大容量存储器与文件系统

林祥，PB16020923

1、大容量存储器简介

我们知道，程序执行时主要将信息存储在内存中，但由于内存太小，且具有易失性，并不适合长久保存数据。因此，计算机需要一些大容量外部存储器来备份信息，比如磁盘、磁带、光盘和固态硬盘等等，其中磁盘和固态硬盘是目前主要的外部存储器。速度、可靠性和价格是衡量大容量存储器性能的三个重要指标。

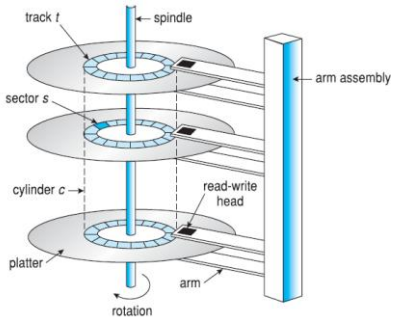


图 1

2、磁盘

（1）磁盘的结构

磁盘（Magnetic Disk）由许多扁圆盘片堆叠而成，每个盘片两面涂有磁性材料，旁边是机械臂杆及其磁臂，磁臂上的磁头在盘片上移动并进行磁记录来实现信息的读取和写入。磁盘片的表面被逻辑地划分为多个**磁道（Track）**，磁道再被划分为**扇区（Sector）**，位于同一磁臂位置的磁道集合组成**柱面（Cylinder）**。

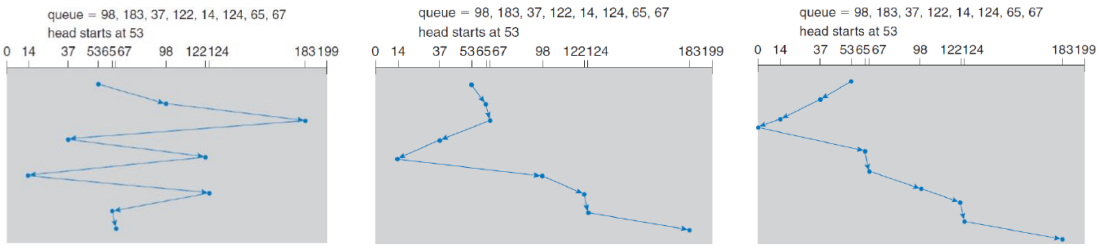
磁盘使用时会被驱动器高速旋转，磁头飞行于盘片的空气层上，定位到数据所在位置所需的时间由两部分组成：**寻道时间**（移动磁臂到所在的柱面）和**旋转等待时间**（等待某扇区旋转至磁头下）。定位时间越小，数据传输速率越高。磁盘旋转的角速度可以是匀速的或者变速的，由于扇区按角度划分，则前者的传输速率是内侧磁道高而外侧低，后者的传输速率是不变的，现在的磁盘主要采用前一方案。

（2）磁盘的管理

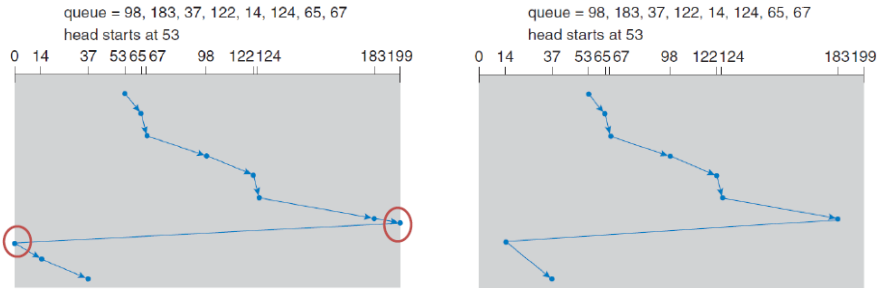
良好的管理可以更高效地使用磁盘。磁盘在低级格式化后被划分为许多扇区（带有特殊数据结构的逻辑块），一般并不直接使用（当成巨大的顺序数组），而是进一步分为一个或多个柱面组成的分区，然后进行逻辑格式化，创建某种**文件系统（File System）**对磁盘进行管理。其中一个重要的方面是对坏块的管理，当某些逻辑块损坏时，一种方法是跳过坏块，顺序后移后续块，另一种方法是每条磁道预留一些块作为备用。

