

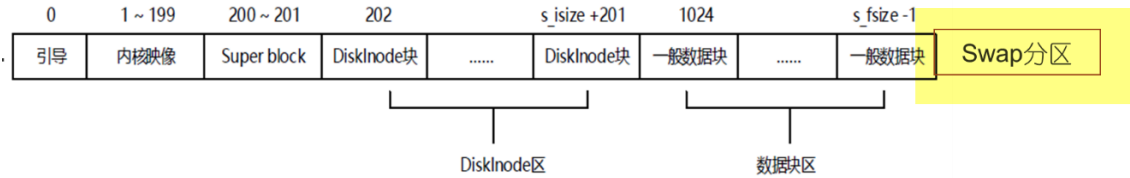
文件系统理论

同济大学操作系统课程讲义 邓蓉 2022-12-28 2024-1-1 修订

主硬盘分 2 个区域，盘交换区用来存放换出的进程图像，扩展内存空间；文件区用来存放文件和目录，永久保存数据。

- 为了提高进程图像换入、换出的速度，交换区使用连续的存储管理方式，进程图像在交换区占据连续存储单元，一次 DMA 操作可以读入、换出整个进程图像。
- 文件区一般允许使用离散存储管理方式，有利于减少碎片，提高磁盘存储空间的利用率。

下图是 Unix V6++ 的主硬盘，扇区 200 ~ s_fsize-1 是文件区。



文件的逻辑结构

文件的逻辑结构是用户看到的文件内容的组织形式，与文件内容在磁盘上的存储结构无关。

1、无结构文件，又叫流式文件。

无结构文件是一个字符序列。对流式文件的访问使用一根读写指针，指出读写起始位置在文件中的偏移量。Unix 文件系统管理的普通文件是无结构文件，内核基于读写指针为应用程序和进程管理系统提供连续字节流访问服务。典型的无结构文件包括，源程序文件，目标代码文件等等。还有好多。。应该大家平时遇到的所有文件都是流式文件，这个要再去找权威的辅证材料。

2、有结构文件。

文件是记录集合。每条记录描述一个实体，是赛博空间中的一个数字宝贝。记录的长度可以是定长的，也可以是变长的。记录中包含的字段数量可以是固定的，也可以各不相同。按记录的组织方式，有结构的文件主要有以下 4 种：

2.1 顺序文件 (Sequential File)


文件是记录集合。记录可以有两种组织形式。

第一种是串结构。记录按其插入文件的时刻追加写在文件末尾。各记录的排序与关键字无关。第二种是顺序结构。记录按主关键字排序。

定长记录顺序文件 和 变长记录顺序文件。

图 a，定长记录文件，每条记录 L 字节。

图 b, 变长记录文件, 需要登记每条记录的长度, 记录 R_0 的长度是 L_0 ……记录 R_i 的长度是 L_i ……。



记录文件 顺序文件 (basic)

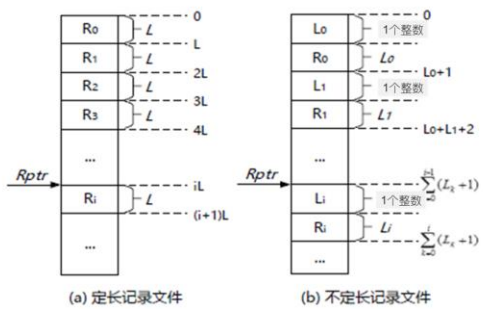
从以下2个方面考察逻辑结构的优劣:

- 1、顺序访问
- 2、随机访问

定长记录文件和不定长记录文件都支持顺序访问

定长记录文件支持随机访问:
 n 记录, 文件中的偏移量是: $n * L$ (字节)

不定长记录文件不支持随机访问:
 求取 n 记录在文件中的偏移量, 需要遍历列表 L_i 。
 文件是存放在磁盘上的, 这会非常耗时。



(a) 定长记录文件 (b) 不定长记录文件

图 7.2: 定长和变长记录文件

操作系统
电信学院计算机系 邓蓉
4

适合顺序文件的应用场合:

