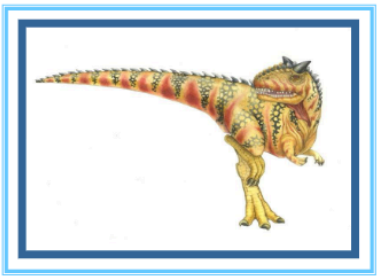
第 14 章：文件系统实现



第 14 章：实现文件系统

- 文件系统结构 文件系统实现 目录实现 分配方法 可用空间管理

目标

- 描述实现文件系统和目录结构的详细信息
- 讨论区块分配和自由区块算法和权衡

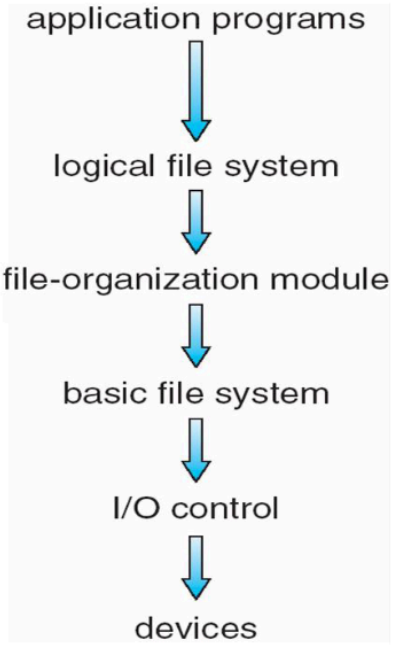
文件系统结构

- 磁盘提供了维护文件系统的大部分辅助存储。磁盘的两个特性使其便于此用途：
-
  - 磁盘可以就地重写;可以从磁盘读取块，修改块，然后将其写回磁盘上的同一位置
  - 磁盘可以直接访问它包含的任何信息块。因此，按顺序或随机访问任何文件都很简单，从一个文件切换到另一个文件只需要移动读写头并等待磁盘旋转 – 在第 11 章中讨论
- 为了提高 I/O 效率，内存和磁盘之间的 I/O 传输以块为单位进行。每个区块包含一个或多个扇区。扇区大小从 32 字节到 4096 字节 （4KB） 不等，通常为 512 字节 （0.5KB）

文件系统结构 （续）

- 文件结构
  - 逻辑存储单元 相关信息的集合
  -
- 文件系统驻留在辅助存储 – 硬盘驱动器或磁盘上
  - 它允许轻松存储、定位和检索数据，从而提供对磁盘的高效便捷访问
  - 它提供用户界面：文件和文件属性、对文件的操作、用于组织文件的目录
  - 它提供用于将逻辑文件系统映射到物理辅助存储设备的数据结构和算法
- 文件系统分为不同的层

分层文件系统



文件系统层

- I/O 控制和设备驱动程序在 I/O 控制层管理 I/O 设备
  - 它由设备驱动程序和中断处理程序组成，用于在内存和磁盘之间传输信息给定的命令，如“读取驱动器 1、柱面 72、磁道 2、扇区 10”（磁盘物理地址），到内存位置 1060”，将低级硬件特定命令输出到硬件控制器
- Basic 文件系统向相应的设备驱动程序发出通用命令，以读取和写入磁盘上的物理块
  - 给定诸如“检索块 123”之类的命令，将其转换为特定的设备驱动程序它管理保存各种文件系统、目录和数据块的内存缓冲区和缓存。（分配、释放、替换）
  - 缓冲区保存传输中的数据。在传输磁盘块之前，会分配内存缓冲区中的块。缓存保存常用的文件系统元数据以提高性能
- 文件组织模块了解文件及其逻辑块以及物理块
  - 将逻辑块 #（地址）转换为物理块 #，将其传递给基本文件系统进行传输管理可用磁盘空间、磁盘块分配

