7.4 外存分配方法

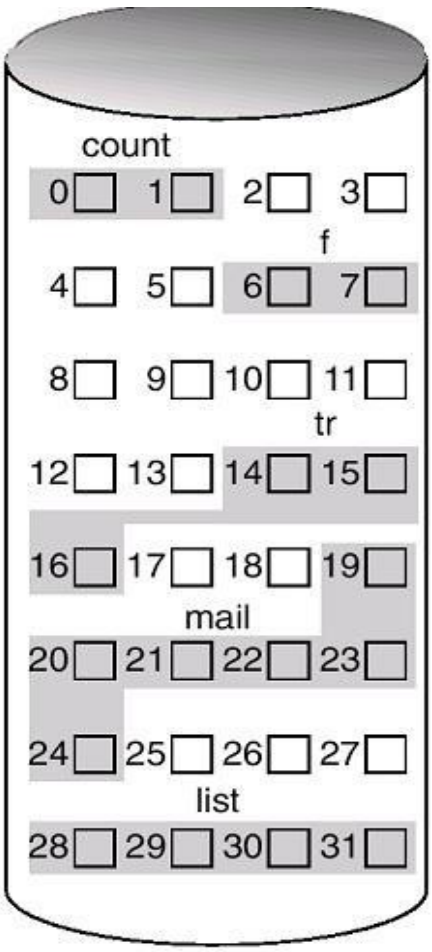
磁盘的直接访问特点使大家能够灵活地实现文件。在绝大多数情况下，一个磁盘可存储许多文件。主要问题是如何为这些文件分配空间，以便有效地使用磁盘空间和快速地访问文件。常用的主要**磁盘空间分配方法**有三个：**连续、链接和索引**。每种方法都有其优点和缺点。有的系统对三种方法都支持。但是，更为常见的是一个系统只提供对一种方法的支持。

1. 连续分配方法

连续分配（contiguous-allocation）方法要求每个文件在磁盘上占有一组连续的块。磁盘地址为磁盘定义了一个线性序列。采用这种序列，假设只有一个作业访问磁盘，在访问块 b后访问块 b + 1 通常不需要移动磁头。当需要磁头移动（从一个柱面的最后扇区到下一个柱面的第一扇区），只需要移动一个磁道。因此，用于访问连续分配文件所需要的寻道数最小，在确实需要寻道时所需要的寻道时间也最小。使用连续分配方法的 IBM VM/CMS 操作系统提供了很好的性能。

文件的连续分配可以用第一块的磁盘地址和连续块的数量来定义。如果文件有 n 块长并从位置 b 开始，那么该文件将占有块 b，b + 1，b + 2，…，b + n – 1。一个文件的目录条目包括开始块的地址和该文件所分配区域的长度，参见图 11.5。

对一个连续分配文件的访问很容易。要顺序访问，文件系统会记住上次访问过块的磁盘地址，如需要可读入下一块。要直接访问一个从块 b 开始的文件的块 i，可以直接访问块 b + i。因此



目录

文件	起始	长度
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

连续分配支持顺序访问和直接访问。

图 7.19 磁盘空间的连续分配

不过，连续分配也有一些问题。一个困难是为新文件找到空间。被选择来管理空闲空间的系统决定了这个任务如何完成；这些管理系统将在后面章节中讨论。虽然可以使用任何管理系统，但是有的系统会比其他的要慢。

连续磁盘空间分配问题可以作为在**动态存储 分配（dynamic storage-allocation）**问题的一个具体应用，即如何从一个空闲孔列表中寻找一个满足大小为 n的空间。从一组空闲孔中寻找一个空闲孔的最为常用的策略是首次适合和最优适合。模拟结果显示在时间和空间使用方面，首次适合和最优适合都要比最坏适合更为高效。首次适合和最优适合在空间使用方面不相上下，但是首次适合运行得更快。