顺序文件, 适合大数据批处理任务。这是因为顺序文件不存在索引结构的使用和维护开销, 全表扫描时效率最高。这是目前很多 NoSQL 数据库使用顺序文件存放大数据集的一个重要原因。

不适合顺序文件的应用场合:

对于需要频繁针对单条记录执行查询、更新、插入或删除的应用场合, 顺序文件不合适。

- 串结构, 按关键字检索需要遍历整个文件。非常耗时。
- 顺序结构, 仅支持一个关键字。按主关键字检索可以用折半查找, 比较快。按其它关键字检索, 依然需要遍历整个文件。

增加删除记录时, 怎样保持顺序结构文件按关键字排序?

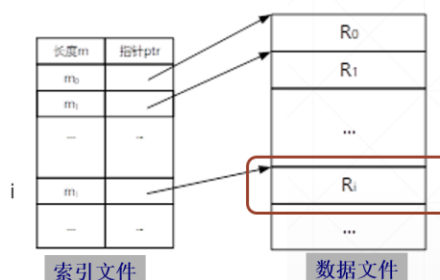
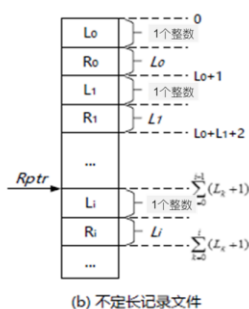
为顺序文件配置一个 **运行记录文件** (Log File) 或称 **为事务文件** (Transaction File), 把试图增加、删除或修改的信息记录于其中, 规定每隔一定时间, 例如 4 小时, 将运行记录文件与原来的主文件加以合并, 产生一个按关键字排序的新文件。HDFS/GFS 日志文件和主文件树合并, 有用到这个思想。

2.2 索引文件

将不定长记录文件中, 每条记录的长度字段抽出来, 组成一张索引表 (一个索引文件)。配上原先的数据文件, 就是索引结构的文件啦。



变长记录文件如何支持随机访问？



(b) 变长记录文件和它的索引文件

数据文件登记原始数据
索引文件登记每条记录的长度和它在数据文件中的偏移量

索引结构的变长记录文件支持随机访问：

访问第 i 条记录的方法：索引文件是定长数组。访问索引文件中的第 i 个元素，可以得到数据文件中第 i 条记录的长度和起始偏移量。

计算消耗的时间与 i 值无关，所以索引文件结构支持随机访问。(随机访问一般要求：给定下标， $O(1)$ 时间复杂度在数据文件中定位所有记录)

现在我们得到了一个支持随机搜索的变长记录文件。

我们以这个结构为基础，为记录文件加索引，支持按关键字搜索记录。

记录，以其索引表下标为 ID。

我们为每个键（关键字）建一张索引表。索引表 2 个字段，登记每条记录的 ID 和 对应关键字的取值。索引表按关键字的值排序，支持折半查找，查找效率很高。

搜索的加速，每个键（关键字）一张索引表

按key排列

键值	记录ID
Alice	
Alice	
.....	
Tom	

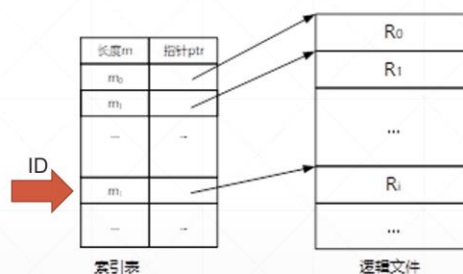
关键字name的表

按key排列

键值	记录ID

第2个关键字的表

.....



支持随机搜索的原始记录文件



插入新记录很简单：首先将原始记录添加在数据文件尾部，之后，在索引表尾部同步增加新记录的长度和指针，最后，新记录按键值插入所有关键字表，保持其有序。

这种结构的好处在于，仅搜索的话，想支持多少关键字都可以。
缺点是，插入删除记录，需要同步维护关键字的索引表。有点烦。

更好的结构，每个键1棵B+树

name的B+树
其它字段的B+树
.....