文件系统层（续）

- 逻辑文件系统管理元数据信息
  - 元数据包括除实际数据之外的所有文件系统结构，即文件的内容 它管理目录结构以提供文件组织模块所需的信息。通过维护文件控制块将文件名转换为文件编号或文件句柄、位置文件控制块 （FCB）（在 Unix 文件系统中称为 inode）包含有关文件的所有信息，包括文件内容的所有权、权限和位置（在磁盘上）
  - 它还负责文件保护
- 分层有助于降低复杂性和冗余，但会增加开销并降低性能
- 目前使用许多文件系统，并且大多数操作系统都支持多个文件系统
  - 每个都有自己的格式 - CD-ROM 是 ISO 9660;Unix 具有基于 FFS 的 UFS（Unix 文件系统）;Windows 有 FAT、FAT32、NTFS（或 Window NT 文件系统）以及软盘、CD、DVD 蓝光，Linux 有 40 多种文件系统，有扩展文件系统 ext2 和 ext3;加上分布式文件系统等。新的应用程序仍在推出 – ZFS、GoogleFS、Oracle ASM、FUSE

文件系统实现

多个磁盘和内存中结构用于实现文件系统。

磁盘结构，它可能包含有关如何引导存储在磁盘上的操作系统、块的总数、可用块的数量和位置、目录结构和单个文件的信息

内存中信息用于文件系统管理和通过缓存提高性能。数据在挂载时加载，在文件系统操作期间更新，并在卸载时丢弃。

磁盘上的文件系统结构

- 启动控制块 （每个卷） 包含系统从该卷启动操作系统所需的信息
  - 如果磁盘不包含 OS，则此块可以是 empty 通常是卷的第一个块。在 UFS 中，它称为引导块。在 NTFS 中，它是分区引导扇区
- 卷控制块 （每个卷） 包含卷 （或分区） 详细信息
  - 总块数 #、可用块数 #、块大小、可用块数和指针数、可用 FCB 计数和指针在 UFS 中，这称为超级块。在 NTFS 中，它存储在主文件表中
- 目录结构（每个文件系统）用于组织文件
  - 在 UFS 中，这包括文件名和关联的 inode 编号（Unix 中的 FCB）。在 NTFS 中，它存储在主文件表中
- 每个文件的文件控制块 （FCB） 包含有关文件的许多详细信息
  - 它具有与目录条目关联的唯一标识符号。在 UFS 中，inode 编号、权限、大小、日期
  - NFTS 使用关系数据库结构存储在主文件表中，每个文件一行

典型的文件控制块

file permissions
file dates (create, access, write)
file owner, group, ACL
file size
file data blocks or pointers to file data blocks

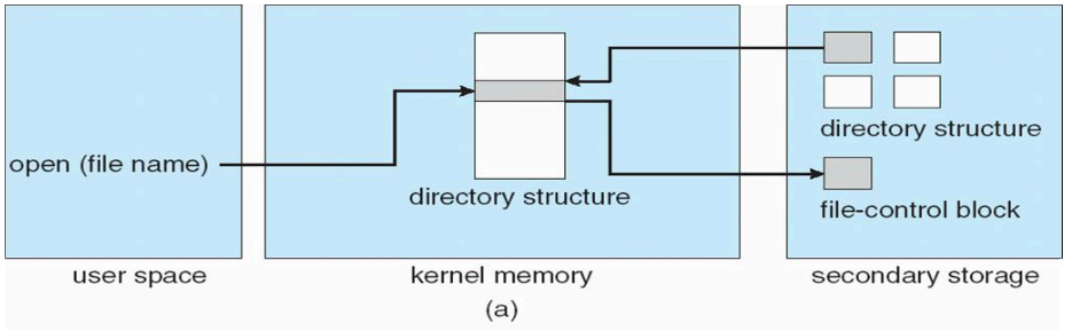
内存中文件系统结构

内存中信息用于文件系统管理和通过缓存提高性能。数据在挂载时加载，在文件系统操作期间更新，并在卸载时丢弃

- 内存中挂载表包含有关每个挂载卷的信息 内存中的 directory-structure 高速缓存保存最近访问的目录的目录信息。
- 系统范围的 open-file 表包含每个打开文件的 FCB 副本，用于查找文件以及其他信息
- 每进程打开文件表包含指向系统范围打开文件故事中相应条目的指针，以及其他信息，例如每进程文件保护和访问权限。
- 缓冲区在从磁盘读取或写入磁盘时保存文件系统块