这些算法都有**外部碎片（external fragmentation）**问题。随着文件的分配和删除，磁盘空闲空间被分成许多小片。只要空闲空间分成小片，就会存在外部碎片。当最大连续片不能满足需求时就有问题；存储空间分成了许多小孔，但没有一个足够大以存储数据。因磁盘空间的总数和文件平均大小的不同，外部碎片可能是一个小问题，但也可能是个大问题。

某些老式微型计算机系统对软盘采用了连续分配。为了防止由于外部碎片而导致的大量磁盘空间的浪费，用户必须运行一个重新打包程序，以将整个文件系统复制到另一个软盘或带上。原来的软盘完全变成空的，从而创建了一个大的连续空闲空间。接着，该重新打包程序又对这一大的连续空闲空间采用连续分配方法，以将文件复制回来。这种方案有效地将所有小的空闲空间**合并（compact）**起来，因而解决了碎片问题。这种合并的代价是时间。这种时间代价对于使用连续分配的大硬盘是很严重的，合并所有的空间可能需要几小时，一周需要运行一次。有些系统要求这个功能线下执行、卸载文件系统。在这**停机期间（down time）**，不能进行正常操作，因此在生产系统中应尽可能地避免合并。大多数现代需要整理碎片的系统能够和正常的系统操作一起在线执行合并，但是性能下降会很明显。

连续分配的另一个问题是确定一个文件需要多少空间。当创建文件时，需要找到和分配文件所需要的总的空间。创建者（程序或人）又如何知道所创建文件的大小？有时，这个确定比较简单（例如，复制一个现有文件）；然而通常来说，输出文件的大小是比较难估计的。

如果为一个文件分配太小的空间，那么可能会发现文件不能扩展。尤其是采用了最优适合分配策略，文件两端可能已经使用。因此，不能在原地扩大文件。这时有两种可能性。第一，可以终止用户程序，并加上合适的错误消息。用户必须分配更多空间并再次运行程序。这些重复运行可能代价很高。为了防止这些问题，用户通常会过多地估计所需的磁盘空间，从而导致了空间浪费。另一种可能是找一个更大的孔，复制文件内容到新空间，释放以前的空间。只要空间存在些动作就可以重复，不过这耗费时间。当然，在这种情况下，用户无需知道这些具体动作；系统虽然有问题但可继续运行，只不过会越来越慢。

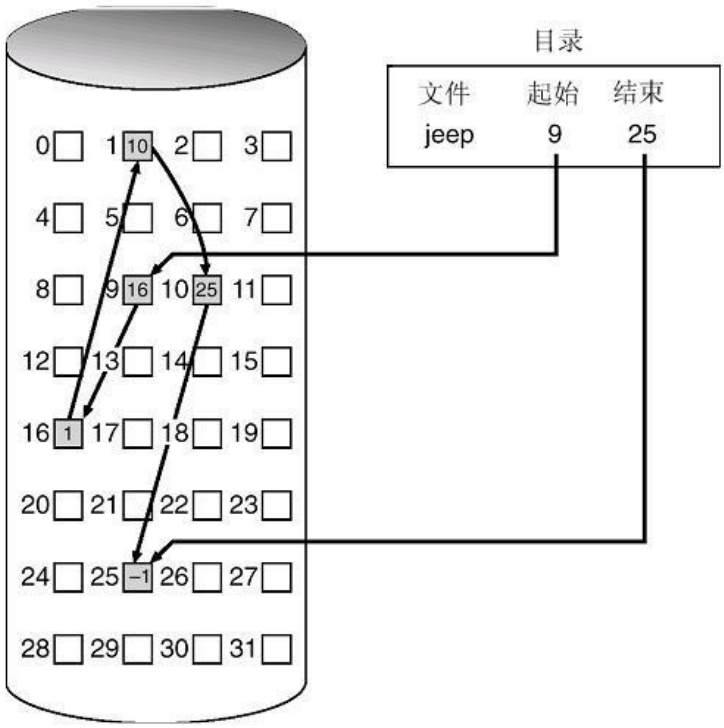
即使事先已知一个文件所需的总的空间，预先分配的效率仍可能很低。一个文件在很长时间内慢慢增长（数月或数年），仍需要为它最后的大小分配足够空间，虽然其中的大部分在很长时间内并不使用。因此，该文件有大量的内部碎片。