索引表

逻辑文件

记录的插入操作：
记录 **append** 在原始数据文件尾部
索引结构也存在文件尾部
每个关键字**1棵B+树**，**B+树的节点**
里装记录**ID**

操作系统

电信学院计算机系 邓蓉

7

查找会更快！

为记录文件建索引，DBMS 有深入研究，这里不再赘述。

在一般的教科书上，还有 2 种理论上的文件逻辑结构，索引顺序文件、直接文件或散列文件。这里一并列出。

索引顺序文件将顺序文件中的所有记录分为若干个组，在索引表中为每组中的第一个记录，建立一个索引项，其中含有记录的键值和指向该记录的指针，如图。查找一条记录时，首先通过索引表找到其所在的组，然后在该组中进行顺序查找。有未尽细节，篇幅所限，不再赘述。

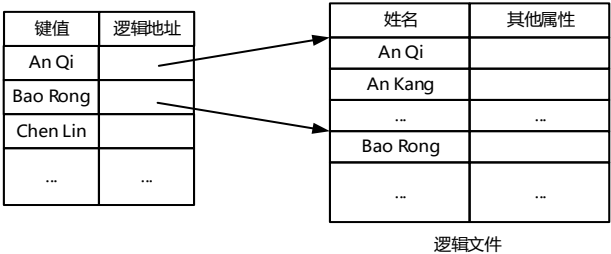


图 7.4：索引顺序文件

直接文件或散列文件。为主文件配 hash 表。在主文件中，记录是没有序的。查找记录时，根据关键字计算 hash 值确定记录在主文件中的起始偏移量。散列文件有很高的存取速度，

但是会有 hash 冲突。更进一步，记录根据其关键字之 hash 值，只能存放在主文件固定的位置。这是直接文件，hash 表都省了。以计算代存储。先进的思路。

总结：文件的逻辑结构研究的是怎样在主文件中快速定位目标记录。常用的技术包括，（1）建索引（2）记录，按键值的 hash 值，存放在主文件固定的若干位置。

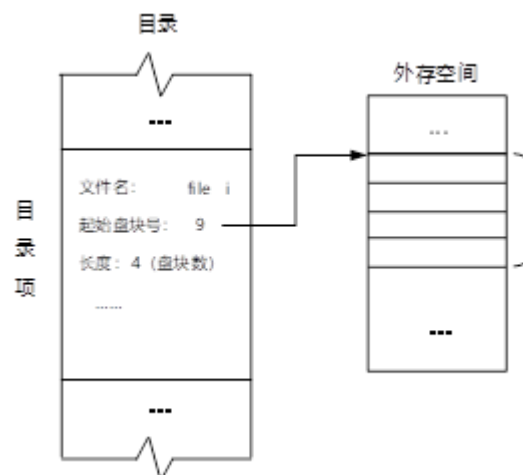
文件的物理结构

下面研究文件的物理结构，为文件分配磁盘存储空间，将数据存放在磁盘合适的位置。有 2 个优化目标：（1）有效利用外存空间。（2）提高文件顺序访问或随机访问速度。

文件在磁盘上的布局会影响文件的读写效率，并进而影响整个文件系统的吞吐率。常见的物理结构有 3 种，连续文件结构、链接文件结构和索引文件结构。

一、连续文件结构

为文件分配连续磁盘数据块。目录项中地址相关的字段只需要包括起始盘块号和文件长度，如图：



连续文件的地址映射公式：

访问逻辑块 n ，需要访问的物理块号 = 起始盘块号 + n 。

文件打开后，目录项是在内存里的（请回顾 Unix V6 系统中的内存 Inode）。上述地址映射可以在内存中完成，无需磁盘 IO，开销近似为 0，与 n 无关。所以，连续文件结构支持随机访问。

此外，连续文件顺序访问性能最优。这是因为逻辑块在磁盘上顺序存放。顺序访问时，无论起始读写位置，无论一次需要读写多少字节，一次 DMA 操作可以搞定 & 磁盘定位耗时最小（寻道时间+旋转延迟）。只需要定位读写的起始盘块。。一次磁头机械定位操作。