（3）磁盘的调度

由于磁盘具有特殊结构，在转速一定时，需要对磁盘的访问队列进行**调度**才能避免过长的寻道时间。**FCFS**（先来先服务）调度是一种公平的调度算法，但一般性能不高。**SSTF**（最短寻道时间优先）调度优先处理靠近当前磁头位置的请求，性能有了很大提高，但可能导致一些请求长时间得不到满足。**SCAN**（扫描）调度将磁臂从一端移到另一端，顺序处理所到之处的请求，到达一端后反向扫描，如此循环，这种算法的问题在于，当一个请求在磁头从请求位置移动走后才被加入队列，则必须等待磁头到达一端并反向移动回来后才能处理。**C-SCAN** 调度是前者的一种改进，磁头到达一端后并不反向扫描，而是直接返回起始端重新开始扫描，这就解决了前一个问题，但实际上，如果在到达一端之前剩余的位置都没有请求，则磁头可以立刻返回，这就是 **C-LOOK** 调度，磁头只移动到一个方向上最远的请求。



FCFS　　SSTF　　SACN



C-SCAN　　C-LOOK

图 2

3、固态硬盘

固态硬盘（SSD，Solid-State Drives）主要由控制器和**闪存（Flash）**芯片组成，它具有速度快、质量轻、能耗小等优点，近来被广泛使用，但容量小，价格昂贵。闪存芯片被划分为不同的层次（Package>Die/Chip>Plane>Block>Page），最基本的存储单元是晶体管，每个单元可以存储一个 bit（**SLC**），也可以存储多个 bit（**MLC**）。固态硬盘的读写以 Page（4KB）为单位，当一个 Page 的信息需要被删除时，它只是被标记为无效，而且不支持对一个 Page 进行修改，只有在适当的时候以 Block 为单位进行擦除，而一个 Block 可以擦除的次数是**有限**的，因此固态硬盘需要地址映射机制和垃圾回收机制，以此延长使用寿命。

4、RAID

单个磁盘读写性能并不高，而且一旦损坏，将导致数据丢失。随着单位存储价格的降低，人们通过增加大量磁盘形成**磁盘冗余阵列（RAID）**，以此提高**性能**和**可靠性**。一方面，通过**冗余**提高可靠性，最简单也最昂贵的方法是**镜像**，即复制每个磁盘。另一方面，通过**分散和并行**改善性能，最简单的方式是**位级分散**，即一个字节的 8 个位分散到 8 个磁盘上，则并行读取时可以提高为 8 倍传输速率，也可以用**块级分散**的方式，一个文件的块分散在多个磁盘上。

RAID 具有不同的**级别**，体现对价格和效果的不同**权衡**，以下以 4 个盘为例介绍各级别。

（1）RAID0：使用块级分散，但没有任何冗余，因此没有提高可靠性。

（2）RAID1：即磁盘镜像，大大提高了可靠性，但花费巨大。

（3）RAID01：即 RAID0+1，先将数据分散到两块磁盘，再整体做镜像，这种方案同时提高了性能和可靠性，然而只要任意一块磁盘损坏，都只剩下一份镜像可用。

（4）RAID10：即 RAID1+0，先做镜像，再整体将数据分散到两块磁盘，这种方案比 RAID01 更优，当一个磁盘损坏时，RAID10 的两份分散皆可用，而 RAID01 只有一份可用。这两种方案都存在开销过大的问题。

（5）RAID2：以**汉明码（Hamming Code）**的方式将数据进行编码后按位级分散到各个磁盘上，汉明码方式是在原有数据中插入若干校验码，并且支持一位数据的纠错，如果某个磁盘损坏，可以通过其他磁盘重新计算以恢复数据。4 位数据的存储需要额外增加 3 位校验码，这比 RAID1 节省了一个磁盘，但读写时需要对数据即时地进行校验，因此效率较低。

（6）RAID3：以**奇偶校验（Parity Check）**的方式在多位数据后增加一位校验位，表示前面数据位中 1 的个数是奇数或偶数。这种方案使用位级分散，只需要一个额外的校验磁盘，但是当读写较频繁时，校验盘将会负载过大，成为系统性能的瓶颈。

（7）RAID4：和 RAID3 类似，但使用块级分散。

（8）RAID5：是对 RAID4 的改进，将奇偶校验信息分散存储到各个磁盘上而不是单个校验磁盘，这就避免了校验盘负载过大的情况，是目前应用较广泛的方案。

（9）RAID6：在 RAID5 基础上增加了一位校验码（不再是奇偶校验），这样就可以容忍两个磁盘出错。

5、文件系统概述

如前文所述，磁盘只是纯粹的存储介质，一般磁盘格式化并不直接顺序存储数据，而是建立特定的**文件系统（File System）**来更好地管理磁盘。文件系统本身并不是磁盘也不是操作系统，而是存储在磁盘上，一个磁盘通常可以支持不同的文件系统。

