OPERATING SYSTEMS

File System Implementation

Operating Systems

郑贵锋 博士
中山大学计算机学院
zhenggf@mail.sysu.edu.cn
https://gitee.com/code_sysu

中山大學 SUN YAT-SEN UNIVERSITY

- **目录**
  - 文件系统结构
  - 文件系统实现
  - 目录实现
  - 分配方法
  - 空闲空间管理
  - 效率与性能
  - 恢复
  - NFS
  - 示例：WAFL文件系统

File System Structure

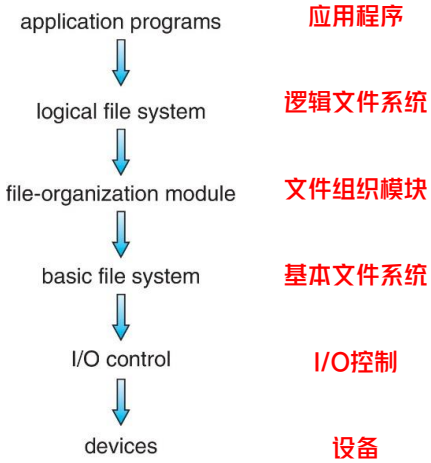
- **概述**
  - 文件结构
    - 逻辑存储单元
    - 相关信息的集合
  - 文件系统驻留在辅助存储（磁盘）上。
    - 提供存储的用户接口，将逻辑映射到物理。
    - 通过方便地存储、定位和检索数据，提供对磁盘的高效、方便的访问。
  - 磁盘提供原地重写（读-改-写回）和随机访问。
    - I/O传输在扇区组成的块中执行（扇区大小通常为512字节）。
      - 在Linux ext2/ext3中，块大小默认为4096字节。

```
$/sbin/tune2fs -l /dev/sda1 | grep "Block size"
```

  - 文件控制块(FCB) – 由文件的信息组成的存储结构。
  - 设备驱动程序控制物理设备。

File System Structure

- **文件系统层级**
  - 文件系统被组织成层
    - 每个层级使用较低层级的功能，创建供较高层级使用的新功能



File System Structure

- **文件系统层级**
  - I/O控制层由设备驱动程序和中断处理程序组成，用于在主存和磁盘系统之间传输信息。
    - 设备驱动程序管理I/O设备。给定诸如“read drive1, cylinder 72, track 2, sector 10, into memory location 1060”之类的命令，设备驱动器向硬件控制器输出特定于硬件的低级命令。
  - 基本文件系统（Linux中称为“块I/O子系统”）只需向相应的设备驱动程序发出通用命令，以读取和写入存储设备里的块。它根据逻辑块地址（如“retrieve block 123”）向驱动器发出命令。
    - 还管理内存缓冲区和缓存（分配、释放、置换）
      - 缓冲区(buffer)保存传输中的数据
      - 缓存(cache)保存经常使用的数据
  - 文件组织模块解析文件、逻辑地址和物理块。
    - 将逻辑块号转换为物理块号
    - 管理空闲空间、磁盘分配

File System Structure

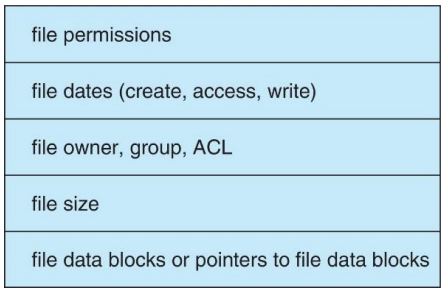
- **文件系统层级**
  - 逻辑文件系统管理元数据信息，包括除实际数据（或文件内容）之外的所有文件系统结构。
    - 通过维护文件控制块（UNIX中的 inodes 索引节点），将文件名转换为文件号、文件句柄和位置
    - 目录管理
    - 保护
  - 分层有助于降低复杂性和冗余，但会增加开销并降低性能。
    - 根据OS设计者，逻辑层可以通过任何编码方法实现。
  - 有多种文件系统，有时操作系统中有多个文件系统
    - 每个都有自己的格式：CD-ROM是ISO 9660；Unix有UFS, FFS；Windows有FAT, FAT32, NTFS以及软盘, CD, DVD蓝光；Linux有40多种类型，扩展文件系统ext2和ext3领先；加上分布式文件系统等
    - 新的类型仍在出现 — ZFS, GoogleFS, Oracle ASM, FUSE.

