列表及有效块数(在该列表之前)复制至缓存中；将该缓存的内容写至设备上；解锁(fp->s_flock=o；)，然后唤醒等待这把锁的所有进程。

7025：将返回的块加入可用列表中。

20.15 iput(7344)

在UNIX中，此过程是最常用的过程之一(有近30处调用此过程)，我们将会经常看到此过程的各种调用方法。

本质上，其操作非常简单，只是将作为参数传送来的“ inode ”的引用计数减1，然后调用“ prele ”(7882)清“ inode ”锁，接着执行某种所需的唤醒操作。

“ iput ”有一个重要副作用。若引用计数正将减少至 0，则表示应释放一资源。这可能是内存“ inode ”，也能是内存“ inode ”以及相关文件本身(若对

20.16 ifree(7134)

此过程由“ iput ”调用(7355)，它将一FSV“ inode ”
如果此列表已满(正如上面提到的)，或者若此列表已上锁(

20.17 iupdat(7374)

此过程由“ statl ”(6050)、“ update ”(7226)和“ iput ”(
特定“ inode ”项。如果对应的内存“ inode ”没有带“ IUP
执行任何操作。

下面的过程可能设置“ IUPD ”标志：

```
unlink (3530)       bmap (6452,6467)
chmod  (3570)       itrunc  (7448)
chown  (3583)       maknode (7462)
link   (5942)       namei   (7609)
writei (6285,6318)  pipe    (7751)
```

下面的过程可能设置“ IACC ”标志：

```
readi   (6232)    maknode   (7462)
writei  (6285)      pipe    (7751)
```

“iput”清标志位(7359)。

7383：如果已将FSV安装为“只读”，则立即返回。

7386：读包含FSV“inode”项的块。正如我们以前在“ iget”中所观察到的，此处用“bread”代替“readi”，我们采用的假定是：“inode”表从#2块开始，而有效“inode”号则从1开始。

7389：从内存“inode”复制相关信息。

7391：如果合适，更新最近存取时间。

7396：如果合适，更新最近修改时间。

7400：将更新后的块写回FSV。