文件系统通常具有**文件**和**目录**两种抽象的对象。

文件（File）是创建者定义的相关信息的集合，操作系统将其映射到存储器中，对用户而言，文件是最小的逻辑分配单元。文件的内容可以是文本、数字或二进制数据，其组织形式可以是自由的，也可以具有严格的格式，这通常由创建者定义。

为了能识别和引用不同的文件，文件通常具有各种**属性（Attributes）**。文件名是一个字符串，它独立于进程、用户甚至操作系统，这个字符串是否区分大小写取决于具体的操作系统。标识符是在文件系统中识别文件的唯一标签，通常为数字。类型则指明了这是特定类型的文件系统的文件。位置是文件在存储设备上的位置指针。大小记录了文件的当前大小或者最大允许容量。时间日期包括文件创建或者上次修改的相关时间信息。保护和权限则指明了不同用户能否对文件进行相关操作（读、写、执行等）。通常文件都具有以上这些属性，还有一些其他属性，但个别文件系统可能并不支持某些属性。值得注意的是，文件内容并不包括属性，所有的文件属性信息都存储在目录结构中。

目录（Directory）是用来管理和组织文件的结构，其本质也是一个文件，如右图。一个目录文件包含许多目录项，每一个**目录项（Entry）**保存一个文件的属性信息（名称、位置等），表示这个文件属于这个目录。一个目录下的允许再包含目录文件，这就形成了以根目录"/"为根的多层次的树状结构，其叶结点为文件。一个文件的名称前面加上各级父目录名称，用"/"分隔，这就形成了文件的路径名称。文件路径通常用于定位一个文件在文件系统中的层次和位置，因为它在整个文件系统中是独一无二的。如果以根目录为起点，称为**绝对路径**，如果以某个目录为起点，则称为**相对路径**。

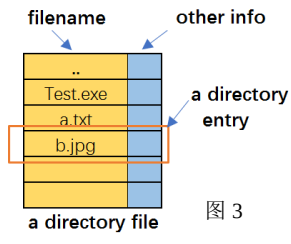


图 3

6、文件系统的布局与操作

各种文件系统的本质区别在于采用不同的存储**布局结构（Layout）**，以及对文件的**操作（Operations）**的不同实现方式。

用户或进程通过操作系统的调用接口对文件进行操作，而操作系统为了能支持不同的文件系统，通常先实现一个**虚拟文件系统（VFS）**，在这个层面上隐藏了实现的细节，定义了抽象的操作集合，在它之下才是各个文件系统具体操作的实现。

（1）操作系统对文件进行的**操作**主要有：

①创建：在磁盘中申请一块空间，然后在文件所在的目录增加一个目录项，记录新文件的相关属性。值得注意的是，创建一个文件并不同时打开文件。

②打开：对文件进行操作时，需要通过路径逐级搜索和定位文件，为了避免频繁地进行这个过程，在首次使用文件时，需要显式（或隐式）地调用 open()来"打开"文件，这将在内存中创建一个 FILE 类型结构体来存储文件的位置信息和其他属性，操作系统则维护一个包括所有打开文件的 FILE 结构体的列表，称为**打开文件表（Open-files table）**。进程需要对文件进行操作时，提供**文件标识符（File Descriptor）**即打开文件表的索引，而不再是路径。由于多个进程可以同时打开一个文件，为了避免相互影响，每个进程还需要单独维护一个进程的打开文件表，用于存储与进程相关的信息，比如对文件读写的偏移指针。

③读：首先进程向操作系统提供文件标识符，操作系统在打开文件表中查找到对应的 FILE 结构，从中获取文件位置和属性，在磁盘中定位和读取相应的数据后存储在内核空间中，最后写入进程指定的用户内存空间中。

④写：与读相反，先从用户内存空间读取数据到内核空间，然后操作系统将根据不同的策略适时地将数据写回磁盘。若采用**写直达法（Write-through mode）**，内核中的数据立刻被写入磁盘；若采用**写回法（Write-back mode）**，只有当内核空间中的数据需要进行更新时，这些数据才被写入磁盘。