File System Structure

- **在存储上的结构**
  - 在存储上，文件系统可能包含如下信息：如何引导存储在其中的操作系统、块的总数、空闲块的数量和位置、目录结构和各个文件的信息。
  - 在存储上的结构
    - 引导控制块（每个卷）包含系统从该卷引导操作系统所需的信息。
      - 如果卷包含操作系统，则需要，通常为卷的第一个块。
      - 也称为引导块、分区引导扇区。
    - 卷控制块（每个卷）包含卷详细信息
      - 卷中的块总数、块大小、空闲块计数和指针，可用FCB计数和指针。
      - 也称为超级块、主控文件表。
    - 目录结构（每个文件系统）用于组织文件。
      - 在UFS中，包括文件名和关联的inode索引节点号。
      - 在NTFS中，存储在主控文件表中。

File System Structure

- **在存储上的结构**
  - 在存储上的结构（续）
    - FCB（每个文件）包含有关该文件的许多详细信息。
      - 它有一个唯一的标识号，允许与目录项关联。
      - UNIX中的inode索引节点编号、权限、大小、日期等。
      - NTFS使用关系数据库结构将信息存储在主控文件表中。



典型的文件控制块（FCB）

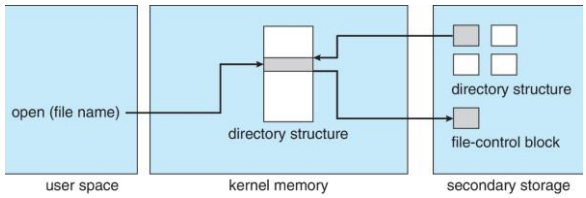
File System Structure

- **内存中的结构**
  - 内存中的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，在卸载时丢弃。可以包括几种类型的结构。
    - 挂载表包含有关每个挂载卷的信息。
    - 目录结构缓存保存最近访问的目录的目录信息。
    - 系统范围的打开文件表包含每个打开文件的FCB副本及其他信息。
    - 每个进程的打开文件表包含指向系统范围的打开文件表中相应条目的指针，以及进程已打开的所有文件的其他信息。
    - 缓冲区在读取或写入文件系统时保存文件系统块。

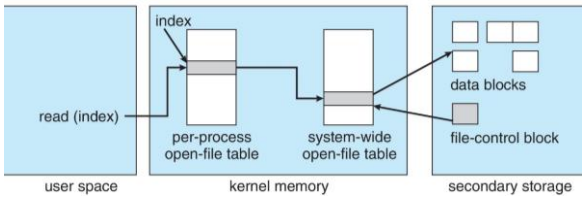
File System Structure

- **文件系统结构的使用**
  - 下图说明了操作系统提供的必要文件系统结构
  - 外加缓冲区保存来自辅助存储器的数据块
  - 打开文件返回一个文件句柄以供后续使用
  - 读取的数据最终复制到指定的用户进程内存地址

(a) 打开文件



(b) 读取文件



分区和挂载

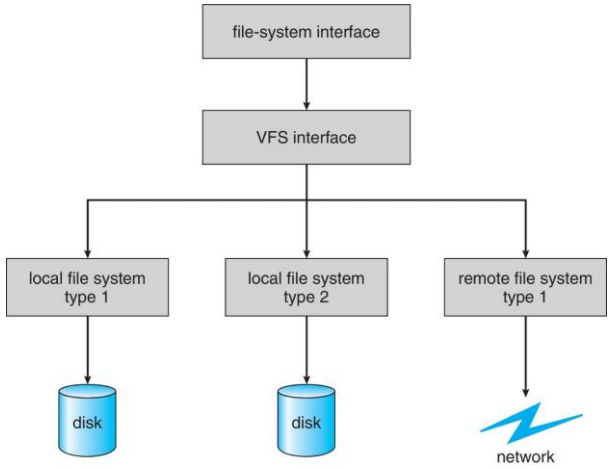
- 分区可以是包含文件系统（“熟”）或原始文件（“生”）的卷。
  - 原始分区只是一系列没有文件系统的块。
- 引导块可以指向引导卷或引导加载程序块集，包含足够代码以从文件系统加载内核。
  - 它也可能是用于多操作系统引导的引导管理程序。
- 根分区包含操作系统，其他分区可以容纳其他操作系统、其他文件系统，或者只是原始分区。
  - 根分区在启动时装入。
  - 其他分区可以自动或手动挂载。
- 挂载时，将检查文件系统的一致性。
  - 所有元数据都正确吗？
    - 如果存在错误，则修复它，再重试。
    - 如果没有错误，添加到挂载表，允许访问。

虚拟文件系统

- UNIX上的虚拟文件系统(VFS)提供了一种实现文件系统的面向对象方法。
- VFS允许对不同类型的文件系统使用相同的系统调用接口(API).
  - VFS将文件系统通用操作与实现细节分开。
  - 实现可以是多种文件系统类型之一，也可以是网络文件系统。
    - 实现vnode以包含索引节点inode或网络文件详细信息。
  - 然后将操作分派到适当的文件系统实现例程
- API是针对VFS接口的，而不是针对任何特定类型的文件系统。

File System Structure

- **虚拟文件系统**
  - 虚拟文件系统的示意图



File System Structure

- **虚拟文件系统**
  - 虚拟文件系统实现
    - 例如，Linux有四种对象类型：
      - inode、文件、超级块、dentry
    - VFS定义了必须实现的对象上的一组操作。
      - 每个对象都有一个指向函数表的指针。
        - 函数表具有在该对象上实现该函数的例程的地址。
      - 例如：

```
int open()          打开一个文件
int close()         关闭已打开的文件
ssize_t read()      从文件中读取
ssize_t write()     写入文件
int mmap()          内存映射文件
```