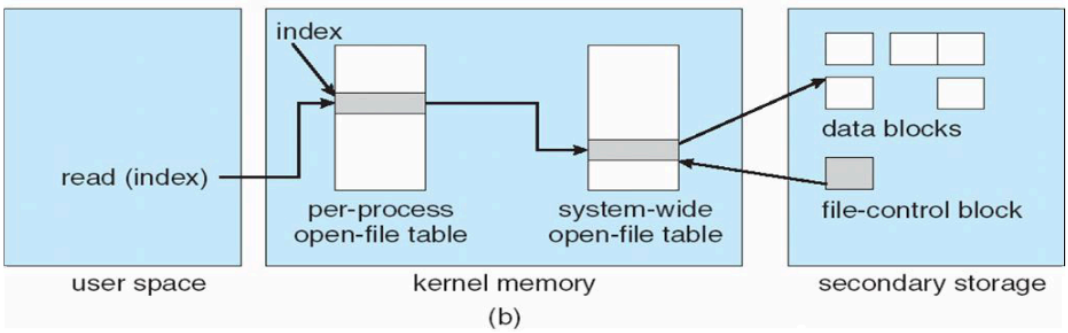
内存中文件系统结构

- 下图是指打开一个文件
  - open（）操作将文件名传递给逻辑文件系统。如果该文件已被另一个进程使用，则仅创建每个进程的 open-file 表条目，该条目指向系统范围的 open-file 表中此文件的相应条目。
  - 如果没有，它会搜索目录结构（其中一部分可能缓存在内存中）。找到文件后，将分别在系统范围的 open-file table 和 per-process open-file 表中创建一个条目。FCB 被复制到内存中的系统范围的 open-file 表中，以供后续访问
  - 系统范围的打开文件表不仅存储 FCB，还跟踪已打开的进程数，因此正在使用此文件 – 文件计数
  - 每个进程的 open-file 表中的其他字段可能包括指向文件中当前位置的指针（用于下一个 read（） 或 write（） 操作）和打开文件的访问模式。



内存中文件系统结构

- 下图是指读取文件
  - open（）调用返回一个指针，该指针指向每个进程的 open-file 表中的相应条目。换句话说，open（）操作在每个进程和系统范围的打开文件表中都会创建相应的条目，后续操作使用此指针来定位文件内容，而无需访问目录结构——这加快了对文件的后续操作
  - 然后，所有文件操作都通过此指针执行。UNIX 系统将其称为文件描述符;Windows 将其称为文件句柄。
  - 来自 read（） 的数据最终复制到指定的用户进程内存地址（进程地址空间的一部分）



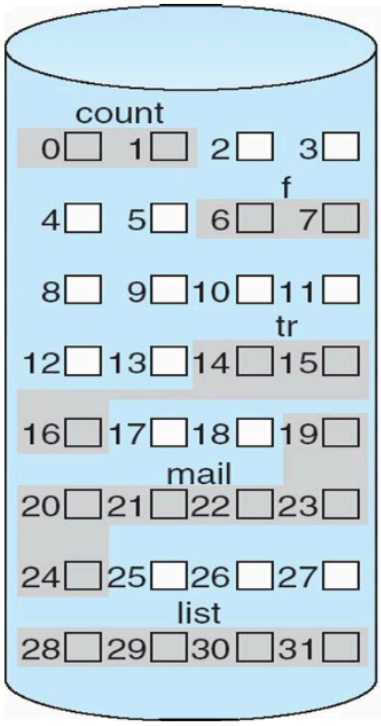
目录实施

- directory-allocation 和 directory management 算法的选择会显著影响文件系统的效率、性能和可靠性。
- 带有指向数据块的指针的文件名的线性列表
  - 编程简单 执行耗时 线性列表的主要缺点是查找文件需要线性搜索时间。
  -
  -
    - 在内存中缓存常用的目录信息 可以通过链表或使用 B+ 树按字母顺序排序
- Hash Table – 具有哈希数据结构的线性列表
  - 减少目录搜索时间 冲突 – 两个文件名哈希到同一位置的情况 仅当条目是固定大小或使用链式溢出方法时才好 哈希表的主要困难是其通常固定的大小以及哈希函数对该大小的依赖性。
  -
  -