⑤文件内重定位：直接修改文件偏移指针为指定值。

⑥关闭：当一个文件不再被使用时，关闭这个文件，打开文件表将移除对应的项。

⑦删除：将文件在磁盘中的空间释放，并删除目录文件中的对应项。

⑧截短：删除文件的内容但保留其属性，文件长度变为 0。

以上①③④⑤⑦⑧构成了对文件操作的最小集合，由它们可以组合出其他的文件操作。

（2）文件系统的布局实例

对于连续的磁盘空间，不同的文件系统对其进行不同的**划分和组织**，这就形成了不同的布局。文件系统的实现至少应该满足以下几条要求：

①被写入的数据能够被重新检索出并读取。

②对文件的操作应该是高效的。

③当删除一个文件时，文件系统能清除对应的空间。

更详细的要求就是对上面各种文件操作的具体实现以及空间的具体分配和对空闲空间的管理，下面介绍几种布局的实现并逐步进行改进。

①连续分配方式

这是最直接最朴素的想法，在磁盘连续空间的头部存放根目录，目录中记录各个文件的起始位置和结束位置，文件紧跟着存放在目录的后续空间中（如图 4）。这种策略使得删除操作十分简单，只需要在目录文件中删除对应项就可以了。但是，在经过长时间使用后，磁盘的空闲空间将**碎片化**地分散在各个位置，文件系统无法迅速统计空闲空间大小。最大的问题是，如果需要存储一个大文件，有可能无法找到一个足够大的空闲片段来存放，即使存在并且使用了最优策略分配后，在需要增加文件内容时将面临两端皆有其他文件而无法**扩展**的窘境。一种解决方法是移动现有的文件使空闲空间连续，但是这将开销巨大。

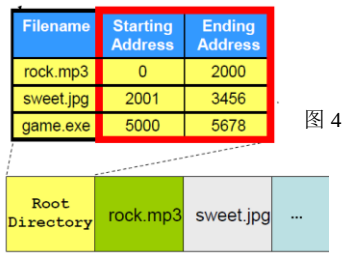


图 4

②链式分配方式

既然大文件可能没有足够大的空闲片段来存放，一个自然的想法是将磁盘划分为**相等大小的块/簇（Cluster）**，文件按块进行存储（如图 5）。这些块可以不在连续的空间里，只需要在目录文件中记录每个文件占用的块的序号即可。但是，由于目录文件是目录项的数组，每一项的长度是固定的，因而对于分散的块序号的记录是比较困难的。

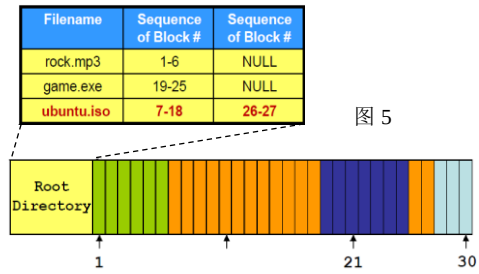


图 5

一种改进的方法是，目录项只记录文件起始块的序号，而在每个块中留出部分空间存放指向下一个块的指针（如图 6），这种情况下，目录项中还需要记录文件的大小，因为文件的最后一块可能并没有填满。另外，还需要在磁盘最前端留出一块空间指向空闲空间的链表。这种策略带来的问题是随机访问性差，访问一个文件需要从头部开始。还有一个问题是关于块大小的选择，如果块太小，则块的指针占用了太多空间，如果块太大，则每个文件最后一个块的浪费将是巨大的。

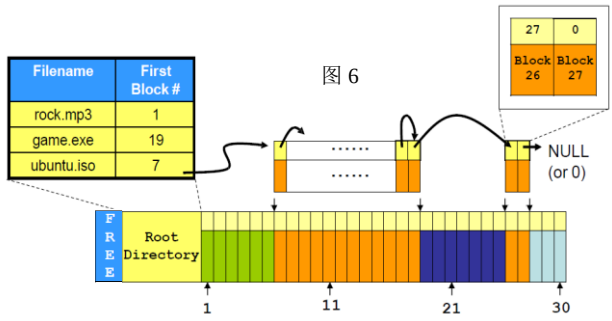


图 6

③链式分配方式的改进——FAT

为了改善随机读写性能，可以将下一个块的位置信息以数组形式统一存放在磁盘头部（如图7），称为**文件分配表（FAT，File Allocation Table）**。数组中每一项存放着当前下标对应的块的下一个块的序号（地址），根据地址长度的不同，可以分为不同版本**（FAT12/16/32）**。这种策略下寻找文件的方式为先从根目录中读取文件的第一个块的地址，然后到 FAT 中依次获得下一个块的地址并读取每个块。写入文件时，则先从空闲空间信息获取下一个空闲块的位置，然后写入数据并更新 FAT 和空闲块信息。删除文件时，相关的 FAT 置为 0，更新空闲空间信息并删除目录中相关项。

