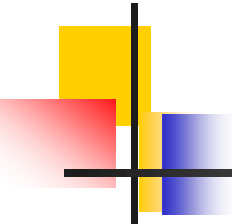




## 第9章 文件系统实现

---



---

第9章 文件系统实现

## 9.1 文件系统结构



# 文件结构

- 文件结构
  - 逻辑存储单元，相关信息的集合
- 文件系统驻留在二级存储（磁盘）上
  - 提供高效、便捷的磁盘访问，实现数据的存储、定位与访问
  - 两个任务：
    - 定义用户接口：定义文件及属性、文件操作、组织文件的目录结构
    - 创建算法和数据结构：实现逻辑单元到物理单元的映射
- 磁盘提供原地的重写、随机的访问
  - I/O传输以块为单位（扇区的集合，一个扇区32—4096Bytes，通常为512Bytes）
- 设备驱动：控制物理设备的系统软件
- 文件控制块FCB： 包含保存文件属性信息的数据结构



# 文件的用户视图和系统视图

- 用户视图
  - 文件管理的数据结构
  - 通过系统调用访问文件，不关心磁盘的存储方式
- 操作系统视图
  - 数据块集合
  - 数据块是逻辑存储单元，扇区是物理存储单元

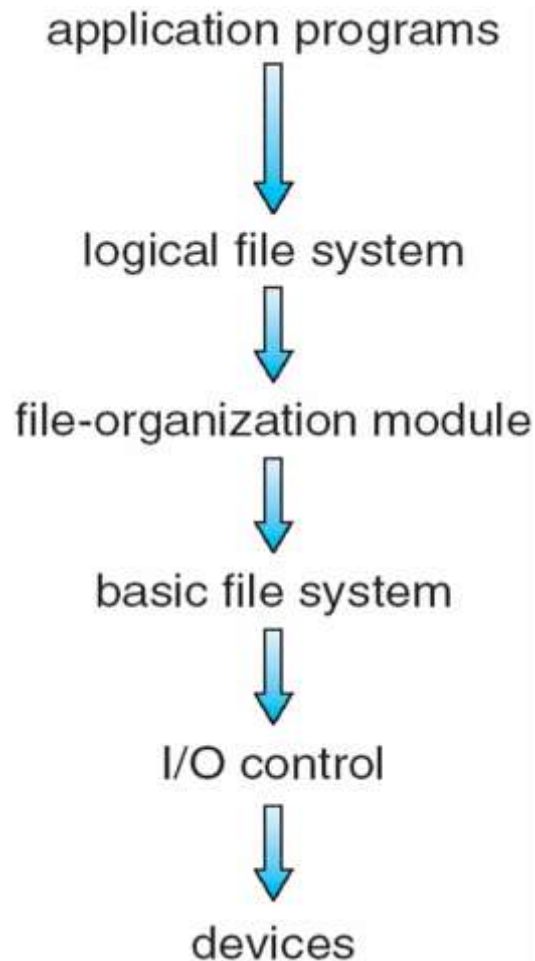


# 用户视图到系统视图的转换

- 读写文件
  - 获取数据块
  - 缓存数据块
  - 修改数据块
  - 写回数据块
- 文件系统中的基本操作单位是数据块
  - 块大小并不等于扇区大小
    - 操作系统读取硬盘的时候不会一个个扇区的读取，这样效率太低，而是一次性连续读取多个扇区，即一次性读取一个“块” (block)
  - getc putc即使每次只访问1个字节，也需要缓存目标数据块，如4096B

# 分层设计的文件系统

- 文件系统是分层组织的





# 文件系统层次

---

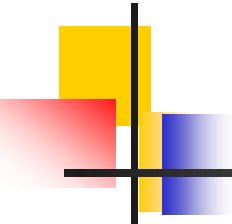
- I/O控制层(I/O Control)
  - 通过设备驱动和中断处理实现对I/O设备的管理
  - 设备驱动程序会对来自于上层的指令进行翻译，并向硬件控制器发出具体的硬件相关的指令
- 基本文件系统(Basic File System)
  - 向设备驱动程序发送通用命令，从而读取和写入磁盘的物理块
  - 管理内存缓冲区和缓存（分配、回收、替换等）
    - Buffer：存储传输时的缓冲数据
    - Cache：存储频繁使用的文件系统元数据，如文件系统、目录、数据块缓存



# 文件系统层次

- 文件组织模块(File-organization Module)
  - 该层知道文件、以及相关的逻辑块和物理块
  - 将逻辑地址转换成物理地址
  - 管理空闲空间、磁盘分配等
- 逻辑文件系统(Logical File System)
  - 负责管理元数据信息，即文件系统的所有结构
  - 根据符号文件名，管理目录结构
  - 根据文件控制块，将文件名翻译成文件标识号、文件句柄和具体位置等，如inode
  - 负责保护和安全
- 分层结构可有效减少复杂性和冗余，但增加了额外开销，导致性能降低。
- 典型的文件系统
  - ISO9660(CD-ROM)、UFS(UNIX)、FAT32、NTFS、ext2、ext3、GoogleFS





---

第9章 文件系统实现

## 9.2 文件系统实现



# 1. 文件系统实现概述

---

- 已有了在API层次的系统调用，如何实现其具体功能呢？
  - 盘上结构：告知如何启动OS、总的块数、空闲块情况、目录结构、具体的文件
  - 内存结构：用于管理文件系统并通过缓存提高性能



# (1) 盘上结构

---

- 引导控制块, Boot control block
  - 包含从相应卷引导操作系统的所需信息, 通常为卷的第一块, 若为空, 则无OS
  - UFS: boot block
  - NTFS: Partition Boot Sector
- 卷控制块, Volume Control Block
  - 包含卷/分区的详细信息: 块的总量、块的大小、空闲块数量和指针、空闲FCB数量及指针
  - UFS: Superblock
  - NTFS: Master File Table



# (1) 盘上结构(cont.)

- 目录结构：Directory Structure
  - 组织所有的文件
  - UFS：包含文件名+inode号码
  - NTFS：存储于master file table中
- 文件控制块：File Control Block, FCB
  - 每个文件对应一个，包含保存文件属性信息的数据结构
  - 具有唯一的标识号，与目录条目关联，往往又被称为文件目录项
  - 文件=文件控制块+文件体
  - UFS：inode号、权限、大小、日期等
  - NTFS：存储在Master File Table中，使用关系数据库存储，每个文件占一行

# 文件控制块的内容

- 文件控制块包含的具体内容因操作系统而异，但至少应包括以下信息：
  - 文件名：标识一个文件的符号名。
  - 文件类型：如文本文件。
  - 文件结构：说明文件的逻辑结构和物理结构。
  - 文件的物理位置：指示文件在外存上的存储位置。包括设备名、存储地址及文件长度等。
  - 存取控制信息：指示文件的存取权限。
  - 管理信息：包括文件建立的日期及时间、上次存取日期及时间、当前文件使用状态信息。

file permissions

file dates (create, access, write)

file owner, group, ACL

file size

file data blocks or pointers to file data blocks



# DOS的文件控制块

偏移	长度(字节)	意义
-7	1	扩展FCB标志(0FFH)
-6	5	保留
-1	1	文件属性字节
标准FCB		
00H	1	驱动器号：0为当前驱动器，1为驱动器A，2为B...
01H	8	文件名（8.3文件名格式）
09H	3	文件扩展名
0CH	2	当前块
0EH	2	文件长度记录，默认值为128
10H	4	文件长度
14H	2	文件建立日期
16H	2	文件建立时间
18H	8	保留
20H	1	当前记录号
21H	4	随机记录号



## (2) 内存结构

---

- 安装表, Mount Table
  - 存储文件系统安装卷、安装点、以及文件系统类型
- 目录结构缓存
  - 最近访问目录的信息
- 系统打开文件表, System-wide open-file table
  - **每个打开文件**的FCB副本以及其他信息
- 进程打开文件表, per-process open-file table
  - **指向与该进程相关的**系统打开文件表中已打开文件的条目, 以及相关信息