为了减少这些缺点，有的操作系统使用修正的连续分配方案。该方案开始分配一块连续空间，当空间不够时，另一块被称为**扩展(extent)**的连续空间会添加到原来的分配中。这样，文件块的位置就成为开始地址、块数、加上一个指向下一扩展的指针。在有的系统上，文件用户可以设置扩展大小，但如果用户设置不正确，将会影响效率。如果扩展太大，内部碎片可能仍然是个问题；随着不同大小的扩展的分配和删除，外部碎片可能也是个问题。商用VFS 使用扩展来优化性能。它是标准 UNIX UFS 的高性能替代品。

2. 链接分配方法

链接分配（linked allocation）解决了连续分配的所有问题。采用链接分配，每个文件是磁盘块的链表；磁盘块分布在磁盘的任何地方。目录包括文件第一块的指针和最后一块的指针。例如，一个有 5 块的文件可能从块 9 开始，然而是块 16，块 1，块 10，最后是块 25（图 7.20）。每块都有一个指向下一块的指针。用户不能使用这些指针。因此，如果每块有 512字节，磁盘地址为 4 字节，那么用户可以使用 508 字节。

要创建新文件，可以简单地在目录中增加一个新条目。对于链接分配，每个目录条目都有一个指向文件首块的指针。该指针初始化为 nil（链表结束指针值）以表示空文件。大小字段也为



0。要写文件就会通过空闲空间管理系统找到一个空闲块，然后这个新块被写入并链接到文件的尾部。要读文件，可以通过块到块的指针，简单地读块。采用链接分配没有外部碎片，空闲空间列表上的任何块可以用来满足请求。当创建文件时，并不需要说明文件大小。只要有空闲块，文件就可以增大。因此，无需合并磁盘空间。

图7.20 磁盘空间的链接分配

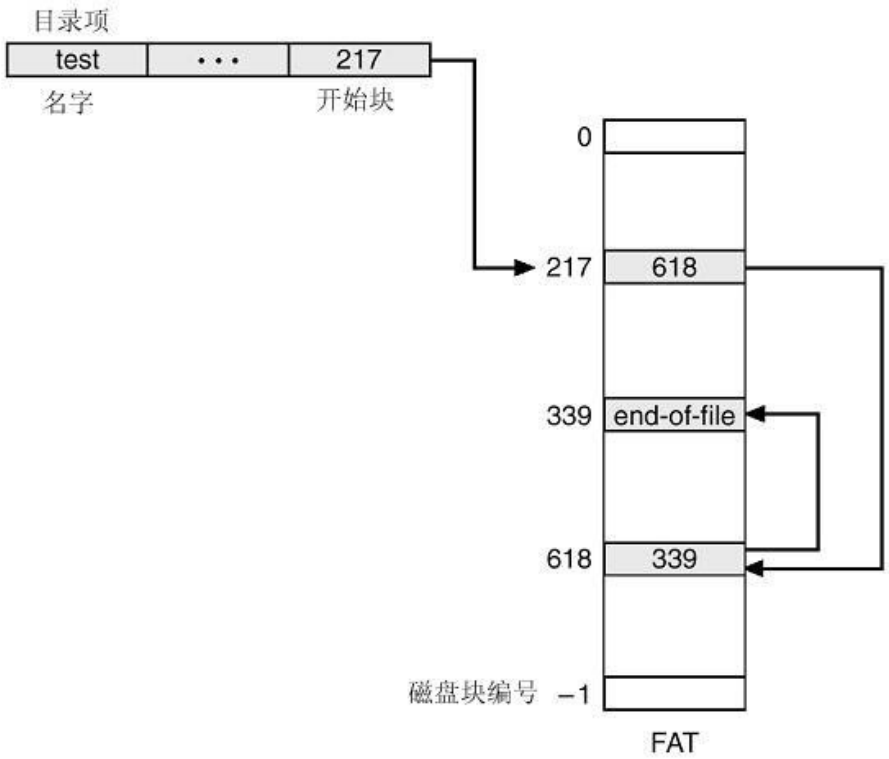
不过，链接分配确实也有缺点。主要问题是它只能有效地用于文件的顺序访问。要找到文件的第 i 块，必须从文件的开始起，跟着指针，找到第 i 块。对指针的每次访问都需要读磁盘，有时需要磁盘寻道。因此，链接分配不能有效地支持文件的直接访问。