连续文件结构的缺点：（1）会产生外部碎片，降低磁盘空间的利用率。（2）为文件分配磁盘空间之前需要预知文件长度。使用过程中，如果文件长度发生变化，管理起来会很麻

烦。

连续文件结构使用的场合：文件内容写入一次便不再会更改。比如归档文件，大数据集文件。另外，如果磁盘好大好大，我们不在乎碎片，那浪费就浪费一点吧（这是在以空间换时间）。这种物理结构性能超级好，不是嘛。

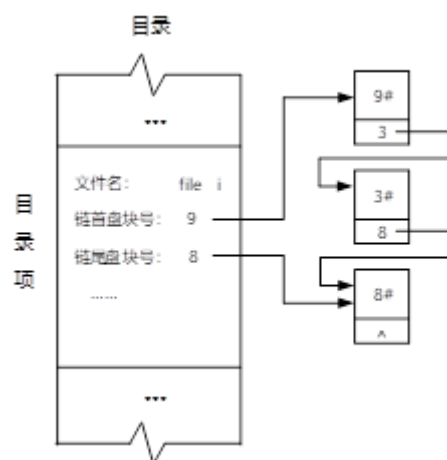
二、链接文件结构

文件内容不连续存放。这是离散的文件结构。有**两种链接结构**。

1、隐式链接

为文件分配离散磁盘数据块。用**数据块的最后 4 个字节**存放分配给**下个逻辑块的物理块号**。

目录项中地址相关的有 3 个字段，链首盘块号，链尾盘块号 和 文件长度，如图是一个有 3 个逻辑块的链接文件，存放在 9#，3# 和 8#物理块。



目录项中增加链尾盘块号字段，是为了方便在**文件尾部追加写**。

隐式链接结构读写性能很差。

- 顺序读写时，每个逻辑块都有定位开销。
- **不支持随机读写。**这是因为需要 IO 前一个数据块，才能够得到下个逻辑块的数据块号。比如，读 0#逻辑块，地址映射开销是 0；读 1#逻辑块，需要先读出 0#逻辑块，才能得到存放 1#逻辑块的物理块号，地址映射的开销是 1 次 IO；一般而言，访问 i#逻辑块，需要读出前面所有数据块，地址映射的开销是 i 次 IO。地址映射的耗时，与被访问逻辑块号 n 线性相关，所以链接结构不支持随机读写。

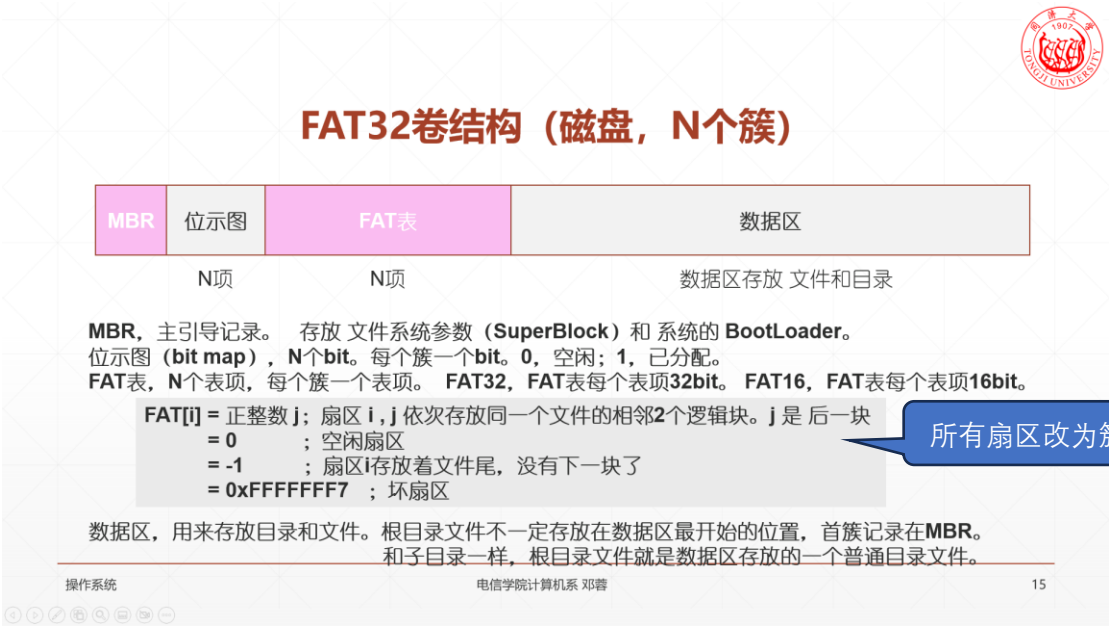
此外，链接结构文件容易出错，只要其中任何一个数据块链接指针出错，整根链就会断掉。