分配方法 - 连续

- 分配方法是指如何为文件分配磁盘块，以便有效地利用磁盘空间，并且可以快速访问文件。
- 分配磁盘空间有三种主要方法，它们被广泛使用：连续、链接和索引。
- 连续分配 – 每个文件占用一组连续的磁盘块
  - 大多数情况下的最佳性能 – 轻松支持顺序和直接访问 简单 – 只需要起始位置（块 #）和长度（块数） 为新文件寻找空间以及文件大小增长时出现问题
  -
  - 这也是前面讨论的一个动态存储分配问题，它涉及如何从一个免费的可变大小的孔列表中满足大小为 n（变量）的请求 - 存在外部碎片 最佳和首次拟合是常见的策略，并且被证明比最差拟合更有效。
  -
  - 对于大型磁盘，压缩成本特别高，可能需要数小时。某些系统要求仅在卸载文件系统的情况下离线时进行压缩
  - 对于在创建文件时必须知道文件大小的文件，这是首选 - 高估会导致大量内部碎片

磁盘空间的连续分配

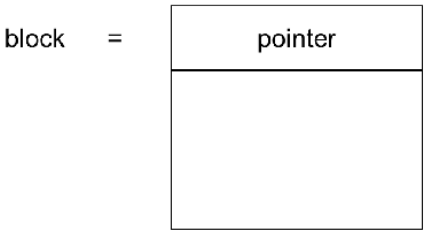


directory

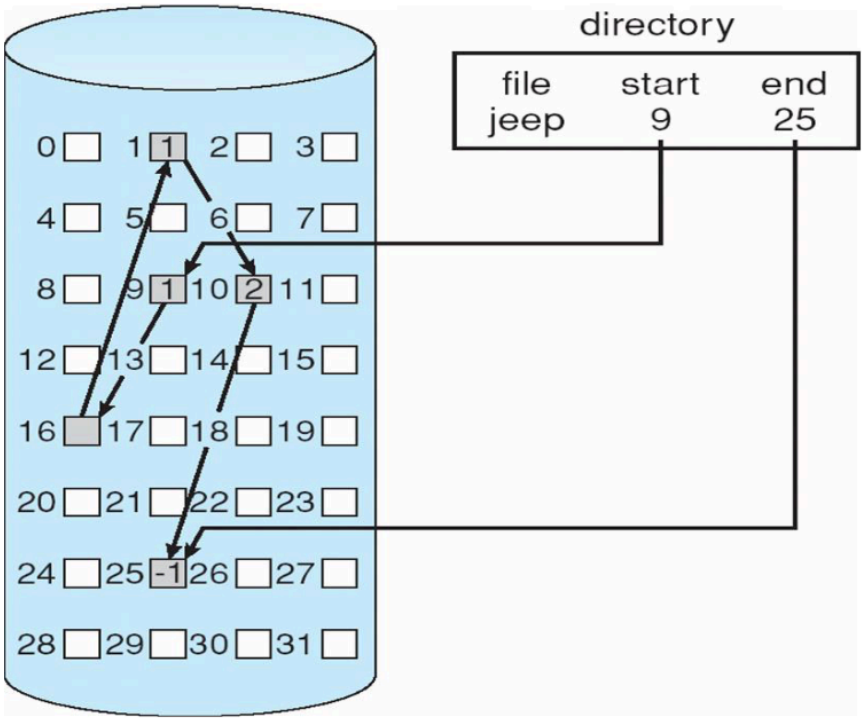
file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