链接分配的另一缺点是指针需要空间。如果指针需要使用 512 字节块中的 4 个字节，那么0.78%的磁盘空间将会用于指针，而不是其他信息。因而，每个文件也需要比原来较多的空间。

对这个问题的常用解决方法是将多个块组成**簇**（cluster），并按簇而不是按块来分配。例如，文件系统可能定义一个簇为 4 块，并以簇为单位来操作。这样，指针所使用的磁盘空间的百分比就更少。这种方法允许逻辑到物理块的映射仍然简单，而且提高了磁盘输出（更少磁头移动），并降低了块分配和空闲列表管理所需要的空间。这种方法的代价是增加了内部碎片，如果一个簇而不是块没有充分使用，那么就会浪费更多空间。簇可以改善许多算法的磁盘访问时间，因此应用于绝大多数操作系统中。

链接分配的另一个问题是可靠性。由于文件是通过指针链接的，而指针分布在整个磁盘上，想一下如果指针丢失或损坏会是什么结果。操作系统软件的 bug 或磁盘硬件的故障可能会导致获得一个错误指针。这种错误可能导致链接空闲空间列表，或另一个文件。一个不彻底的解决方案是使用双向链表或在每个块中存上文件名和相对块数。不过，这些方案为每个文件增加了额外开销。

一个采用链接分配方法的变种是**文件分配表（FAT）**的使用。这一简单但有效的磁盘空间



分配用于 MS-DOS 和 OS/2 操作系统。每个卷的开始部分用于存储该 FAT 表。每块都在该表中有一项，该表可以通过块号码来索引。FAT 的使用与链表相似。目录条目含有文件首块的块号码。根据块号码索引的 FAT 条目包含文件下一块的块号码。这种链会一直继续到最后一块，该块对应 FAT 条目的值为文件结束值。未使用的块用 0 值来表示。为文件分配一个新的块只要简单地找到第一个值为 0 的 FAT 条目，用新块的地址替换前面文件结束值，用文件结束值替代 0。一个由块 217, 618, 339 组成的文件的 FAT 结构如图 7.21 所示。

图 7.21 文件分配表

如果不对 FAT 采用缓存，FAT 分配方案可能导致大量的磁头寻道时间。磁头必须移到卷的开头以便读入 FAT，寻找所需要块的位置，接着移到块本身的位置。在最坏的情况下，每块都需要两次移动。其优点是改善了随机访问时间，因为通过读入 FAT 信息，磁头能找到任何块的位置。