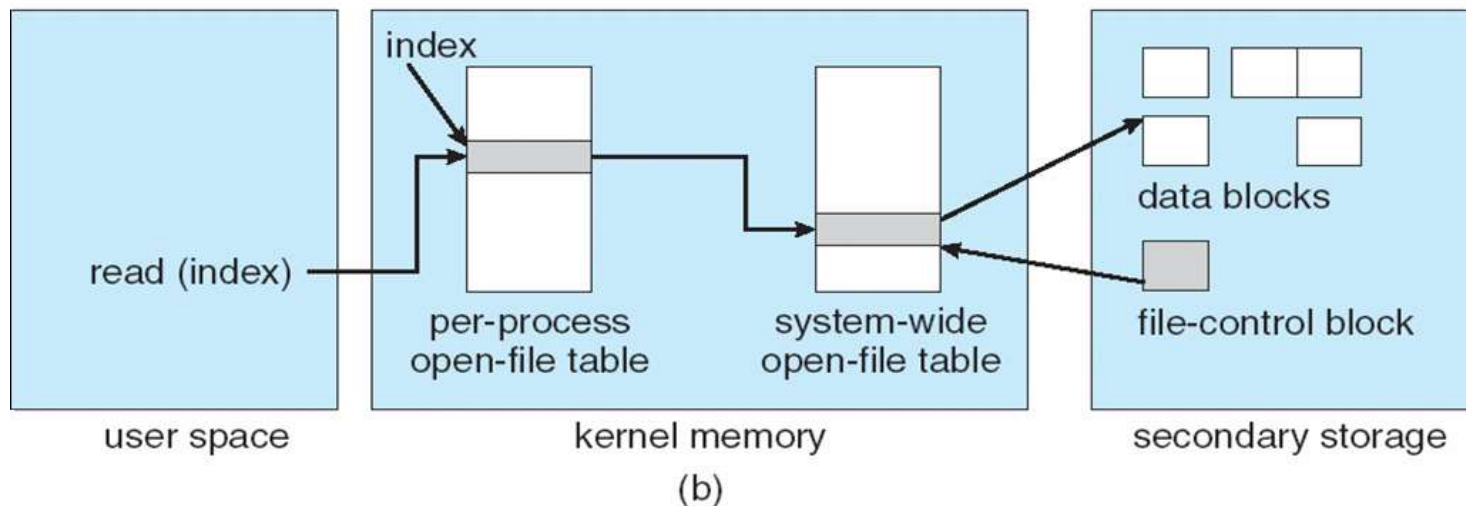
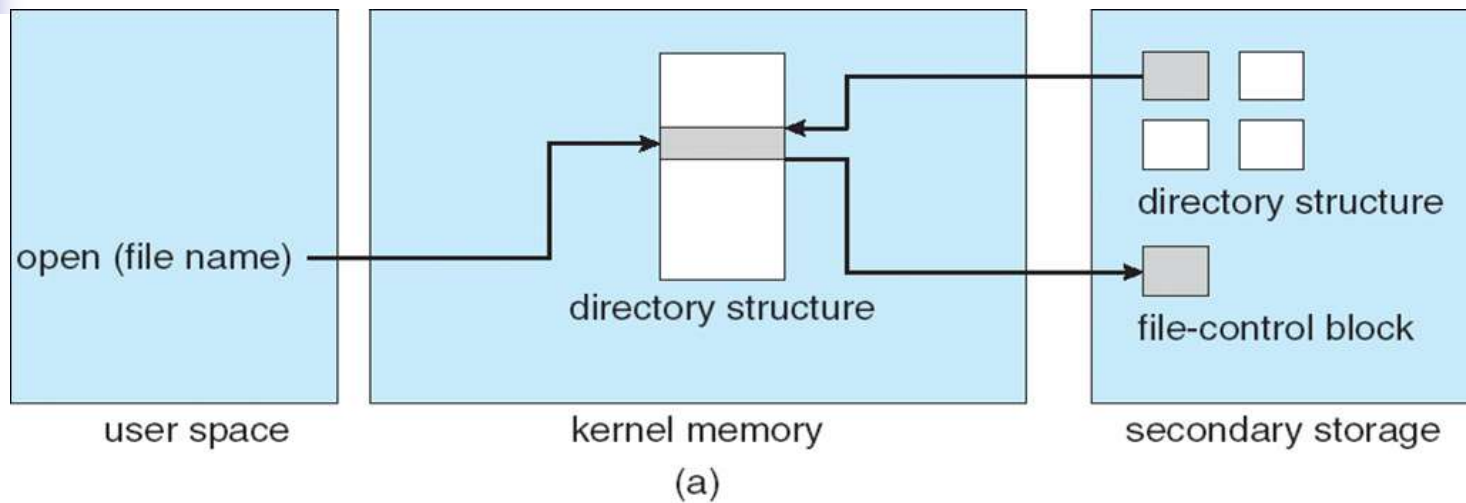
## (2) 内存结构：系统调用open()

### 基本步骤

- ① 系统调用open () , 传递到逻辑文件系统层
- ② 逻辑文件系统层查找系统打开文件表, 确认这个文件是否被其他进程使用中
  - ① 如果是, 就在**进程打开文件表中创建一个条目**, 让其指向系统打开文件表
  - ② 如果不是, 则根据给定文件名**搜索目录结构**, 目录结构往往缓存在内存中, 以加快速度。
    - ① 找到文件之后, 就将他的FCB复制到系统打开文件表中。
    - ② 进程打开文件表中会创建新条目, 指向系统打开文件表的对应条目。
- ③ 为open()返回进程打开文件表的对应指针, read()等就能使用了



## (2) 内存结构





## (2) 内存结构

- 文件名不是一直必须的，为了查找目录性能，可以缓存
- 打开文件表的条目名称：UFS：File Descriptor，NTFS：File Handle
- 当文件关闭时
  - 单个进程表的条目**会被删除**，整个系统表的条目的**打开数量会递减**。
  - 当所有打开该文件的用户关闭它时，任何更新的元数据会被复制到基于磁盘的目录结构，并且整个系统的打开文件表的条目会被删除。



## 2. 分区和安装

- 磁盘布局：取决于OS
  - 一个磁盘可以分成多个分区
  - 一个卷也可以跨越多个磁盘的多个分区
- 分区, Partitions
  - 可以是一个包含文件系统的卷, 也可以是没有文件系统的raw disk
- 引导块, Boot Block
  - 指向引导卷或者引导加载程序(Boot Loader), 包含代码, 实现如何从文件系统中加载内核
  - 或引导管理程序来支持多OS引导
- 根分区, Root Partition
  - 包含操作系统内核、其他的系统文件
  - 在引导时安装
- 其他分区可以在引导时安装, 或手动安装
- 在安装时, 文件系统一致性会被检查: 针对元数据
  - 如果格式无效, 修复, 并重试
  - 如果格式正确, 则加入内存的安装表中



## 3. 虚拟文件系统

---

- 虚拟文件系统Virtual File System
  - 目标：支持多类型文件系统，并实现无缝迁移
  - 基本思路：面向对象的技术
  - 原则：
    - 采用相同的系统调用接口支持不同的文件系统类型
    - 采用通用数据结构和操作隔离实现细节

# 3. 虚拟文件系统

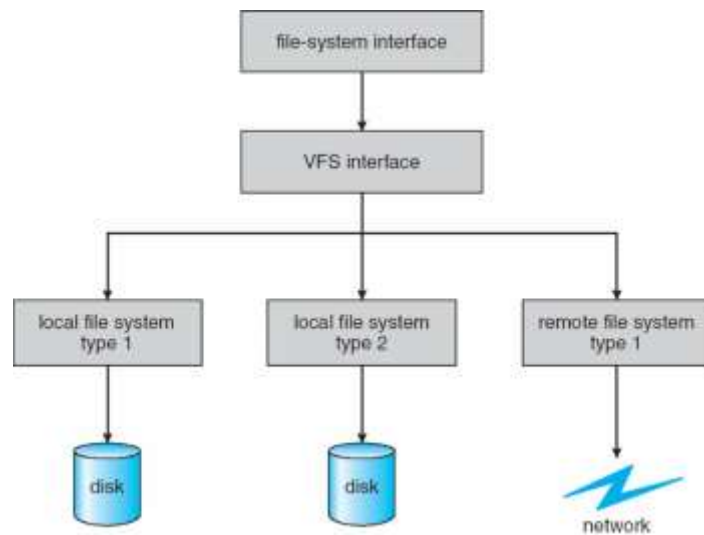
## ■ 文件系统的基本组成

### ■ 第一层：文件系统接口

- open(), read(), write(), close(), 以及文件描述符

### ■ 第二层：虚拟文件系统层VFS

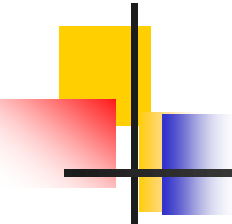
- VFS接口：将通用操作和实现分离；
- VFS接口的不同实现可共存于同一个物理主机，允许透明访问不同的文件系统，同时也支持网络文件
- vnode：对于网络文件提供唯一标识(本地采用inode)
- VFS根据不同文件类型，提供不同处理流程