File System Structure

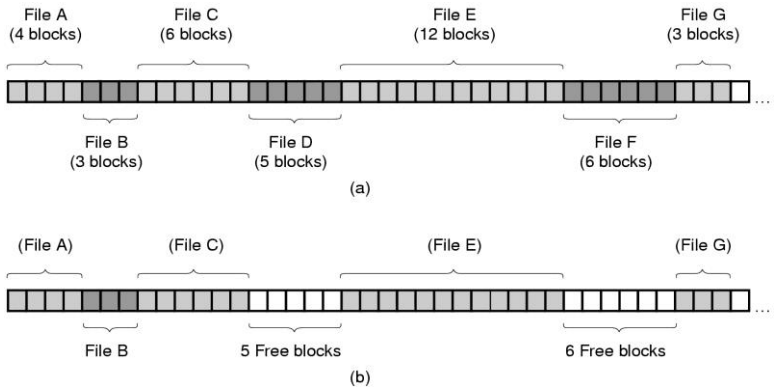
- **目录实现**
  - 目录分配和目录管理算法的选择会显著影响文件系统的效率、性能和可靠性。
  - 线性列表
    - 文件名和指向数据块的指针的线性列表。
      - 易于编程
      - 执行耗时
        - 线性搜索时间
        - 可以通过链表或使用B+树按字母顺序排序
  - 哈希表
    - 具有哈希数据结构的线性列表。
      - 缩短目录搜索时间
      - 碰撞 – 两个文件名散列到同一位置的情况
      - 只在使用固定大小条目或采用溢出链接方法时才有效

Allocation Methods

- **分配方法**
  - 分配方法是指如何为文件分配磁盘块。
    1. 连续分配
    2. 链接分配
    3. 索引分配
    4. 组合方案

Allocation Methods

- **连续分配**
  - 实例



（a） 为7个文件连续分配磁盘空间。
（b） 删除文件D和F后磁盘的状态。

Allocation Methods

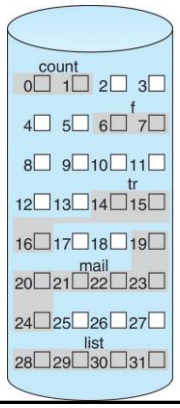
- **连续分配**
  - 每个文件占用磁盘上的一组连续块。
    - 简单：只需要起始位置（块号）和长度（块数）
    - 支持随机存取
    - 在大多数情况下性能最佳。
  - 缺点：
    - 浪费空间（动态存储分配问题）。
    - 文件无法增长。
  - 问题包括查找文件空间、需要知道文件大小、外部碎片、需要离线（停机期间）或在线合并空闲空间。

Allocation Methods

- **连续分配**
  - 从逻辑地址到物理地址的映射（每个块512字节）

    逻辑地址 = Q × 512 + R

    逻辑地址 / 512 = Q … R (商Q，余数R)
    - 要访问的物理块号是 Q (块)
    - 文件块中的偏移为 R (字节)
  - 实例

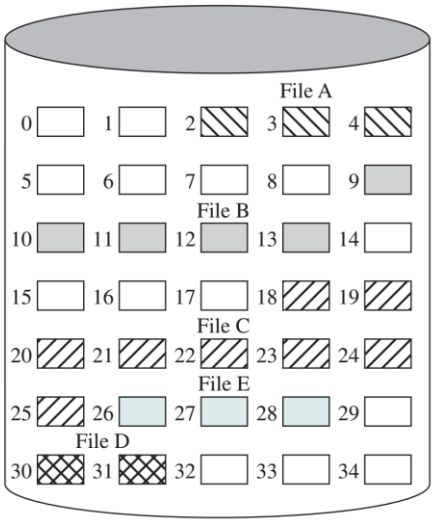


directory

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

Allocation Methods

- **连续分配**
  - 实例

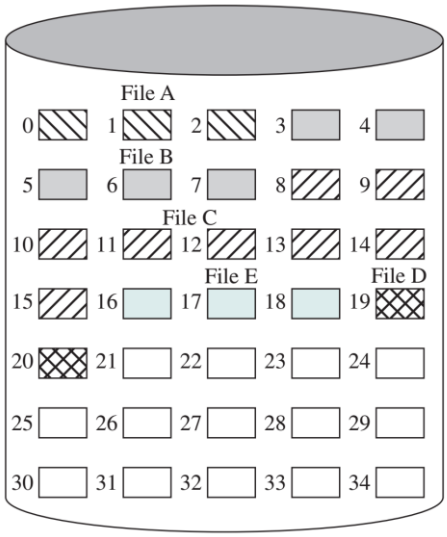


File allocation table

File name	Start block	Length
File A	2	3
File B	9	5
File C	18	8
File D	30	2
File E	26	3