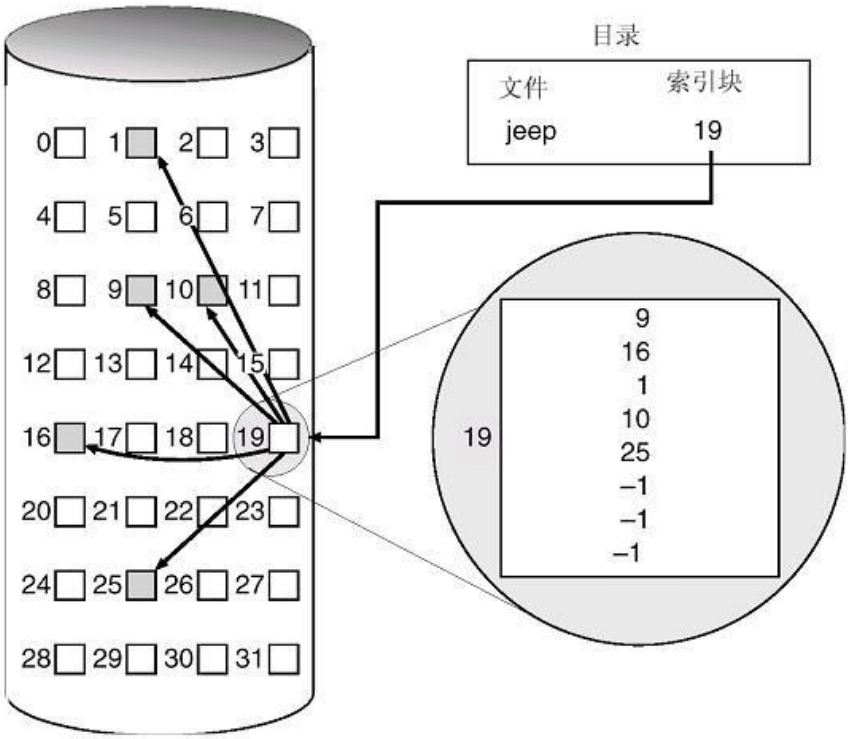
3. 索引分配方法

链接分配解决了连续分配的外部碎片和大小声明问题。但是，如果不用 FAT，那么链接分配就不能有效支持直接访问，这是因为块指针与块一起分布整个磁盘，且必须按顺序读取。**索引分配（indexed allocation）**通过把所有指针放在一起，即通过**索引块**解决了这个问题。

每个文件都有其索引块，这是一个磁盘块地址的数组。索引块的第 i 个条目指向文件的第 i 个块。目录条目包括索引块的地址(图 7.22)。要读第 i 块，通过索引块的第 i 个条目的指针来查找和读入所需的块。这一方法类似于内存管理所描述的分页方案。

当创建文件时，索引块的所有指针都设为 nil。当首次写入第 i 块时，先从空闲空间管理器中得到一块，再将其地址写到索引块的第 i 个条目。

索引分配支持直接访问，且没有外部碎片问题，这是因为磁盘上的任一块都可满足更多空



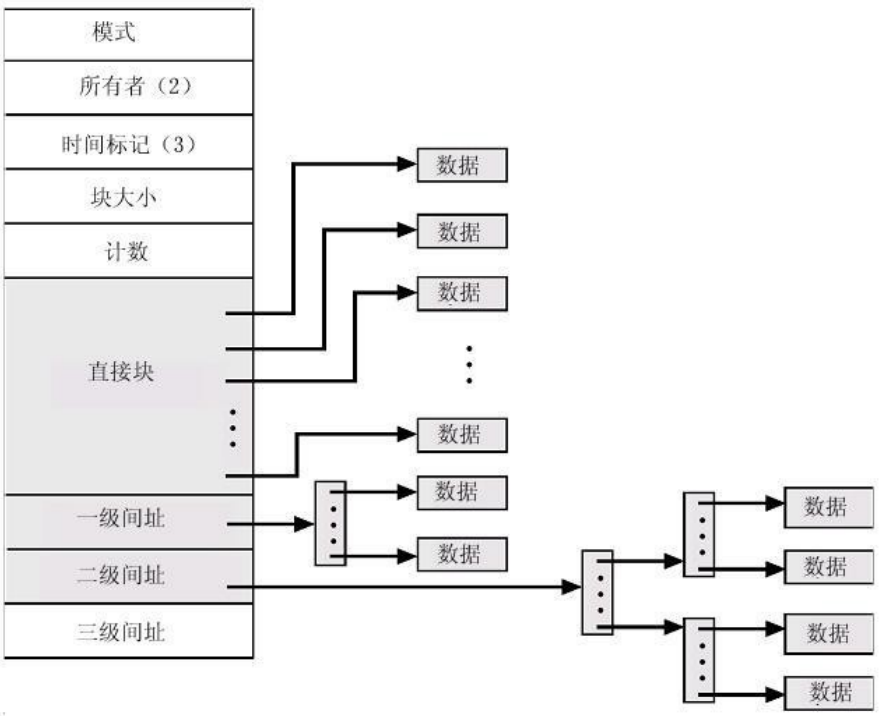
间的要求。索引分配会浪费空间。索引块指针的开销通常要比链接分配的指针开销要大。设想一下一般情况，每个文件只有一块或两块长。采用链接分配，每块只浪费一个指针。采用索引分配，尽管只有一个或两个指针为非空，也必须分配一个完整的索引块。