分配方法 - 链接

- 链接分配 – 每个文件由一个块的链接列表组成
  - 每个文件都是磁盘块的链接列表，这些块可能分散在磁盘上的任何位置该目录包含指向文件的第一个和最后一个块的指针
  - 文件以 null 指针（列表结尾指针值）结束每个块都包含指向下一个块的指针
  - 无需压缩，也无需外部碎片只要有空闲块可用，文件就可以继续增长
  - 支持直接访问文件效率低下，仅适用于顺序访问指针所需的额外磁盘空间。如果指针需要 512 字节块中的 4 个字节，则 0.78% 的磁盘空间用于指针。
  - 可靠性可能是一个问题;例如，如果指针丢失或损坏，会发生什么情况。



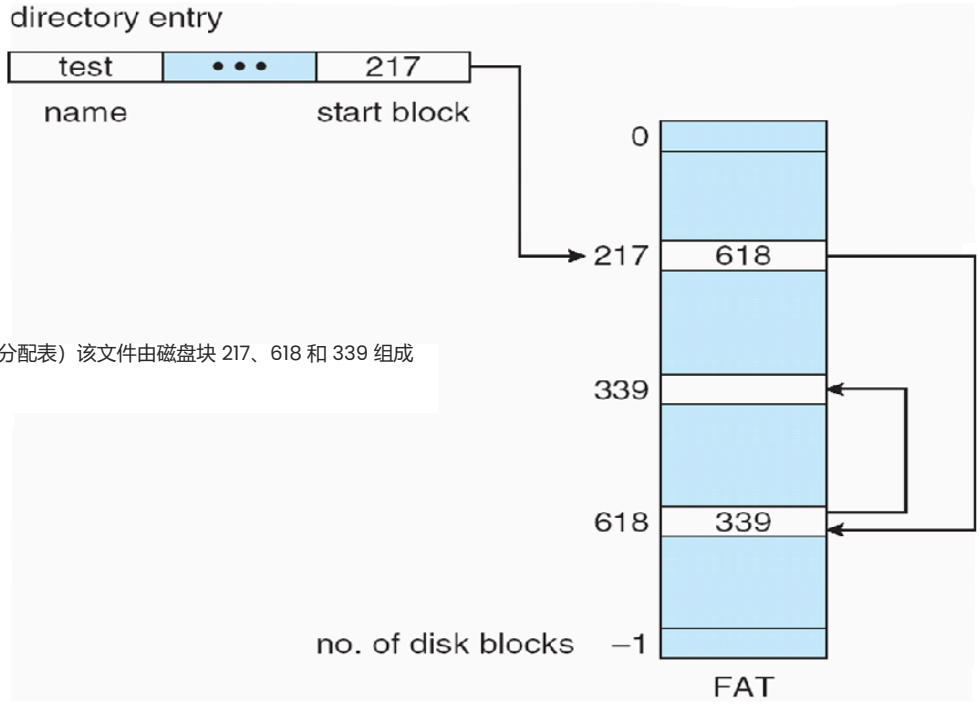
链接分配



分配方法 - FAT

- FAT （File Allocation Table） – 链接分配的重要变体
  - MS-DOS 使用了这种简单但有效的磁盘空间分配方法：卷开头的磁盘部分被留出来包含一个名为 FAT 的表。该表的每个磁盘块都有一个条目，并按块编号进行索引
  - FAT 的使用方式与链表大致相同。目录条目包含文件的第一个块的块号。由该块号索引的表条目包含文件中下一个块的块号。这种情况一直持续到它到达最后一个块，该块具有特殊的 end-of-file 值作为表条目
  - 未使用的块由表项值 0 表示。为文件分配新块是一个简单的问题，只需找到第一个 0 值表条目即可。
  - FAT 可以缓存。随机（直接）访问时间得到了改进，因为磁盘头可以通过读取 FAT 中的信息来找到任何块的位置，而不是在链接分配方案中存储的块之间移动。