Allocation Methods

- **连续分配**
  - 实例：压实(compaction)后



File allocation table

File name	Start block	Length
File A	0	3
File B	3	5
File C	8	8
File D	19	2
File E	16	3

Allocation Methods

- **连续分配**
  - 基于扩展(extent-based)的文件系统
    - 许多较新的文件系统（SPARC Veritas文件系统、Linux 2.6.19 EXT4等）使用改版的连续分配方案。
    - 基于扩展的文件系统在扩展（或称范围）中分配磁盘块。
    - 一个扩展由多个连续磁盘块组成。
      - 扩展在文件分配时分配。
      - 文件由一个或多个扩展组成。
    - 例如： ext4文件系统的扩展结构

```
struct ext4_extent {
  __le32  ee_block;   /* 扩展覆盖的首个逻辑块 */
  __le16  ee_len;     /* 覆盖的块数 */
  ...
};
```

Allocation Methods

- **链接分配**
  - 链接分配(Linked allocation, aka chained allocation)
    - 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。
      - 每个块都包含指向下一个块的指针。
      - 文件以NIL指针结束。
  - 优势：
    - 不需压缩(comaction)，没有外部碎片。
    - 当需要新块时，调用空闲空间管理程序。
    - 通过将块聚集成簇(cluster)来提高效率，但内部碎片会增加
    - 简单：只需要起始地址
    - 空闲空间管理：不浪费空间
  - 缺点：
    - 不支持随机存取
    - 定位块可能需要许多I/O时间和磁盘搜索。
    - 一些系统定期合并整理(consolidate)文件，需要很多I/O.
    - 可靠性可能是个问题（指针丢失或损坏）

Allocation Methods

- **链接分配**
  - 从逻辑地址到物理地址的映射(每个块512字节，块地址指针4字节)

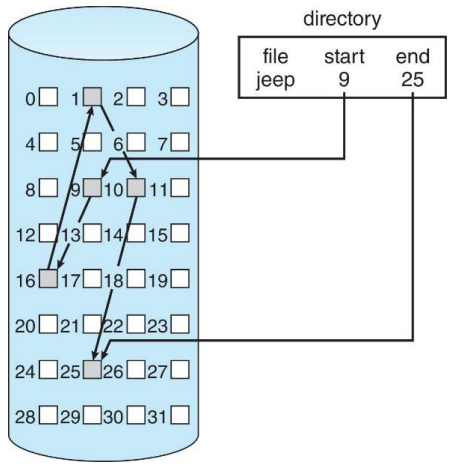
    逻辑地址 = Q × (512 - 4) + R
    - 要访问的块是文件的块链表中的第Q个块