## 3. 虚拟文件系统：Linux

- Linux VFS定义了四种对象：
  - 索引节点对象，inode：表示一个单独的文件
  - 文件对象，file：表示一个已打开文件
  - 超级块对象，superblock：表示整个文件系统
  - 目录条目对象：dentry：表示单个目录条目
- Linux VFS相关对象的操作
  - 每个对象都有一个指针，指向一个函数表，file\_operations
  - 函数表包含了实现特定对象操作的实际函数地址
    - int open(...)
    - int close(...)
    - ssize\_t read(...)
    - ssize\_t write(...)
    - int mmap(...)



---

## 第9章 文件系统实现

# 9.3 目录实现

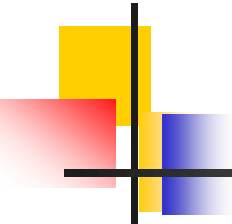


# 目录实现

---

- 目录分配与目录管理的算法选择会影响文件系统效率、性能和可靠性
- 线性列表
  - 文件名+数据块指针，用文件名作为检索关键词
  - 管理简单，但搜索费时
    - 线性搜索时间，可采用软件缓存提高搜索效率
    - 排序列表可减少平均搜索时间，但需比较大代价维护文件的创建删除等问题，可用B+树等
- 哈希表
  - 线性列表+哈希数据结构
  - 将文件名哈希，减少目录搜索时间
  - 需要考虑碰撞问题





---

第9章 文件系统实现

## 9.4 文件存储空间的分配方法



## 9.4 文件存储空间的分配方法

---

- 主要分配方法
  - 连续分配
  - 链接分配
  - 索引分配

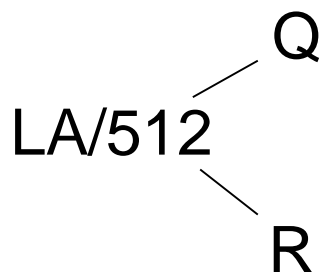


# 1. 连续分配

- 分配的目标：
  - 如何为文件分配磁盘块？
  - 如何实现有效使用磁盘空间，且快速访问文件？
- 连续分配，Contiguous allocation
  - 每个文件占用一组连续的盘块
  - 在大多数情况下性能最优：寻道数最小
  - 访问简单：对于顺序访问和直接访问，只需要上次访问的磁盘位置和长度
  - 问题：为新文件寻找空间
    - 文件信息：需要知道文件占用多大空间
    - 分配算法：首次适应，最优适应
    - 外部碎片问题：紧凑合并技术，离线 or 在线？

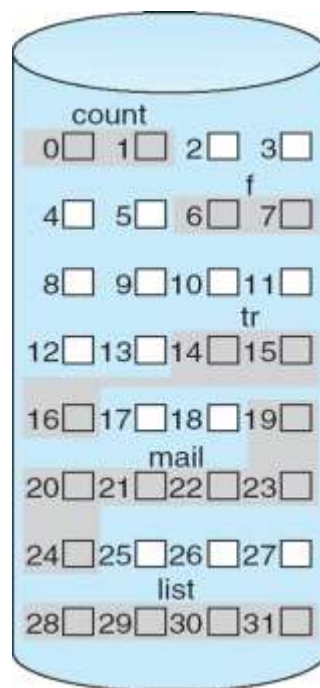
# 1. 连续分配

- 从逻辑地址映射到物理地址



物理块号 =  $Q + \text{起始地址}$

块内偏移地址 =  $R$



directory

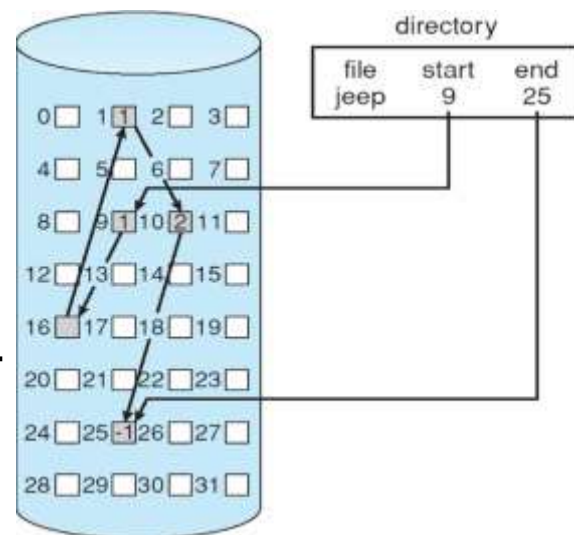
file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

- 扩展问题：基于扩展的文件系统
  - 添加扩展块

## 2. 链接法

### ■ 链接法：

- 每一个文件是磁盘块的链表
- 文件以空指针为结尾
- 无外部碎片，不需要合并
- 每个块包含了指向下一块的指针
- 问题
  - 不支持直接访问，需遍历链表
    - 定位某个块需要许多I/O和磁盘寻道
  - 增加了效率，但是记录指针需要存储开销
    - 由按块改为采用分簇（一簇对应多块）方法，减少块数降低开销，但是引入内部碎片
  - 可靠性：指针丢失或损坏问题



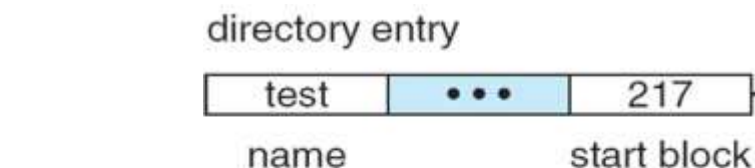


## 2.链接法举例：FAT

- FAT (File Allocation Table)
  - 用于MS-DOS
  - 每个卷开头的磁盘用于存储该表
  - 表中，每个磁盘块都有一个FAT条目，并可按块号索引
  - 使用方法类似链表
    - 目录项包含文件首块的块号，根据该块号索引的FAT条目包含文件的下一块块号
    - FAT如果不采用缓存，会带来大量的寻道时间，磁头必须不断移到卷开头，读入FAT，找到所需块位置，再移到块本身位置

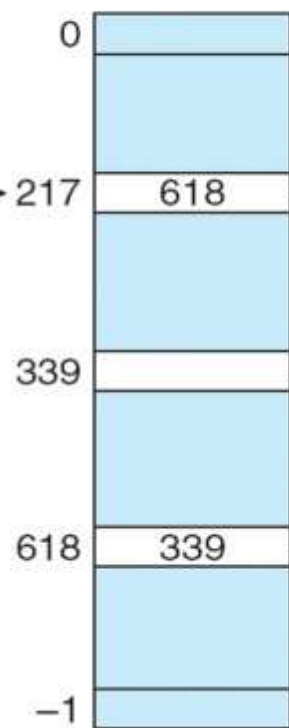
## 2.链接法举例： FAT

FAT表以盘块（簇）号为索引，下标即为簇号



文件的首地址（第一个盘块号）存放在目录中。

从目录中找到文件的首地址后，就能找到文件在磁盘上的所有存放地址。

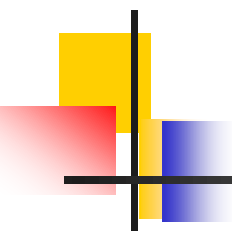


每个表项中的内容为存放文件数据的下一个盘块号。

# 文件分配表例

- 假定磁盘块的大小为1KB，对于1.2MB的软盘，其文件分配表FAT需要占用多少存储空间？若硬盘容量为200MB时，FAT需要占用多少空间？
  - 软盘大小为1.2MB，磁盘块的大小为1KB，所以该软盘共有盘块： $1.2\text{M}/1\text{K}=1.2\text{K}$ （个）
  - 又  $1\text{K} < 1.2\text{K} < 2\text{K}$ ,
  - 故1.2K个盘块号要用11位二进制表示，为了方便存取，每个盘块号用12位二进制描述，即文件分配表的每个表目为1.5个字节。
  - FAT要占用的存储空间总数为： $1.5 \times 1.2\text{K} = 1.8\text{KB}$