图 7.22 磁盘空间的索引分配

这也提出了一个问题：索引块应为多大？每个文件必须有一个索引块，因此需要索引块尽可能地小。不过，如果索引块太小，那么它不能为大文件存储足够多的指针。因此，必须采取一定机制来处理这个问题。针对这一目的的机制包括如下：

- **链接方案**：一个索引块通常为一个磁盘块。因此，它本身能直接读写。为了处理大 文件，可以将多个索引块链接起来。例如，一个索引块可以包括一个含有文件名的头部和一组头 100 磁盘块的地址。下一个地址（索引块的最后一个词）为 nil（对于小文件）或指向另一个索引块（大文件）。
- **多层索引**：链接表示的一种变种是用第一层索引块指向一组第二层的索引块，第二层索引块再指向文件块。为了访问一块，操作系统通过第一层索引查找第二层索引，再用第二层索引查找所需的数据块。这种方法根据最大文件大小的要求，可以继续到第三或第四层。对于有 4096 字节的块，可以在索引块中存入 1024 个 4 字节的指针。两层索引允许 1048576 个数据块，这允许最大文件为 4GB。
- **组合方案**：用于 UFS 的另一方案将索引块的头 15 个指针存在文件的 inode 中。这其中的头 12 个指针指向**直接块**；即，它们包括了能存储文件数据的块的地址。因此，（不超过 12 块）小文件不需要其他的索引块。如果块大小为 4KB，那么不超过 48KB

的数据可以直接访问。其他 3 个指针指向**间接块**。第一个间接块指针为**一级间接块**的地址。一级间接块为索引块，它包含的不是数据，而是那些包含数据的块的地址。接着是一个**二级间接块**指针，它包含了一个块的地址，而这个块中的地址指向了一些块，



这些块中又包含了指向真实数据块的指针。最后一个指针为**三级间接块**指针。采用这种方法，一个文件的块数可以超过许多操作系统所使用的 4字节的文件指针所能访问的空间。32 位指针只能访问 2^{32}字节，或 4GB。许多 UNIX如 Solaris 和 IBM AIX 现在支持高达 64 位的文件指针。这样的指针允许文件和文件系统为数 T 字节。图 7.23 显示了一个 Unix 的 inode。

索引分配方案与链接分配一样在性能方面有所欠缺。尤其是，虽然索引块可以缓存在内存中，但是数据块可能分布在整个分区上。

图 7.23 UNIX 的 inode

4. 分配方法性能分析

前面所讨论的分配方法在存储效率和数据块访问时间上各有特点。这两个特性是为实现操作系统而选择合适算法时的重要依据。

在选择分配方法之前，需要确定将如何使用系统。一个主要为顺序访问的系统不应该与一个主要为随机访问的系统采用相同的方法。

不管什么类型的访问，连续分配需要访问一次就能得到磁盘块。由于能将文件的开始地址放在内存中，可以马上计算出第 i 块的磁盘地址（或下一块），并能直接读。

对于链接分配，也能将下一块的地址放在内存中，并能直接读取。对于顺序访问，这种方法还可以；但对于直接访问，对第 i 块的访问可能需要读 i 次磁盘。这一问题也说明了为什么链接分配不适用于需要直接访问的应用程序。

因此，有的系统通过使用连续分配以支持文件的直接访问，通过链接分配以支持文件的顺序访问。对于这些系统，所使用的访问类型必须在文件创建时加以说明。用于顺序访问的文件可以链接分配，但不能用于直接访问。用于直接访问的文件可以连续分配，且能支持直接访问和顺序访问，但是在创建时，必须说明其最大文件大小。在这种情况下，操作系统必须有合适的数据结构和算法以支持这两种分配方式。文件可以从一种类型转换成另一种类型：创建一个所需的类型，将原来文件的内容复制过来，删除原来文件，重新命名新文件。