      Q = 逻辑地址 % (512 - 4)
    - 每个块的前4个字节保留给块指针。文件块中的偏移

      偏移量 = R + 4

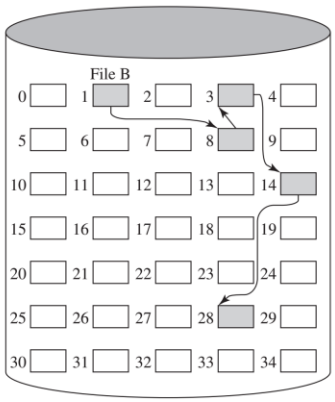
Allocation Methods

- **链接分配**
  - 实例



Allocation Methods

- **链接分配**
  - 实例

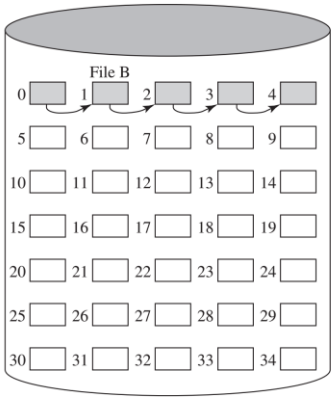


File allocation table

File name	Start block	Length
• • •	• • •	• • •
File B	1	5
• • •	• • •	• • •

Allocation Methods

- **链接分配**
  - 实例（合并整理后）



File allocation table

File name	Start block	Length
File B	0	5

Allocation Methods

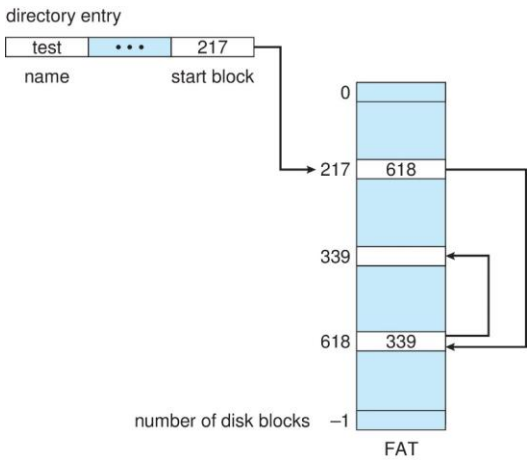
- **链接分配**
  - 文件分配表(FAT)
    - 链接分配的一个重要变种是文件分配表(FAT)的使用。在每个卷的开头留出一段存储空间来存储FAT（卷范围的数据结构）。
    - FAT对于卷中的每个块都有一个条目，并按块号索引。
      - 目录项包含文件第一个块的块号。
      - 由该块号索引的FAT条目包含文件中下一个块的块号。
      - 此链将继续，直到到达最后一个块，该块具有一个特殊的EOF(end-of-file)文件结束值作为表条目。
    - 未使用的块在FAT表中用0值表示。
      - 将新块分配给文件
        1. 查找第一个空闲块，即值为0的表项
        2. 其索引就是新块的地址，填入原文件结尾块的值中，即使原结尾块指向新块。
        3. 新块作为新的结尾，将新块的0值替换为EOF.

Allocation Methods

- **链接分配**
  - 文件分配表(FAT)
    - 除非缓存FAT，否则FAT分配方案可能导致大量磁头寻道。
      - 磁头必须移动到卷的开头以读取FAT并找到相关块的位置，然后移动到块本身的位置。在最坏的情况下，每个块都会发生两次移动。
    - 一个好处是随机访问时间得到了改善，因为磁头只需读取FAT中的信息就能找到任何块的位置。

Allocation Methods

- **链接分配**
  - 文件分配表(FAT)
    - 示例：由磁盘块217、618和339组成的文件。

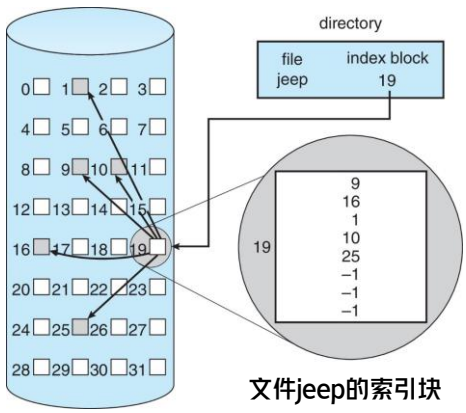


Allocation Methods

- **索引分配**
  - 链接分配解决了连续分配的外部碎片和声明大小问题。
    - 在没有FAT的情况下，链接分配不能支持有效的直接访问，因为必须按顺序检索块。
  - 索引分配通过将所有指针集中到一个位置（索引块）来解决此问题
    - 每个文件都有自己的索引块，它是存储块地址的数组。索引块中的第i个条目指向文件的第i个块。
    - 索引块的地址包含在目录结构中。
  - 索引分配支持直接访问而不受外部碎片的影响，但仍受到空间浪费的影响。
    - 即使文件只有一个或两个块，也必须分配整个索引块。这一点提出了一个问题：索引块应该有多大，应尽可能小？

Allocation Methods

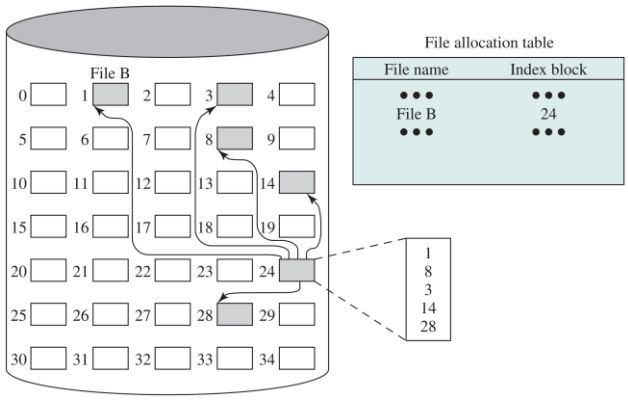
- **索引分配**
  - 实例



文件jeep的索引块

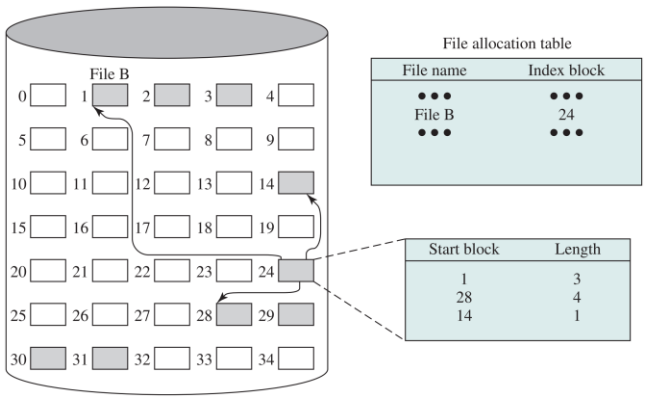
Allocation Methods

- **索引分配**
  - 实例（索引块基于单块建立索引）



Allocation Methods

- **索引分配**
  - 实例（索引块基于可变长度的连续多块建立索引）



Allocation Methods

- **索引分配**
  - 索引块方案
    - 单块方案（示例）
      - 假设块大小为512字节，只用一个索引块时，可保存索引表的条目数为128个（每条目4字节磁盘指针）
      - 则只用一个索引块时，最大可为大小为64K字节的文件，实现从逻辑地址到物理地址的映射（128×512字节)