- 
- 假定磁盘块的大小为1KB，对于1.2MB的软盘，其文件分配表FAT需要占用多少存储空间？若硬盘容量为200MB时，FAT需要占用多少空间？
    - 若硬盘大小为200MB，硬盘共有盘块： $200\text{M}/1\text{K}=200\text{K}$
    - 又  $128\text{K} < 200\text{K} < 256\text{K}$ ,
    - 故200K个盘块号要用18位二进制表示。为方便文件分配表的存取，每个表目用20位二进制表示，即文件分配表的每个表目大小为2.5个字节。
    - FAT要占用的存储空间总数为： $2.5 \times 200\text{K} = 500\text{KB}$

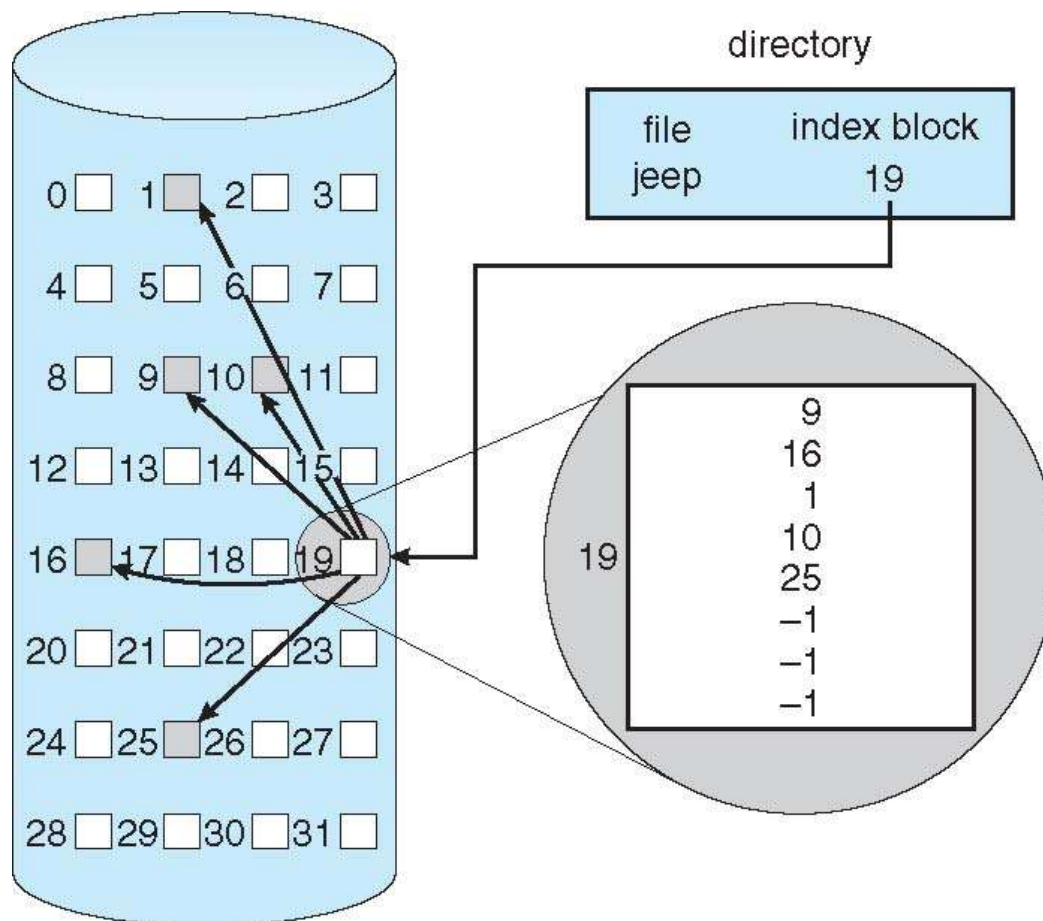


## 3. 索引分配

---

- 链接分配方式虽解决了连续分配方式中存在的问题，但又出现了新的问题：
  - 不支持随机存取
  - 链接指针要占用一定数量的磁盘空间
- 在索引分配方法中，系统为每个文件分配一个索引块，索引块中存放索引表，索引表中的每个表项对应分配给文件的一个物理块。

# 索引分配示意图





# 索引分配的特点

---

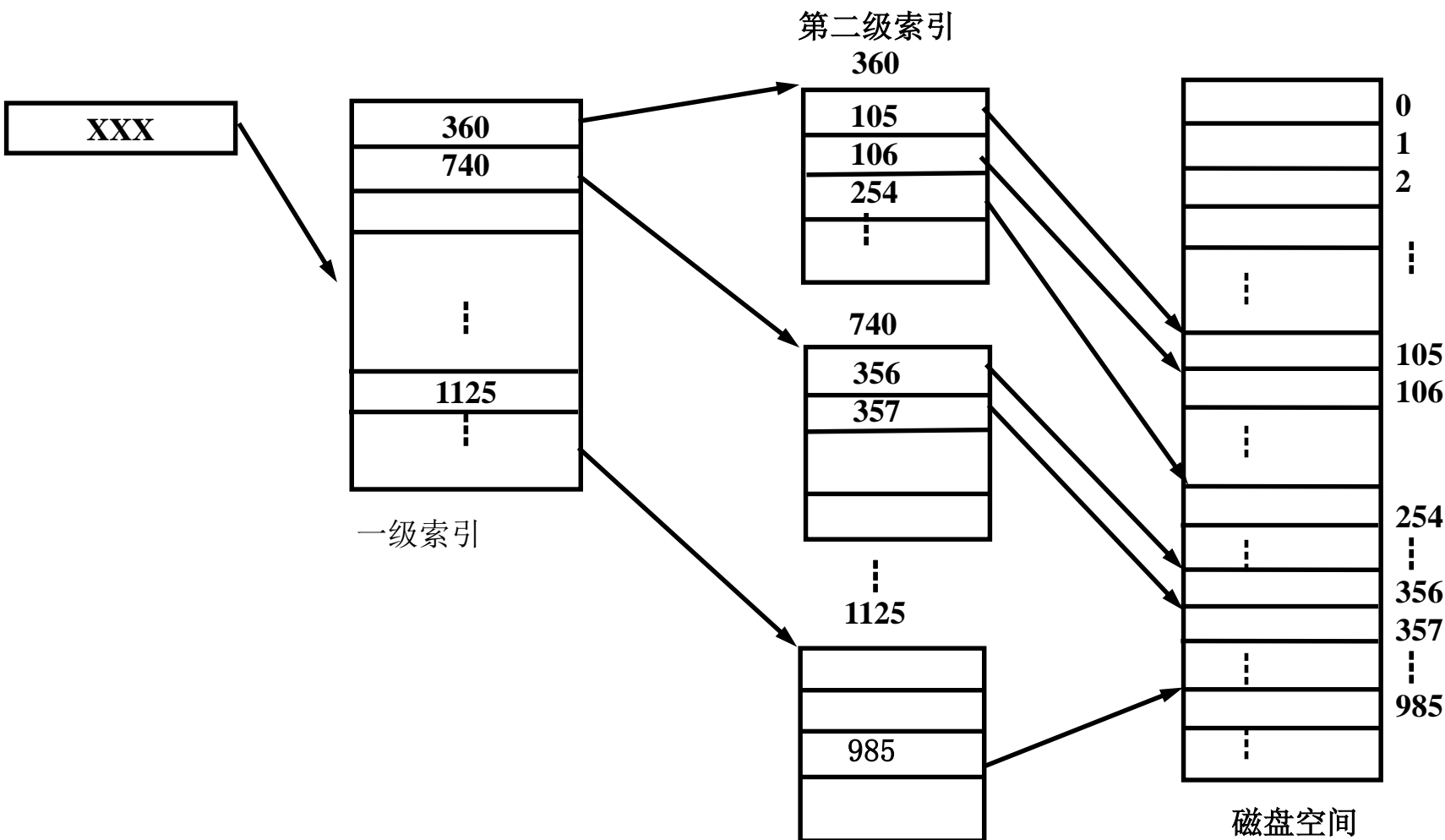
- 索引分配方法支持直接访问，不会产生外部碎片；
- 但索引块要占用一定的存储空间，存取文件需要两次访问外存。
- 采用索引分配，即使只有一个或两个指针非空，也需要分配完整的索引块，浪费空间！
  - 索引块应为多大？越小越好吗？
  - 对于大文件，应该采用什么样的机制来设计索引？