还有，磁盘数据块的**长度通常是 2^n 字节**。用最后 4 个字节存放链接指针，每个数据块可以保存的数据量就不是 2^n 了。地址映射时需要根据文件偏移量计算逻辑块号。如果每个数据块保存的字节数是 2^n ，这个计算是移位操作，甚至是取高位 bit 这种可以在一个 CPU cycle 内完成的操作，非常快。隐式链接结构无法利用这个优势，地址映射需要执行除法操作，太慢了。

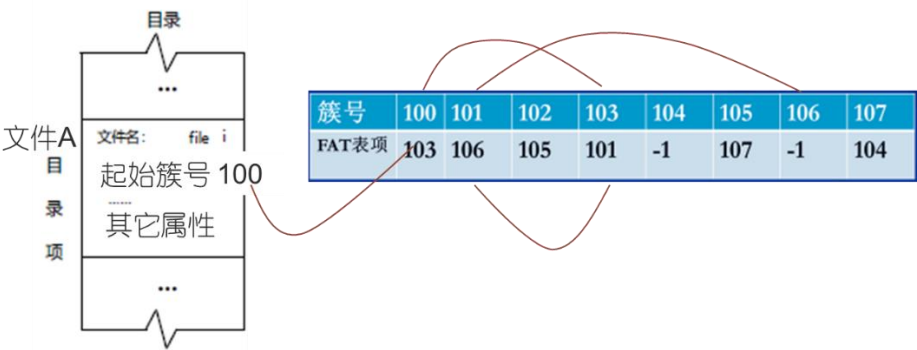
总之，隐式链接是历史发展过程中出现的一种理论模型。它首次实现了离散文件结构，适用于容量极小或单位存储空间特别昂贵的存储单元。现代计算机系统，磁盘好大好大，这种物理结构就显得不太适用了。

2、显式链接

显式链接是微软 FAT 文件系统使用的物理结构。下图是一个 FAT 文件卷。
每个 FAT 文件卷有一个位示图和一个 FAT 表。



每个文件的目录项, 记录文件首簇。多一个字段, 记录链尾簇也可以。这张图没有画。
根目录文件的首簇, 是 MBR 中的一个字段。
其它文件, 首簇在目录项里, 目录项存放在父目录文件中。



如图 FAT 表片段。文件 A, 首簇 100, 之后是 103, 101 和 106。

装载 FAT 文件卷时, 将位示图 和 FAT 表一并装入内存。文件卷卸载时, 写回磁盘。

访问文件之前, 需要打开文件, 将目录项装入内存。之后借助 FAT 表, 计算得出目标逻辑块所在的簇号。举例说明。假设某 FAT 文件卷每个簇 512 字节, 进程需要写 1000#字节, 地址映射过程如下:

- 1、计算逻辑块号 n (1) 和 块内偏移量 $offset$ (488)
- 2、 $next =$ 起始簇号;
循环 n 次
 $next = FAT[next]$
- 3、借助缓存, 写物理块 $next$ 中的第 $offset\#$ 字节

地址映射过程的耗时与目标逻辑块号成正比, 严格说来 FAT 表并不支持随机访问。但是, 查找 FAT 表可以在内存中完成, 不涉及 IO 操作, 所以性能还是可以的。