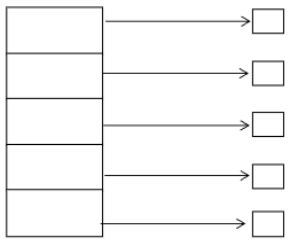
文件分配表



- FAT 示例（文件分配表）该文件由磁盘块 217、618 和 339 组成

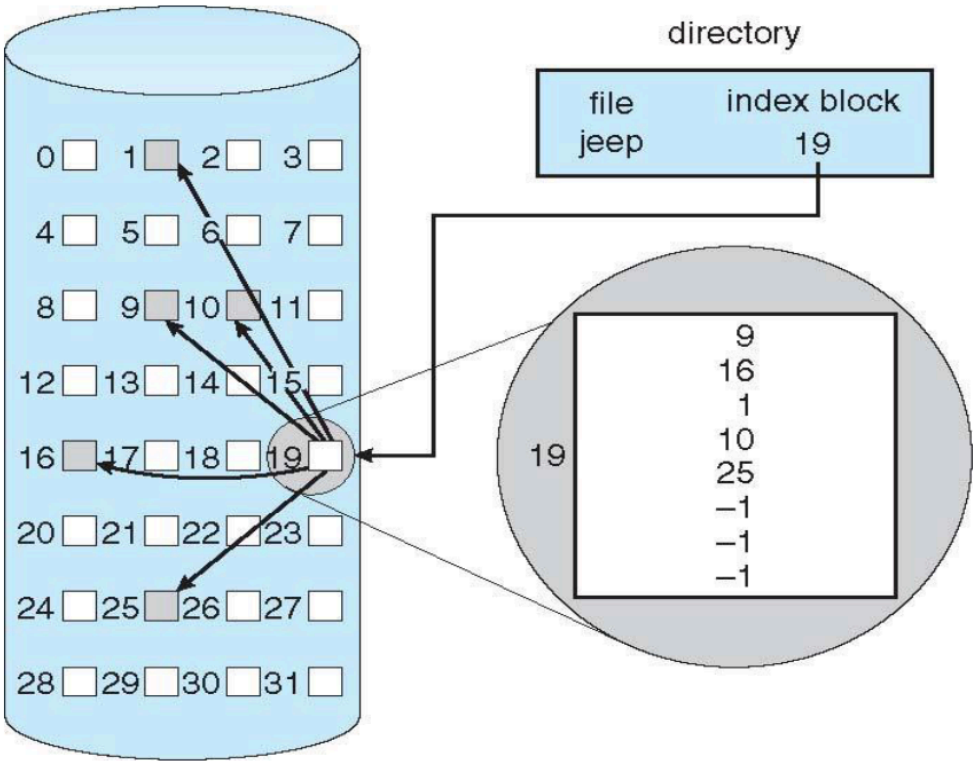
分配方法 - 索引

- Indexed allocation – 将所有指针汇集到一个位置，即索引块
  - 每个文件都有自己的索引块，其中包含指向其数据块或磁盘块地址的指针数组
- 逻辑视图