# 二级索引和多级索引

- 当文件很大，其索引表的大小超过了一个物理块时，可以将索引表本身作为一个文件，再为其建立一个“索引表”，该“索引表”是文件索引的索引，从而构成了二级索引。
- 第一级索引表的表目指向第二级索引，第二级索引表的表目指向文件信息所在的物理块号。以此类推可再逐级建立索引，进而构成多级索引。

# 两级索引分配示意图



# 两级索引分配允许的文件最大长度

- 在两级索引分配方式下，如果每个盘块的大小为1KB，每个盘块号占4字节，则：
- 一个索引块中可以存放：  
 $1\text{KB}/4\text{B}=256$ 个盘块号
- 两级索引最多可以存放的盘块数为：  
 $256 \times 256 = 64\text{K}$ 个盘块号
- 因此可以允许的最大文件长度为：  
 $64\text{K} \times 1\text{KB} = 64\text{MB}$

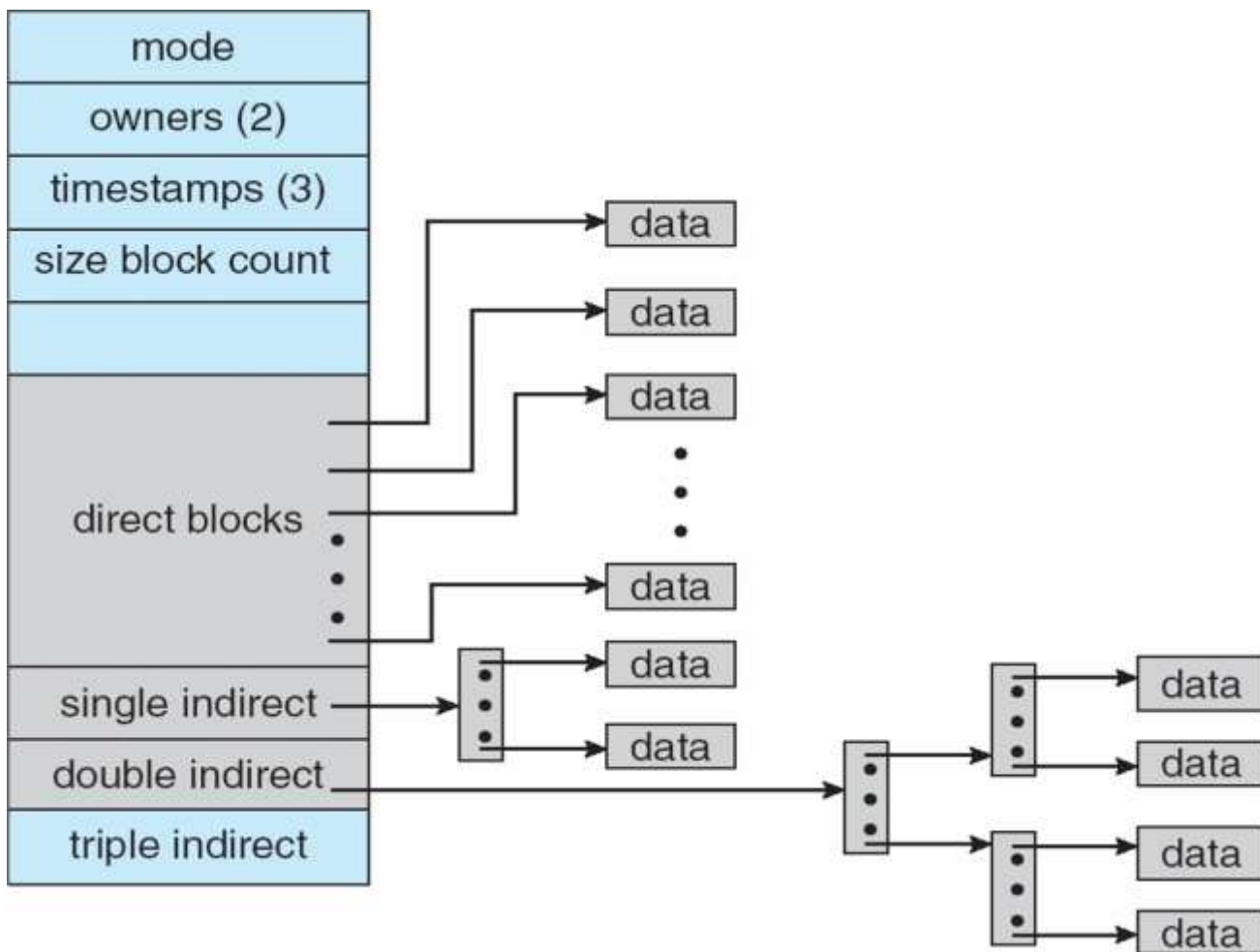


# 混合索引分配方式

- 混合索引分配方式是将多种索引分配方式相结合而形成的一种分配方式。这种方式已用于UNIX、Linux等系统中。
- 在UNIX中，共设有15个地址项，包括12个直接地址项、1个一次间接地址项、1个二次间接地址项和1个三次间接地址项。
- 假定一个盘块的大小为4KB，32位地址空间，一个盘块号占4字节。



# 混合索引方式示意图





# 寻址范围

- 直接地址：
  - 为了提高对文件的检索速度，在索引节点中建立了12个直接地址项，每个地址项中存放相应文件所在的盘块号。
  - 寻址范围： $12 \times 4\text{KB} = 48\text{KB}$
- 一次间接寻址：
  - 这里第一级存放的不是存储文件数据的盘块号，而是先将多个盘块号存放在一个磁盘块中，再将该磁盘块的块号存放在一次间接地址项中。
  - 盘块大小为4KB，若一个盘块号占4字节，则一个盘块中可以存放下： $4\text{KB} / 4\text{B} = 1\text{K}$ 个磁盘块号。
  - 寻址范围为 $1\text{K} \times 4\text{KB} = 4\text{MB}$
- 二次间接寻址
  - 寻址范围： $1\text{K} \times 1\text{K} \times 4\text{KB} = 4\text{GB}$
- 三次间接寻址
  - 寻址范围： $1\text{K} \times 1\text{K} \times 1\text{K} \times 4\text{KB} = 4\text{TB}$
- 总支持文件的大小空间 $4\text{TB}^+$



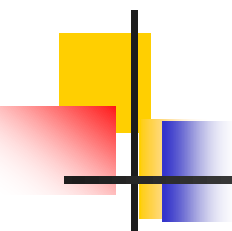
# 考研真题解析

- 设文件索引节点中有7个地址项，其中4个地址项是直接地址索引，2个地址项是一级间接地址索引，1个地址项是二级间接地址索引，每个地址项大小为4字节。若磁盘索引块和磁盘数据块大小均为256字节，则可表示的单个文件最大长度是多少？
- 大家自己练习一下！



# 曾经的期末考试试题

- 设某文件系统采用索引文件结构
  - 假定文件目录项中有10个表目用于描述文件的物理结构（每个表目占2B）
  - 磁盘块的大小和文件逻辑块大小相等，都是512B。
  - 经统计发现，此系统处理的文件具有如下特点，60%的文件其大小  $\leq 8$  个逻辑块，30%的文件其大小  $\leq 260$  个逻辑块，10%的文件其大小  $\leq 65000$  个逻辑块。
  - 设计此文件系统的索引结构，使得系统能够处理各类文件，并使读磁盘的次数理论上减少。