链接文件长度增加时比较方便。为新增内容分配新数据块, 把链接信息加入 FAT 表即可。这是它最大的优点之一。

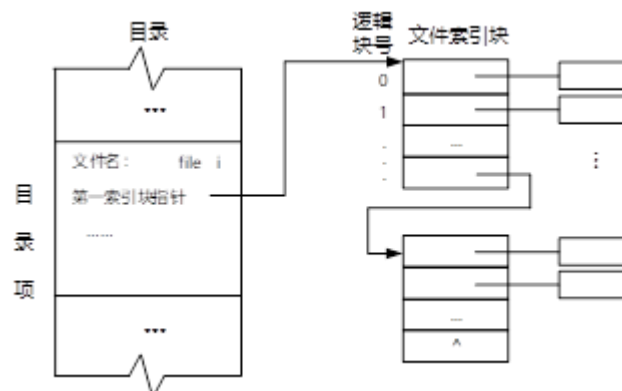
另外, FAT 表和位示图是文件系统的重要元数据。失效将导致文件系统部分或全部不可用。为了保证文件系统的安全性, 通常设置有第 2 位示图和第 2 FAT 表, 存放在磁盘的另一端。

三、索引文件结构

链接分配, 尤其是 FAT 显式链接技术, 解决了连续分配的外部碎片和文件大小管理问题。但是仍然有 3 个亟待解决的问题。(1) FAT 表内存空间消耗问题。FAT 表包含所有文件的链接信息。未使用的文件, 链接信息没必要装入内存 (2) 地址映射的时间开销。地址映射过程需要从起始簇号开始遍历整个文件的链接信息, 太耗时 (3) FAT 表需要为每个簇分配一个单元, 大容量磁盘, FAT 表需占用较大磁盘空间和内存空间。

改进之, 索引文件结构为每个文件设置独用的索引结构, 用来存放该文件的链接信息。使用中的文件, 索引块读入主存。其它文件的索引块不需要使用, 没有必要为其分配内存空间,

下图是单级索引结构: 索引块是磁盘上的数据块, 专门用来存放分配给文件的物理块号。下图所示文件 $file\ i$ 有 2 个索引块, 0#索引块 512 字节, 可以存放 128 个整数, 前 127 个整数是分配给 0~126#逻辑块的物理块号, 最后一个整数是分配给 1#索引块的物理块号。1#索引块可以再存放 127 个物理块号。。。文件长度增加时, 只需在索引结构的尾部追加索引块, 非常方便。

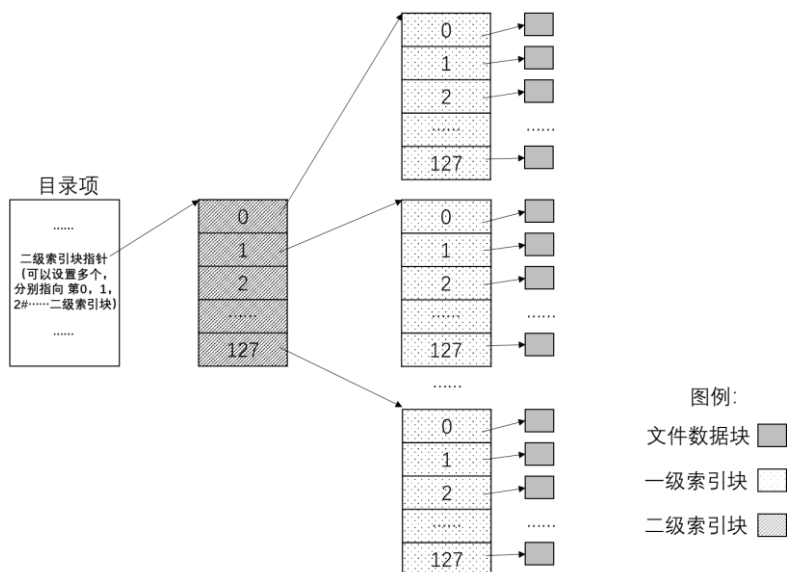


单级索引文件结构可以在一定程序上支持随机访问。访问逻辑块 num , 地址映射方法如

下:

- 1、 $\text{num}/127$ ，商是索引块号 n ，余数是索引块内下标 m 。
- 2、 next = 第一索引块指针；
循环 n 次
 next = next 指向的索引块中，第 127 个整数（下标是 126）。
- 3、分配给逻辑块 num 的物理块号 = next 指向的索引块中，第 m 个整数。