索引表

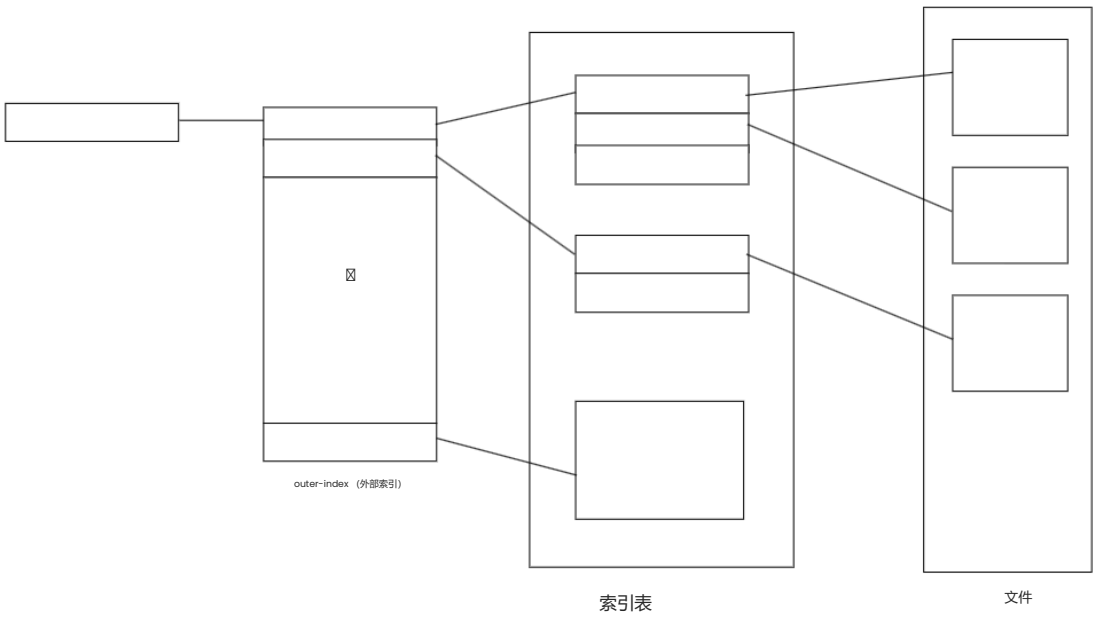
Indexed Allocation 示例



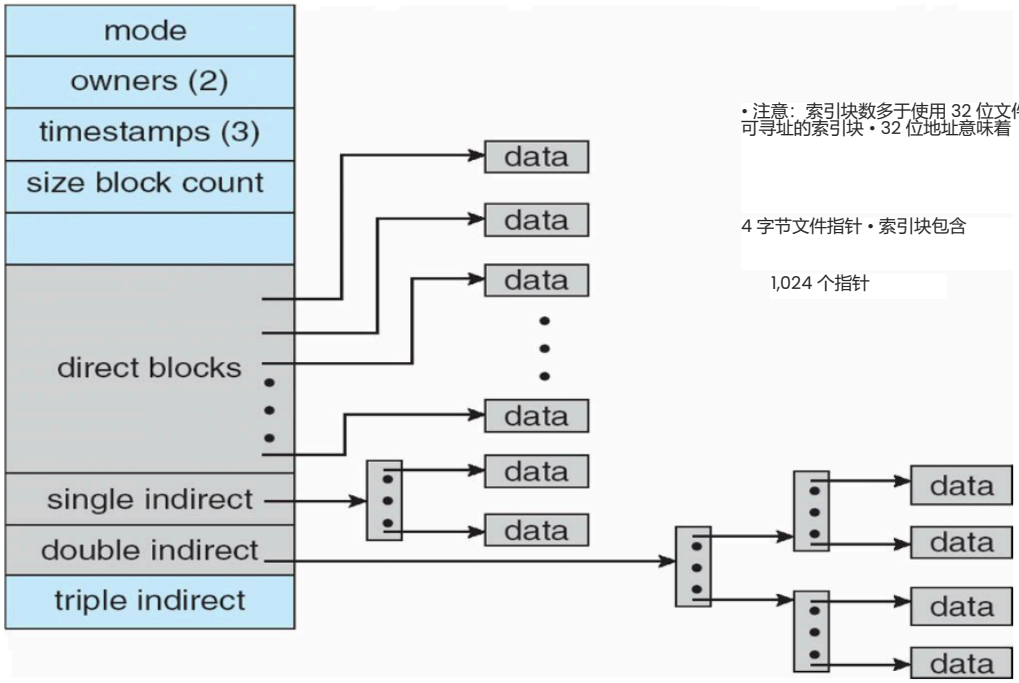
Indexed Allocation

- 索引分配支持直接访问，而不会受到外部碎片的影响，因为磁盘上的任何空闲块都可以满足对更多空间的请求
- Indexed allocation 存在一些与 linked allocation 相同的性能问题。具体来说，索引块可以缓存在内存中，但数据块仍可能分布在卷（磁盘）上。索引分配确实会浪费空间。索引块的指针开销通常大于链接分配中的指针开销
-
  - 假设一个文件只有一个或两个块。索引分配会丢失整个索引块，而链接分配只会丢失每个块一个指针的空间
- 到目前为止，一个索引块占用一个磁盘块。如果文件太大，以至于一个索引块太小而无法容纳足够的指针，会发生什么情况
  - 链接方案 – 将多个索引块链接在一起 多级方案 – 第一级索引块指向一组二级索引块，这些索引块又指向文件块。这可以继续到第三级或第四级，具体取决于所需的最大文件大小。对于一个 4,096 字节的块，我们可以在一个索引块中存储 1,024 个四字节指针。两个级别的索引允许 1,048,576 个数据块，文件大小高达 4GB。
  - 组合方案 – 用于小文件的直接块和用于较大文件的间接块（单个间接块、双间接块和三重间接块），用于基于 UNIX 的文件系统。