- 
- 从效率来看, 直接索引 > 单次索引 > 二级索引
  - 表目大小2B, 盘块大小512B, 单块可保存256条目
  - 单级索引可表示256, 二级索引可表示65536
  - 60%的文件小于8个逻辑块: 8个条目作为直接索引
  - 30%的文件小于256个逻辑块:
    - 直接索引不能表示, 只能通过一级索引, 一级索引和8个直接索引一起可表示的最大逻辑块数为 $256 + 8$ .
  - 10%的文件小于65500个逻辑块: 可设1个二级索引
  - 综上, 10个条目中设置8个直接索引, 1个一级索引, 1个二级索引



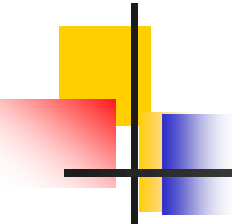
## 4. 分配方法—性能优化

- 如何选择最好的分配方法，依赖于对文件的访问方式
  - 连续分配适合于顺序访问和直接访问
  - 链接分配仅适合顺序访问，对于直接访问不适合，*Why?*
- 某些系统以连续分配支持直接访问，以链接分配支持顺序访问
  - 在创建时声明：选择以何种方式访问
- 索引分配更复杂
  - 单个区块访问要求先读入索引到内存
  - 分簇技术有利于提升吞吐率，减少CPU负载



# 性能优化 (Cont.)

- 通过在执行序列中增加数千条指令，实现优化并减少磁盘I/O
  - Intel Core i5-8250U (2017) , 1.6Ghz, 65,770 MIPS, 4 cores
    - [https://en.wikipedia.org/wiki/Instructions\\_per\\_second](https://en.wikipedia.org/wiki/Instructions_per_second)
  - HDD
    - 7200rpm, SATA 3: 100 I/Os per second
    - $65,770 \text{ MIPS} / 100 = 657.7\text{M instructions per disk I/O}$
  - SSD drives
    - Samsung SSD 850 PRO : 100,000 IOPS(读), 90,000 IOPS(写),
    - $65,770 \text{ MIPS} / 100,000 = 657.7\text{K instructions per disk I/O}$
    - <https://en.wikipedia.org/wiki/IOPS>



---

第9章 文件系统实现

## 9.5 空闲存储空间管理





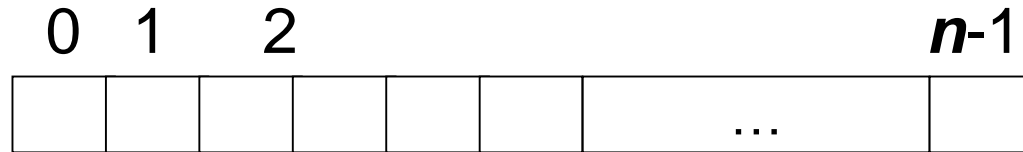
# 空闲存储空间管理

---

- 为了实现文件存储空间的分配，首先应记住空闲存储空间的情况。
- 常用的空闲存储空间管理方法有：
  - 位图法(Bit map)
  - 链接法(Linked List)
  - 分组法(Grouping)
  - 计数法(Counting)
  - 空间图法(Space Maps)

# (1)位图法: Bit vector or bit map

- 文件系统维护一个空闲空间列表去跟踪可用的区块/簇



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

- 寻找依据: 找到第一个为1的位, 其指向的块即空闲块
- 空闲块号的计算方法
  - 字长  $\times$  (值为0的字的个数) + 第一个为1的位的偏移
- 许多CPU都具有指令: 能返回在一个word中, 第一个等于1的bit偏移量, 如x86的BSF/BSR、popcnt等



# (1)位图法 (Cont.)

---

- 位图法需要额外存储

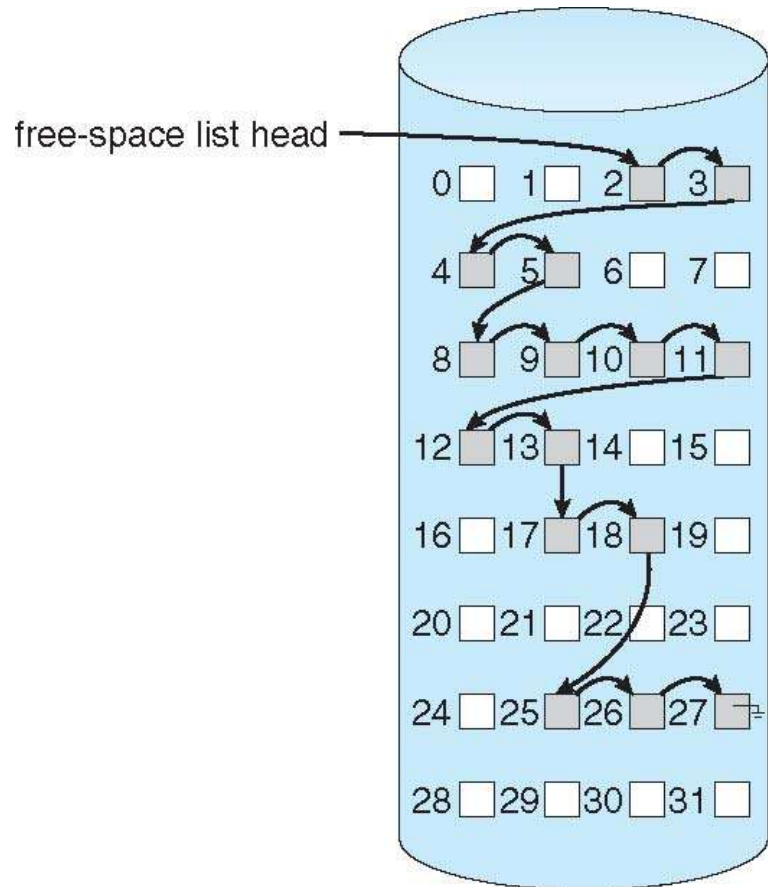
- Example:

- 块大小 4KB, 磁盘1TB
    - 则共有268,435,456块
    - 需要占用32M空间, 8k块
  - 如果以4块为1簇, 则共有67,108,864簇
    - 需要占用8M空间, 512簇
  - 当磁盘有1PB时候, 占用空间为32G

- 位图法适合连续存储法

## (2)链接法

- 无法很容易地分配连续空间
- 但不浪费空间





## (3)分组法

- 分组法

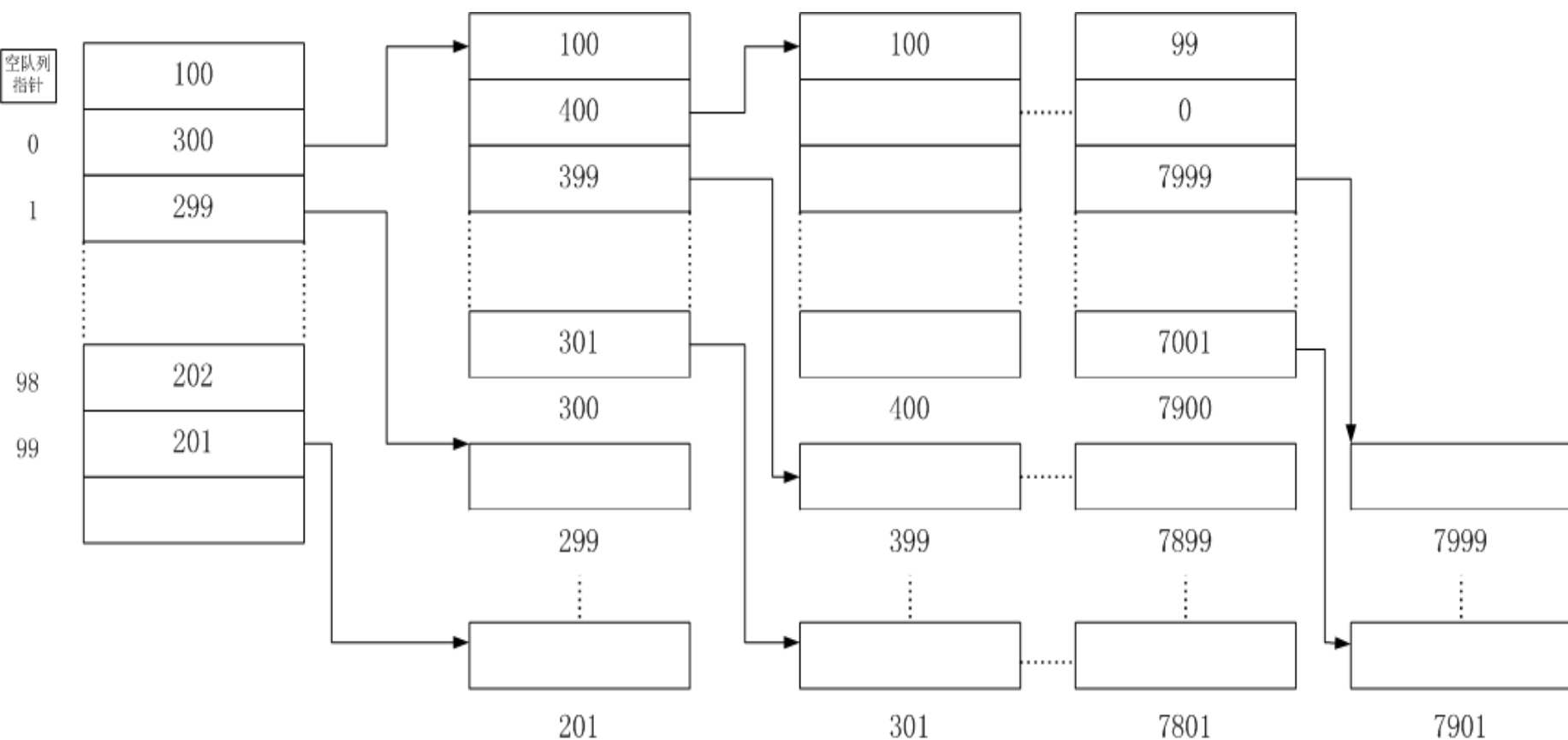
- 将n个空闲块的地址存储到第一个空闲块中，在最后一块，再添加一个指针指向另外n个空闲块的地址，如此类推
- 有利于快速找到大量空闲块