索引分配更为复杂。如果索引块已在内存中，那么可以直接访问。不过，将索引块保存在内存中需要相当大的空间。如果内存空间不够，那么可能必须先读入索引块，再读入所需的数据块。对于两级索引，可能需要读两次索引块。对于一个非常巨大的文件，访问文件结束附近的块需要读入所有索引块，最后才能读入所需要的数据块。因此，索引分配的性能依赖于索引结构、文件大小，以及所需块的位置。

有的系统将连续分配和索引分配组合起来：对小文件（只有 3 或 4 块）采用连续分配；当文件大时，自动切换到索引分配。由于绝大多数文件都较小，小文件的连续分配的效率又高，所以平均性能还是不错的。

例如，SUN 公司的 UNIX 操作系统版本在 1991 年修改过，以改善文件系统分配算法的性能。性能测试表示在一个典型工作站（12-MIPS SPARCstation1）最大磁盘吞吐量使用了CPU 的 50%，产生了 1.5MB/s 的磁盘带宽。为了改善性能，SUN 作了改进，只要可能就按大小为 56KB 的簇来分配空间。（56KB 是当时 SUN 系统一次 DMA 传输的最大能力）。这种分配减低了外部碎片、寻道和延迟时间。另外，也优化了读磁盘程序以方便读这些大簇。索引节点（inode）结构没有改变。这些改进，加上使用了向前读和后释放，降低了 25%CPU 使用，大大地提高了磁盘吞吐量。

还有许多其他优化方法在使用。由于 CPU 和磁盘速度的不等，就是花费操作系统数千条指令以节省一些磁头移动都是值得的。再者，随着时间推移，这种不等程度会增加，甚至花费操作系统数十万条指令来优化磁头移动，也是值得的。

5. 空闲空间管理

因为磁盘空间有限，所以如果可能，需要将删除文件的空间用于新文件。（只写一次的光盘只允许向任何扇区写一次，因而不可能重新使用。）为了记录空闲磁盘空间，系统需要维护一个**空闲空间链表（free-space list）**。空闲空间链表记录了所有空闲磁盘空间，即未分配给文件或目录的空间。当创建文件时，搜索空闲空间链表以得到所需要的空间，并分配给新文件。这些空间会从空闲空间链表中删除。当删除文件时，其磁盘空间会增加到空闲空间表上。空闲空间链表虽然称为链表，但不一定实现为链表，这一点随着后面的讨论将会清楚。

(1). 位向量

通常，空闲空间表实现为**位图（bit map）**或**位向量（bit vector）**。每块用一位表示。如果一块为空闲，那么其位为 1；如果一块已分配，那么其位为 0。

例如，假设有一个磁盘，其块 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, 27 为空闲，其他块为已分配。那么，空闲空间位图如下：

0011110011111100011000000011100000…

这种方法的主要优点是查找磁盘上第一空闲块和 n 个连续空闲块时相对简单和高效。确实，许多计算机都有位操作指令，能有效地用于这一目的。例如，从 80386 开始的 Intel 系列和从