        逻辑地址 = Q × 512 + R
      - Q = 索引块内索引表的偏移
      - R = 文件块内的偏移

Allocation Methods

- **索引分配**
  - 索引块方案
    - 链接方案
      - 在一个不限大小且块大小为512字节的文件中，从逻辑地址映射到物理地址，使用链接方案链接索引块，其中每个索引块中含文件名，100个磁盘块地址集（索引表），以及一个指向下一索引块的地址。

        逻辑地址 = $Q_1 \times (512 \times 100) + Q_2 \times 512 + R$
      - Q_1 = 所在块的索引块在链表中的序号

        Q_1 = floor(逻辑地址 / (512 × 100))
      - Q_2 = 所在块的索引块的索引表内的偏移

        Q_2 = floor((逻辑地址 % (512 × 100)) / 512)
      - R = 文件块内的偏移

** floor(x)：对 x 下取整*

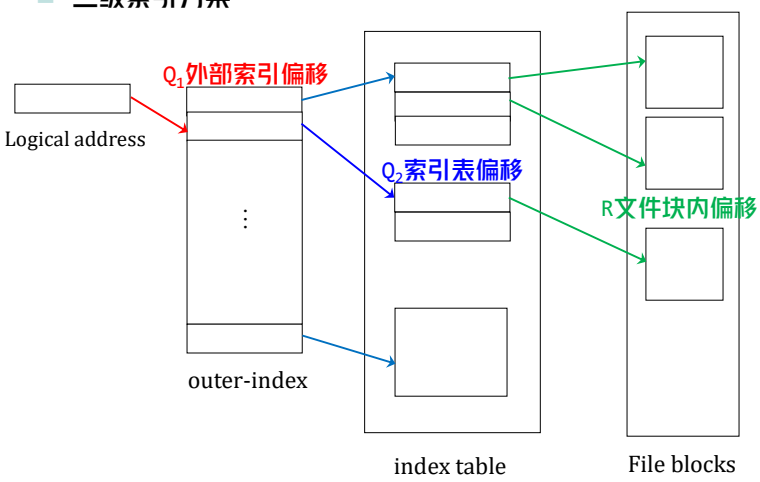
Allocation Methods

- **索引分配**
  - 索引块方案
    - 二级索引方案
      - 4K字节块可以在外部索引中存储2^{10}个4字节指针
        - 最多可表示的文件块数
          - $2^{10} \times 2^{10} = 1{,}048{,}567$
        - 文件大小不超过
          - $4(KB) \times 2^{10} \times 2^{10} = 4(GB)$
      - 从逻辑地址到物理地址的映射

        逻辑地址 = $Q_1 \times (4096 \times 1024) + Q_2 \times 4096 + R$
      - Q_1 = 进入外部索引的偏移
      - Q_2 = 进入索引表块的偏移
      - R = 文件块中的偏移

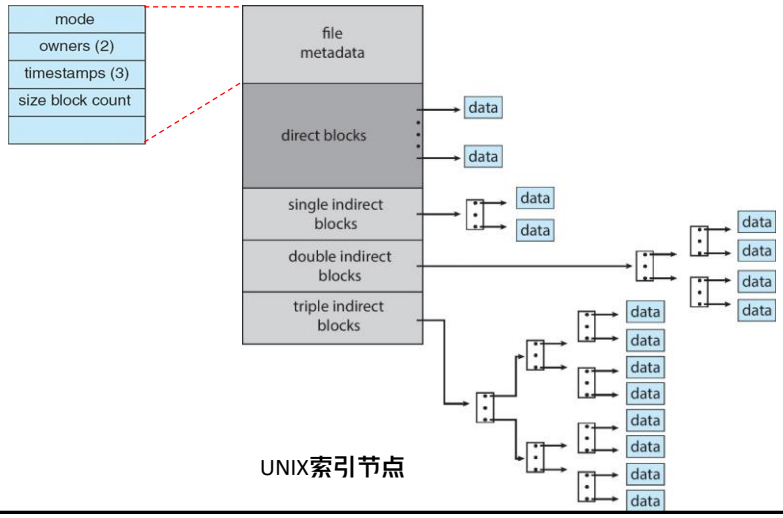
Allocation Methods

- **索引分配**
  - 索引块方案
    - 二级索引方案



Allocation Methods

- **索引分配**
  - 索引块方案
    - 组合方案：UNIX UFS（每个块4K字节）



UNIX索引节点

Allocation Methods

- **比较**
  - 文件分配方法的比较

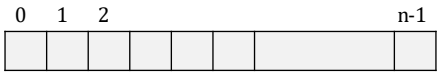
	连续分配	链接分配	索引分配	
预分配?	必需	可能	可能	
单块/可变块数?	可变长	固定块	固定块	可变长
每份空间大小	大	小	小	中等
分配频率	一次性	从低到高	高	低
分配时间	中等	长	短	中等
文件分配表大小	一项	一项	大的	中等