索引分配 – 多级方案



组合方案：UNIX UFS（每个块 4K 字节，32 位地址）



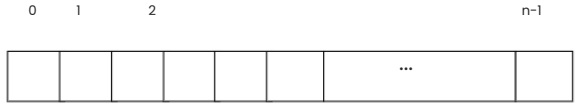
• 注意：索引块数多于使用 32 位文件指针可寻址的索引块 • 32 位地址意味着

4 字节文件指针 • 索引块包含

1,024 个指针

自由空间管理

- 文件系统维护可用空间列表以跟踪可用磁盘空间
  - （为简单起见，使用术语“块”）
- 位向量或位图（n 个块）



位[i] = 1 ⇒ 块[i] 免费
0 ⇒ 区块[i] 被占用

- 主要优点是它在磁盘上查找第一个空闲块或 n 个连续空闲块的简单性和效率
- 除非整个向量可以保存在内存中，否则位向量效率低下，但需要额外的空间

块大小 = 4KB = 212 字节 磁盘大小 = 240 字节（1TB） n = 240/212 = 228 位（或 32MB）

磁盘上链接的可用空间列表

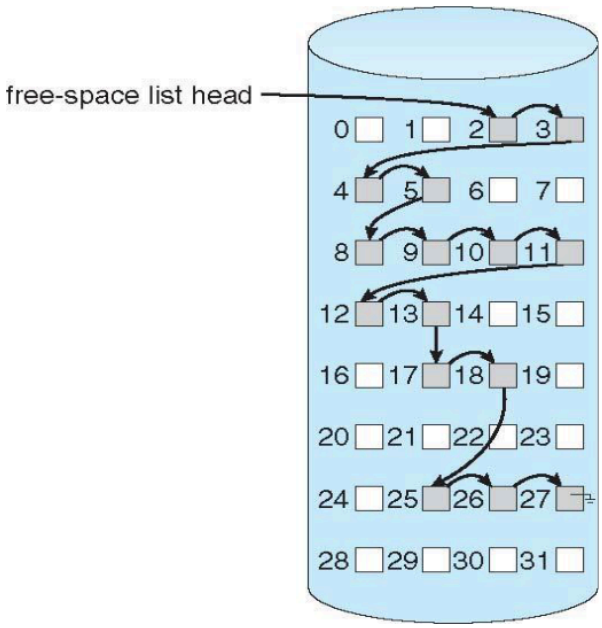
Linked-List - 将所有空闲磁盘块链接在一起

将指向第一个空闲块的指针保存在磁盘上的特殊位置，可以缓存在内存中

第一个块包含指向下一个空闲块的指针，依此类推

轻松定位一个空闲区块，不易获得连续区块

遍历整个列表效率不高，因为它必须读取每个块，这需要大量的 I/O 时间。幸运的是，这不是一个频繁的操作



自由空间管理 （续）

- 分组
  - 修改 linked-list 以将 n 个免费区块的地址存储在第一个空闲区块中，这些区块中的前 n-1 个是免费的。最后一个块包含另外 n 个空闲块的地址，依此类推。大量空闲块的地址可以比 linked-list 更快地找到
  -
- 计数 - 因为可以同时分配和释放多个连续块，尤其是在使用连续分配算法或扩展数据块时
  - 通过利用这一点，我们可以保留第一个空闲块的地址和后续空闲块的计数，而不是保留 n 个空闲磁盘地址的列表空闲空间列表中的每个条目都由一个磁盘地址和一个计数组成
  -
  - 尽管每个条目比简单磁盘地址需要更多的空间，但只要计数通常大于 1，则整个列表会更短
  - 请注意，这种跟踪可用空间的方法类似于分配块的 extent 方法。

第 14 章结束