多级索引结构为索引块建索引，可以是二级索引，也可以是多级索引。多级索引可以支持容量很大的文件，为其提供性能优良的随机访问能力。如图，二级索引结构的文件。



目录项使用一个二级索引块指针，文件长度可以达到 $128 \times 128 \times 512$ 字节。

多级索引结构**支持随机访问**。不考虑缓存命中，访问任何字节，需要 3 次磁盘 IO，一次读入二级索引块，一次读入一级索引块，最后一次读入文件数据块。

磁盘高速缓存可以减小文件读写需要执行的 IO 次数。缓存竞争不激烈的话，二级索引块永远缓存命中，一级索引块读入内存后，可以满足 128 个逻辑块的地址映射需求。效果合在一起，多级索引结构地址映射不会太耗时。

回到上张图，二级索引结构。

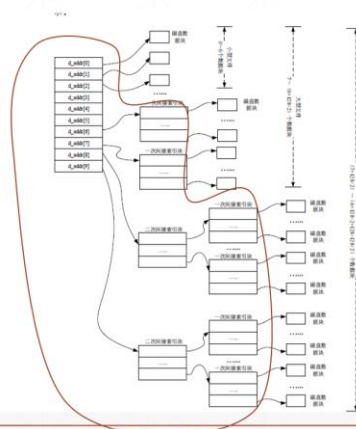
- 考虑只有一块数据的小文件。该文件需要消耗 3 个数据块，一块放文件内容，一块放一级索引，一块放二级索引。**小文件，索引结构消耗了过多的磁盘空间。**
- 无论文件大小。访问任何字节需要 3 次磁盘 IO。二级索引结构支持随机访问，访问任何字节开销一摸一样，但是结果是同样的慢。对小文件，访问每个数据块 3 次磁盘 IO 的开销是不可忍受的。

为了改进多级文件系统的性能，Unix V6 设计了**混合索引结构**。这是我们已经牢固掌握的内容。



索引文件

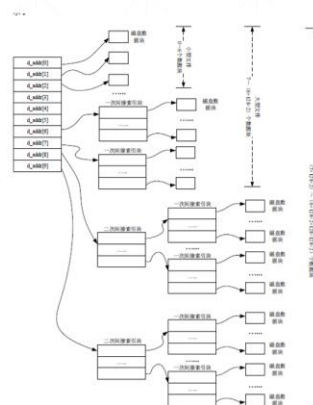
- 文件数据 和 地址指针 分开存放
- 索引结构的文件有两种类型：
 - 第一种，Unix风格的混合索引结构。
 - 每个文件有一个索引结构（很多书叫它索引表）。里面存放地址指针。
 - 除了索引结构，分配给文件的其它物理块全部用来存放文件数据。



327

这也不能算是缺点。能做到这样已经很好了，工程问题有严格的约束条件，还能要求怎样？

- 这种结构的缺点是
 - 读inode，读索引块，都需要IO磁盘。
 - 例，Unix V6++系统
 - 目录搜索，确定 inode号之后，需要IO一次磁盘，读DiskInode。
 - 访问文件时，地址映射也要IO磁盘。考虑地址映射的IO次数，
 - 读写 0~5#逻辑块，0
 - 后续256*2个逻辑块，1
 - 其它逻辑块，2
 - 有磁盘高速缓存池，考虑到缓存命中是大概率事件，混合索引结构 性能还是不错的。
- 优点
 - 支持随机访问。访问文件的不同位置，地址映射只有3种耗时。



327

混合索引结构的优点：访问小文件，地址映射不需要IO的。。这就克服了多级索引结构，地址映射IO开销大的缺点。此外，小文件的前6个链接信息直接登记在DiskInode里，没有索引块的额外存储开销。