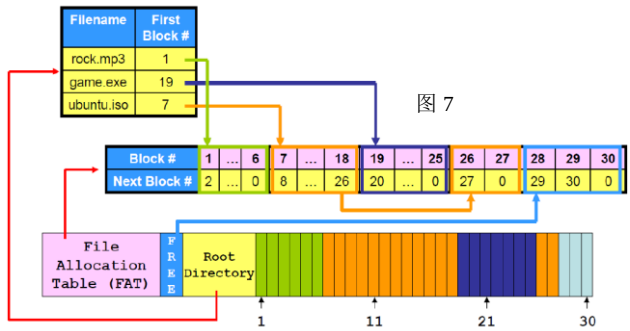


图 7

以 FAT32 为例，磁盘空间分配如图 8 所示，头部是引导扇区（存储扇区大小等信息）、文件系统的相关信息（包括空闲空间信息）和预留的空间，其后是两个文件分配表，接下来就是根目录和存储的文件。

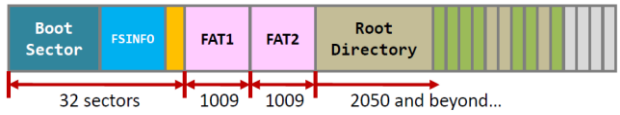


图 8

④索引分配方式

如果对 FAT 进行分割，每个文件对应的 FAT 单独存放在一个**索引节点（Index Node）**中，而目录项中指向每个文件对应的索引节点，这就是索引分配方式（如图9）。由于每个文件的 FAT 大小不同，则索引节点将是变长的，这对于管理十分不便。事实上的做法是，维护一个索引数组，每个索引节点长度固定，索引节点内部有 12 个指针指向**直接块（Direct Block）**，即数据块，还有一个指针指向**一级间接块（Indirect Block）**，间接块中的每一项才指向数据块，另外还各有一个指针指向二级间接块和三级间接块。假设一个块的大小为 2^Xbyte，块地址 32 位（4byte），则这种方式下单个文件最大可存储$12 * 2^X + 2^{2X-2} + 2^{3X-4} + 2^{4X-6}$byte。

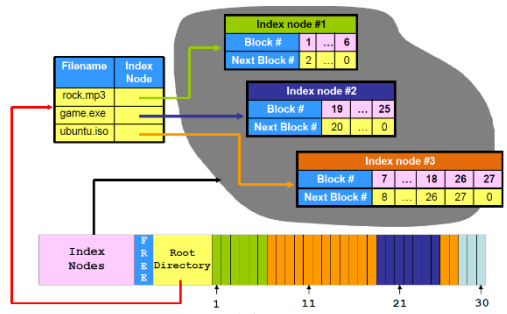


图 9

UNIX 和 Linux 使用类似的索引方式的文件系统，称为 **Ext2/Ext3/Ext4**。以 Ext2 为例，磁盘被划分为许多组（如图 11）。每个组的头部都是一个**超级块（Superblock）**，其中包括全部索引节点数、全部空闲块数等信息。其后是**组描述符表（GDT，Group Descriptor Table）**，包括之后各部分的起始位置信息。然后是**块位图（Block Bitmap）**和**索引位图（Inode Bitmap）**，它们的每一位表示一个块/索引是否已经被使用（1 表示已使用），最后就是索引节点数组和数据块了。

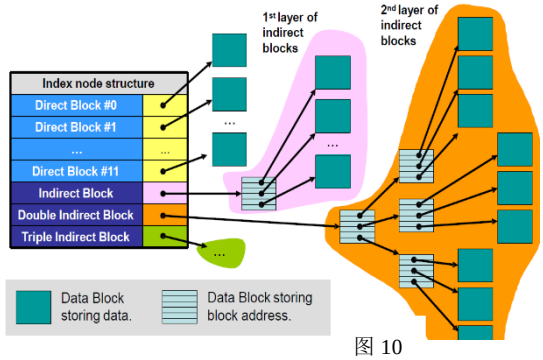


图 10

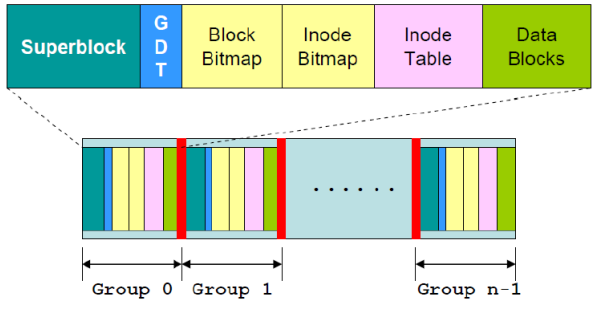


图 11

参考文献:

[1] Abraham Silberschatz. 操作系统概念. 高等教育出版社, 2010.1.