- 分组法的一种变形：空闲块成组链接法

- UNIX系统采用成组链接法对空闲盘块加以组织。
- 空闲盘块的组织：将若干个空闲盘块划归一组，将每组中的所有盘块号存放在其前一组的第一个空闲盘块号指示的盘块中，而将第一组中的所有空闲盘块号放入专用块的空闲盘块号表中。
- 在工作时，会把当前磁盘专用块复制到主存系统工作区

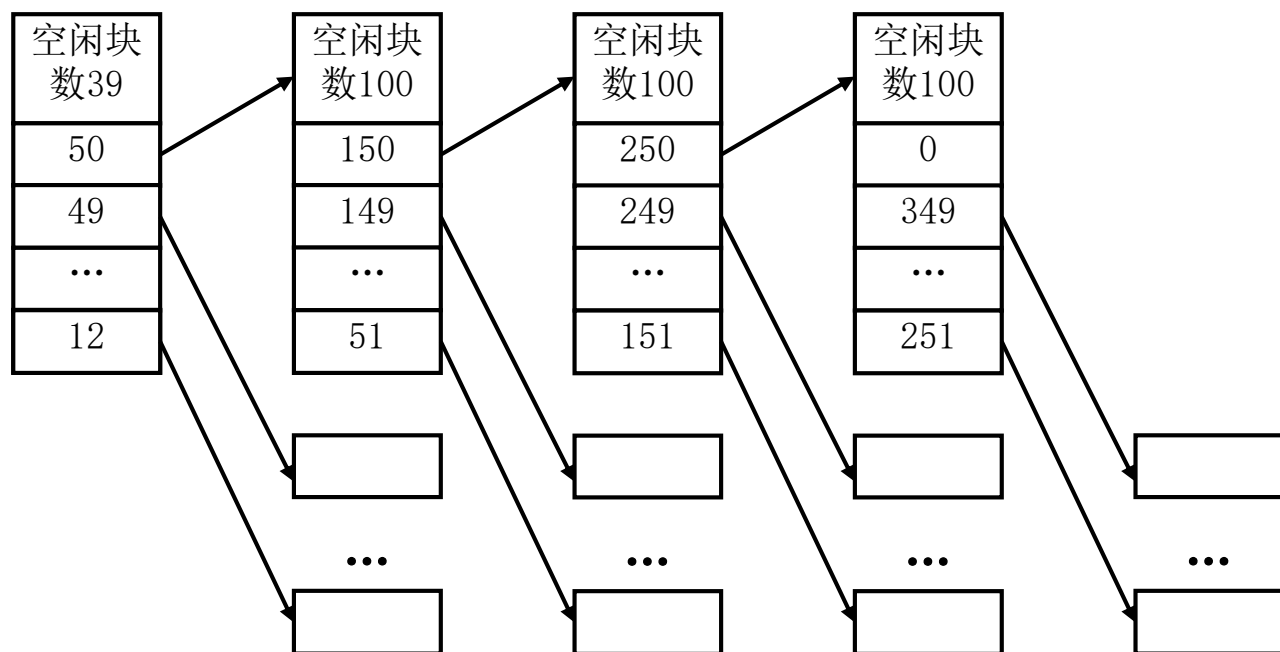
# 分组法示意图

- 100块一组，10,000个盘块
- 201—7999号存放文件



# UNIX/Linux空闲块成组连接法

- 存储空间分成512字节一块。假定文件卷启用时共有可用文件338块，编号从12至349。每100块划分一组，每组第一块登记下一组空闲块的盘物理块号和空闲总数。





## ■ 空闲盘块的分配

- 当要分配一个盘块时，首先将专用块空闲盘块号表中下一个可用盘块分配出去；
- 如果所分配盘块号是专用块中最后一个可用盘块号（记录了下一个空闲组的信息），则先将该盘块中的内容读入内存中专用块空闲盘块号表中，然后才将该盘块分配出去。

## ■ 空闲盘块的回收

- 在回收空闲盘块时，如果专用块中的空闲盘块号表未满，可直接将回收盘块的编号放入空闲盘块号表中；
- 若空闲盘块号表已满，需先将空闲盘块号表中的所有盘块号复制到新回收的盘块中，再将新回收盘块的编号放到专用块空闲盘块号表第一个位置，此块号就成了表中惟一的盘块号，空闲块计数重置为1，然后继续。





## (4)计数法

---

- 计数法

- 针对频繁、连续的分配与释放

- 不记录n个空闲块的地址，而是只记录第一个空闲块的地址，以及连续的空闲块个数
    - 有利于减少表的总长度



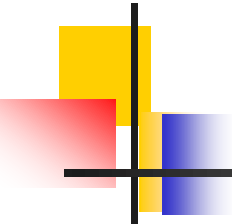
## (5)空间图法

- 空间图法：in Oracle' s ZFS
  - 问题：当在巨大的文件系统上进行大规模容量的I/O申请与删除时：
    - 如 4G数据量的 I/O操作影响有多大？
      - 当以4k为单位大小，1,048,576块 → 1,048,576次I/O
    - 当申请4G的数据，怎么办？
      - 将bit map分成小段，分段处理
    - 如删除4GB的数据，怎么办？rm -rf
      - 2M I/O, Bit map很无奈.....
    - 延迟写回
      - 建立延迟写回列表，不断进行排序和归并，等待一段时间后再进行写回操作
      - 可减轻随机释放带来的冲击
    - 更进一步
      - 空间图法



## (5)空间图法

- 空间图法：
  - 思想：如果不将延迟释放列表写回，而是把延迟释放列表本身，用来表示空闲空间会怎么样呢？
  - 将设备空间分成若干metaslabs单元，并对metaslabs进行管理
    - 一个卷会包含成百个metaslabs
  - 每个metaslab关联一个space map，描述metaslab的空闲空间
    - 使用计数法构建存储空闲块信息
  - 把对区块的操作行为，依据时间顺序，以日志的形式记录到space map中
    - 对区块的申请、释放行为，都只是对空间图的记录操作
  - 当实施真正分配时：
    - 将metaslab空间图从磁盘读入内存
    - 在内存中构造一棵按照偏移排序的高效数据结构，如平衡树
    - 将记录的日志在该数据结构上重放，归并、抵消一些操作
    - 得到空间图的精简形式，并进行操作