缺点：文件内部插入、删除数据块，索引结构改动非常大。

空闲资源管理

一个存储设备可以整体用于文件系统，也可以细分。比如，一个磁盘可以划分成2个分区，每个分区有单独的文件系统。

包含文件系统的分区叫做卷。卷可以是磁盘的一部分，也可以是整张磁盘，还可以是多个磁

盘组成的 RAID（磁盘阵列）。

在每个卷中，文件的 FCB 和文件内容是分开存放的。Windows 文件系统，文件的 FCB 存放在父目录文件里，文件内容存在分配给自己的数据块里。Unix 文件系统，文件的 FCB 存放在 Disklnode 和父目录文件里，文件内容存在分配给自己的数据块里。

卷，物理存储空间分成若干数据块。数据块大小相等，是存储空间分配 和 IO 的基本单位。有 4 种常见的空闲资源管理技术。

一、空闲表法

适用于连续分配方式。系统设置空闲区表，登记每个空闲区的起始数据块号和长度。如图

序号	起始盘块号	空闲区的长度
1	20	10
2	40	5
.....

空闲区的分配与内存的动态分配类似，可以采用首次适应、最佳适应等算法为文件分配连续空间。

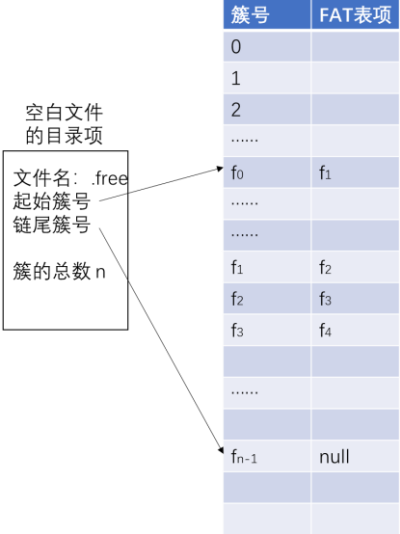
删除文件时，需要将文件原先占用的空间与相邻空闲区合并。

二、空闲链表法

有两种形式

1、空闲区链。将空闲区表中的元素，拉成一根链。与空闲表法相比，好处是（1）空闲区插入和删除比较方便，不会导致已有空闲区移位（2）空闲区数量不必受到静态数组容量的限制。

2、空闲盘块链。空闲盘块链中的元素是尺寸相等的数据块。典型的应用是早期 FAT 文件系统使用的空白文件。系统利用 FAT 表将所有空闲盘块串成一个空白文件。如图。



空白文件是根目录下的隐藏文件.free。首簇是下一个分配的空闲数据块，新回收的空闲块插队尾。

空闲盘块分配操作:

```
alloc = 起始簇号;  
起始簇号 = FAT[alloc];  
簇的总数--;  
return alloc;
```

空闲盘块回收操作 (回收 num):

```
FAT[链尾簇号] = num;  
FAT[num] = null;  
链尾簇号 = num;  
簇的总数++;  
return;
```

从程序实现的角度来看, 非常机智优雅, 不是嘛?

但是, 这个算法没有考虑到硬盘的机械特性。我们很难为文件分配连续物理块。

此外, 使用这种方法会导致 FAT 表好长, 存储空间消耗过大。

三、位示图法

用一个 bit 表示数据块的分配情况, 0 空闲, 1 已分配。

位示图是一个整数数组。假设整数字长 n , 所有编号从 0 开始。位示图中 bit 编号与数据块号的对应关系如下:

i #字, j #bit 对应的数据块号 $= i * n + j$;

数据块号 b , 对应位示图中如下 bit:

字号 $= b \text{ 整除 } n$;

位号 $= b \bmod n$;

通常, n 是 2 的幂次。所以, 计算非常省时间。

用位示图管理空闲数据块, 好处在于便于为文件分配连续数据块。是一个对硬盘硬件特征敏感的优秀的数据结构。此外, 用一个 bit 管理一个数据块, 存储空间消耗适中。

四、成组链接法

Unix V6++ 管理空闲数据块的方法。用空闲数据块本身登记后续空闲块的号码。好处是, 没有必要单独设置数据结构管理磁盘空闲资源, 这会节约文件系统元数据消耗的存储空间。缺点是, 很难为文件分配连续物理块, 文件读写 IO 性能低下。