Allocation Methods

- **性能**
  - 最佳方法取决于文件访问类型。
    - 连续分配适用于顺序和随机分配。
    - 链接分配适用于顺序分配，而不是随机分配。
    - 如果在创建时声明了访问类型，那么我们可以选择连续分配或链接分配。
  - 索引分配更为复杂。
    - 单块访问可能需要读取2个索引块，然后读取数据块。
    - 聚簇可以帮助提高吞吐量，减少CPU开销。
  - 向操作系统添加数千条额外指令以节省少量磁头移动是合理的。
    - 英特尔酷睿i7极限版990x (2011), 3.46Ghz = 159000 MIPS（每秒百万指令）
    - 典型的磁盘驱动器的IOPS为250（每秒I/O次数）。
      - 159000 MIPS/250 = 每磁盘I/O 相当 630 百万指令
    - 快速SSD驱动器提供60000 IOPS。
      - 159000 MIPS/60000 = 每磁盘I/O 相当 2.65 百万指令

Free Space Management

- **空闲空间管理**
  - 文件系统维护空闲空间列表以跟踪空闲块/簇 (block/cluster)
    - 为简单起见，使用术语“块”
  - n块的位图
    - 位图(bitmap) 又称位向量或位表



$$bit[i] = \begin{cases} 1 & block[i] \text{ 空闲} \\ 0 & block[i] \text{ 被占用} \end{cases}$$

    - 实例：考虑一个磁盘，其中块 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, 和27是空闲的，其余的块被分配。空闲空间位图将是

      00111100111111000110000011100000 ...

Free Space Management

- **空闲空间管理**
  - n块的位图
    - 如何在使用位图分配空间的系统上找到第一个空闲块?
      - 顺序检查位图中的每个字(word)，查看该字的值是否非0，因为0值的字的位都是0，表示一组占用块。扫描第一个非0字的第1个值为1的位，这是第一个空闲块的位置。
    - 定位块编号的计算如下所示：
      （每个字的位数）×（0值字数）+（第一个"1"位的偏移量）
      - CPU有指令返回字内第一个"1"位的偏移量。

Free Space Management

- **空闲空间管理**
  - n块的位图
    - 将位图表保存在磁盘或主内存中需要额外的空间。
    - 实例

      块大小 = 512字节 = 2^9字节

      磁盘大小 = 2^{34}字节（16 GB）

      块数 n = $2^{34}/2^9 = 2^{25}$

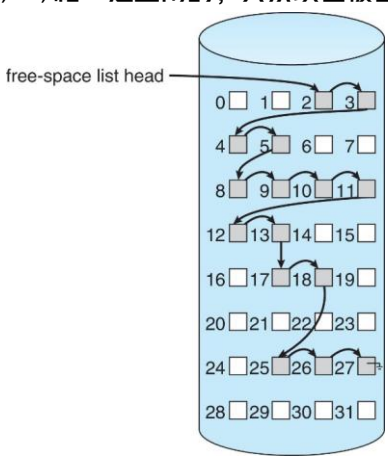
      位图大小 = 2^{25}位 = 2^{22}字节 = 4MB.
    - 4MB对于主内存来说是一个巨大的块。另一种方法是将位图表放在磁盘上。
      - 4MB位表需要 $2^{22-9} = 2^{13} = 8192$个磁盘块。我们负担不起每次需要一个块时搜索那么多的磁盘空间，因此需要在内存中保留一个位表。
    - 即使位表在主内存中，对该表的彻底搜索也会将文件系统性能降低到不可接受的程度。

Free Space Management

- **空闲空间管理**
  - 空闲块列表（链表）
    - 空闲块列表将所有空闲块链接在一起，将指向第一个空闲块的指针保留在文件系统中的特定位置，并将其缓存在内存中。每个空闲块都包含指向下一个空闲块的指针。
    - 要遍历列表，必须读取每个块，这需要硬盘上大量的I/O时间。
      - 事实上，并不会频繁地遍历空闲列表。通常，操作系统只需要一个空闲块，使用空闲列表中的第一个块就可以了。
    - 无法轻松获得连续空间
    - 不会浪费空间

Free Space Management

- **空闲空间管理**
  - 空闲块列表（链表）
    - 回想一下我们前面的示例，其中块 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, 和27是空闲的，其余块已被占用