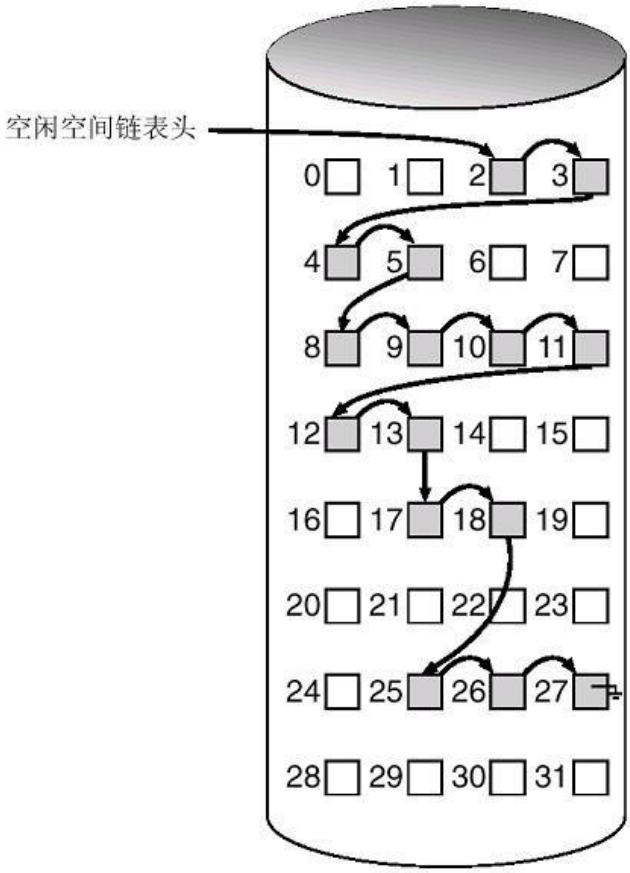
68020 开始的 Motorola 系列都有能返回一个字中第一个值为 1 的位的偏移的指令。在使用位图的系统上找到第一个空块来分配磁盘空间的一种技术是按顺序检查位图的每个字以检查其是否为 0，因为一个值为 0 的字表示其对应的所有块都已分配。再对第一个值为非 0 的字进行搜索值为 1 的位偏移，该偏移对应着第一个空闲块。该块号码的计算如下：

(值为 0 的字数)×(一个字的位数) + 第一个值为 1 的位的偏移

这里，再次看到硬件特性简化了软件功能。不过，除非整个位向量都要保存在内存中（并时而写入到磁盘用于恢复的需要），否则位向量的效率就不高。对于小磁盘，完全保存在内存中是有可能的，但对于大的计算机就不行了。对于一个每块为 512 字节，容量为 1.3G 的磁盘，可能需要 332KB 来存储位向量，以便跟踪空闲空间。虽然如果采用按合并 4 个扇区为一个簇，那么该数字会变为每个磁盘需要 83KB 的内存。一个 40G、每块为 1K 的磁盘需要超过 5MB 的空间存储位图。

(2). 链表

空闲空间管理的另一种方法是将所有空闲磁盘块用链表连接起来，并将指向第一空闲块的指针保存在磁盘的特殊位置，同时也缓存在内存中。第一块包含一个下一空闲磁盘块的指针，如此继续下去。对上一个例子（11.5.1），有一个指向块 2（第一个空闲块）的指针。块2 包含一个指向块 3 的指针，块 3 指向块 4，块 4 指向块 5，块 5 指向块 8，等等（图 7.26）。不过，这种方案的效率不高；要遍历整个表时，需要读入每一块，这需要大量的 I/O 时间。好在遍历整个表并不是一个经常操作。通常，操作系统只不过简单地需要一个空闲块以分配给一个文件，所以分配空闲表的第一块就可以了。FAT 方法将空闲块的计算结合到分配数据结构中，不再需要



另外的方法。

图7.24 采用链接方式的磁盘空闲空间链表

(3). 组

对空闲链表的一个改进是将 n 个空闲块的地址存在第一个空闲块中。这些块中的头 n – 1 个确实为空。而最后一块包含另外 n 个空闲块的地址，如此继续。大量空闲块的地址可以很块地找到，这一点有别于标准链表方法。

(4). 计数

另外一种方法是利用这样一个事实：通常，有多个连续块需要同时分配或释放，尤其是在使用连续分配和采用簇时更是如此。因此，不是记录 n 个空闲块的地址，而是可以记录第一块的地址和紧跟第一块的连续的空闲块的数量 n。这样，空闲空间表的每个条目包括磁盘地址和数

量。虽然每个条目会比原来需要更多空间，但是表的总长度会更短，这是因为连续块的数量常常大于 1。