Free Space Management

- **空闲空间管理**
  - 分组和计数
    - 分组
      - 空闲块列表的改进，将n个空闲块的地址存储在第一个空闲块中。这些块中的前n-1实际上是空闲的，最后一个块包含另n个空闲块的地址，依此类推。现在可以快速找到大量空闲块的地址。
    - 计数
      - 事实上，空间经常被连续使用和释放，具有连续分配的分配、扩展或聚簇。
        - 保存第一个空闲块i的地址及其后的连续空闲块的数量n
        - 空闲空间列表中的每个条目由磁盘地址和计数组成。
      - 这些条目可以存储在平衡树中，而不是链表中，以便高效地查找、插入和删除。

Free Space Management

- **空闲空间管理**
  - 空间图
    - Oracle的ZFS（Zettabyte文件系统，Sun Solaris 10）旨在包含大量的文件、目录甚至文件系统。在这些规模上，元数据I/O会对性能产生很大影响。
      - 像位图这样的完整数据结构无法适用，需要数千个I/O.
    - 将磁盘空间划分为metaslab单元并对其进行管理。
      - 给定的卷可以包含数百个metaslab。
      - 每个 metaslab 都有一个关联的空间图。
        - 使用计数算法
      - 但记录要写到日志文件而不是文件系统。
        - 以计数格式按时间顺序记录所有块活动（分配/释放）
    - Metaslab活动是以平衡树结构将空间图加载到内存中，并按偏移量进行索引。
      - 重播日志把空间装入平衡树结构中
      - 合并连续的空闲块到单个条目中

Other Considerations

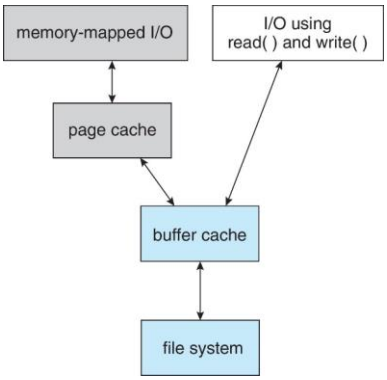
- **效率与性能**
  - 效率取决于
    - 磁盘分配和目录算法。例如UNIX inode预先分配在卷上
    - 文件在目录项中保存的数据的类型。例如“最后访问日期”
    - 元数据结构的预先分配或按需分配
    - 固定大小或可变大小的数据结构
  - 性能
    - 磁盘缓存
      - 主存储器的独立部分，用于经常使用的块。
    - 随后释放（删除用过的）与预先读取（多读一些页面）
      - 优化顺序存取的技术。
    - 将部分内存专用于虚拟磁盘或内存磁盘来提高PC的性能。

Other Considerations

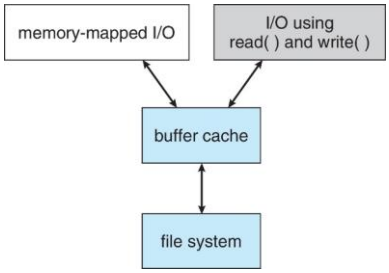
- **效率与性能**
  - 缓冲区（磁盘）缓存 Buffer-Cache
    - 有些系统提供独立内存作为缓存，给文件系统的例行I/O使用
  - 页面缓存 Page-Cache
    - 页面缓存使用虚拟内存技术缓存页面而不是磁盘块。
    - 内存映射I/O接口使用的是页面缓存。
    - 统一虚拟内存：多个系统采用页面缓存来缓存进程页面和文件数据，包括Solaris, Linux 和 Windows.
  - 统一缓冲区缓存 Unified Buffer Cache
    - 使用相同的页面缓存来缓存内存映射页面和普通文件系统I/O，以避免双重缓存。

Other Considerations

- **效率与性能**
  - 统一缓冲区缓存



缺少统一缓冲区缓存的I/O

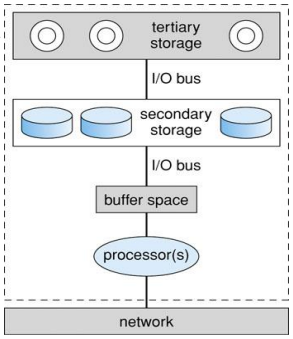


采用统一缓冲区缓存的I/O

Other Considerations

- **恢复**
  - 一致性检查
    - 将目录结构中的数据与磁盘上的数据块进行比较，并尝试修复不一致之处。
  - 使用系统程序将数据从磁盘备份到另一个存储设备（磁带、其他磁盘、光盘）。
    - 通过从备份中恢复数据来恢复丢失的文件或磁盘。

存储备份



Other Considerations

- **日志结构文件系统**
  - 日志结构文件系统将每个元数据更新到文件系统的操作作为事务记录在日志里
  - 所有事务都会写入日志。
    - 事务写入日志（按顺序）后即被视为已提交。
      - 有时会将数据传输到单独的设备或磁盘的独立部分。
    - 但是，文件系统可能尚未更新。
  - 日志中的事务以异步方式写入文件系统结构。
    - 修改文件系统结构后，事务将从日志中删除。
  - 如果文件系统崩溃，则仍必须执行日志中的所有剩余事务。
    - 更快地从崩溃中恢复，消除元数据不一致的可能性。