---

第9章 文件系统实现

## 9.6 效率和性能



# 效率和性能

- 效率依赖于
  - 磁盘分配和目录算法
    - 预先分配空的inode，并分散
    - 使文件数据块靠近inode块，减少寻道
    - 簇的大小，大簇+小簇
  - 存储于目录项中的数据类型
    - 最后写日期，决定是否备份
    - 最后访问时间，确定最后读取时间，文件被再次读取时，相应数据结构都要更新
  - 访问数据的指针大小：
    - 32位，64位，未来需求的变化
  - 定长数据结构和可变长度
    - 如进程打开文件表限定文件打开上限
    - 变长：更复杂，性能代价



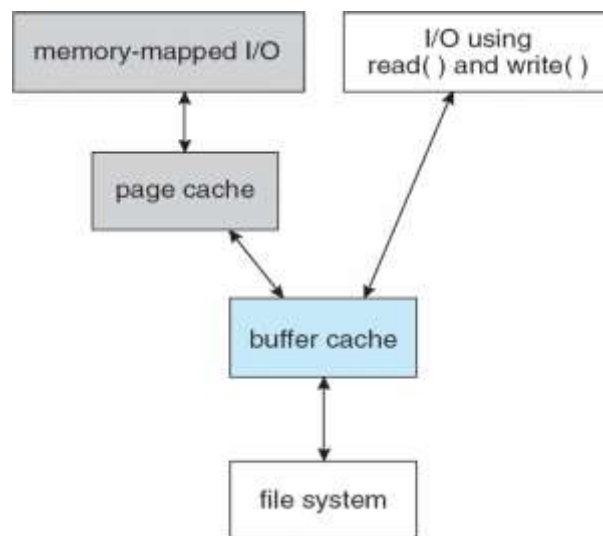
# 效率和性能

## ■ 性能

- 将数据和元数据近临保存，减少寻道
- 缓冲区缓存：将频繁使用的块放入内存独立区域
- 同步写问题：
  - 无缓冲区和缓存：必须按照接收到的请求顺序来进行写入，如对metatdata的更新
  - 异步写：更常见，可采用缓冲区，速度更快
- 和内存页面优化相关
  - 马上释放：假设以前页不会被使用，则一旦请求下一页，马上从缓存释放上一页
  - 预先读取：请求的页面和一些之后的页面可以一起读入缓存

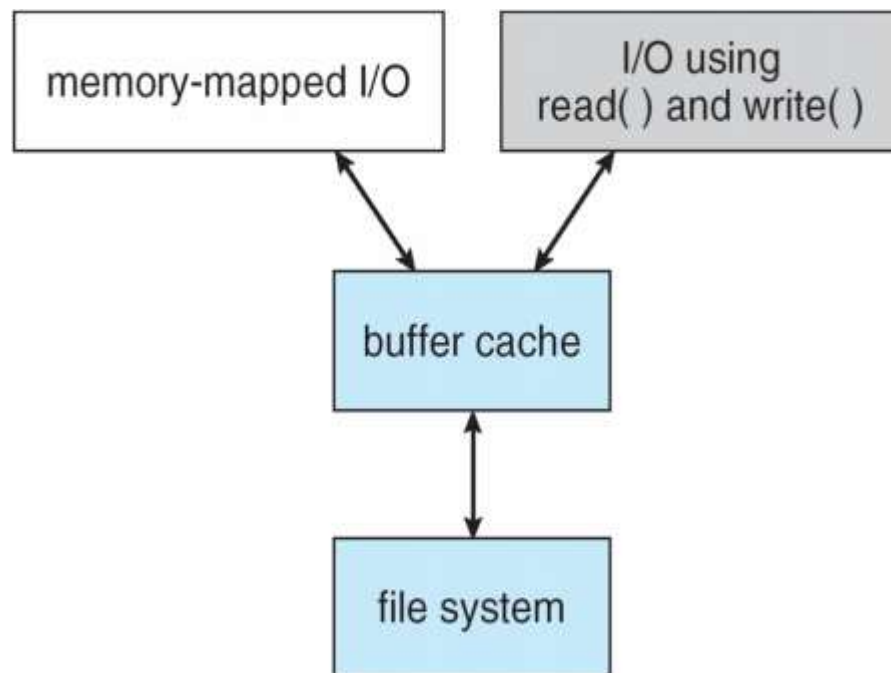
# 效率和性能—Page Cache

- 页缓存技术采用VFS，将文件数据按页面，而不是文件系统块来进行缓存
- 通过VFS来缓存，是通过虚拟内存进行访问，比文件系统要快
- Memory-mapped I/O 技术就使用页缓存来实现
- 如果标准I/O采用文件系统buffer，则出现需要双缓存问题：

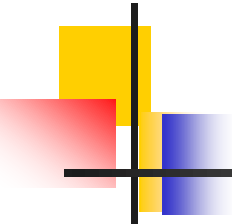


# 效率和性能—统一缓冲区缓存

- Unified Buffer Cache
  - 使用相同的页面缓存来缓存两条路径的访问
  - 新问题：哪一个缓存优先，怎么进行替换呢？







---

第9章 文件系统实现

## 9.7 文件共享



# 文件共享

---

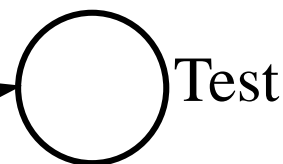
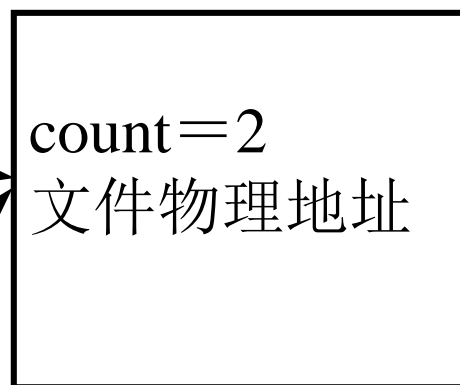
- 文件共享是指不同用户可以共同使用某文件。
- 文件共享的动机是：
  - 用户合作
  - 减少磁盘空间的开销
  - 减少文件的不一致性
- 共享语义：是文件系统对共享文件或目录冲突访问的处理方法。不同共享语义定义了对缓存一致性问题不同解决方案。

# (1)基于索引节点的共享

Wang用户文件目录

⋮	
Testw	
⋮	

索引节点

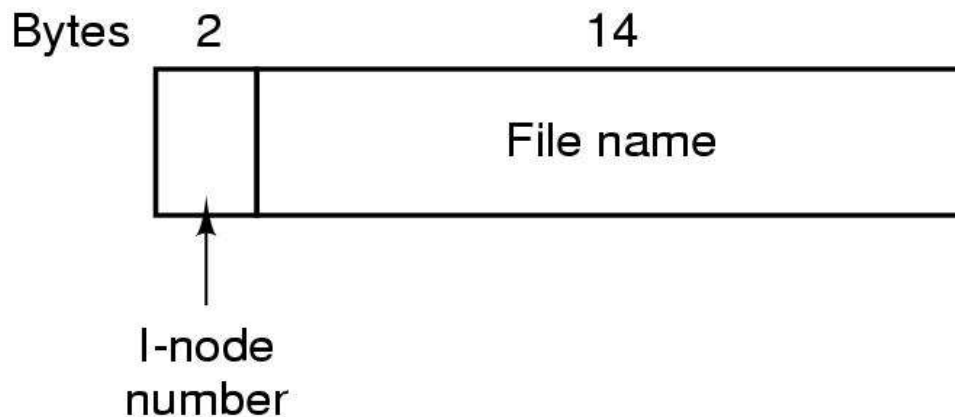


Lee用户文件目录

⋮	
Testl	
⋮	

- 索引节点：文件属性信息构成的数据结构，又称inode
- 文件名和文件属性分离
- 任何用户对文件的修改都会反映在索引节点中，其他用户可以通过索引节点存取文件。

# UNIX中FCB: inode索引节点



## 目录文件的内容

文件名 14B	i-node # 2B
.....	.....
mytest.c	56
.....	.....

## 磁盘上的i-node表

i-node #	文件属性
56	存取权限, 大小, 文件主, 建立/修改日期, 磁盘地址 等
.....	.....

64B



# inode基本信息

---

- inode包含文件的元信息：
  - Size, User ID, Group ID, Access权限
  - 文件的时间戳: Change, Modify, Access
  - Links 链接数, 即有多少文件名指向这个inode
  - Inode 文件数据block的位置
  - Blocks 块数
  - IO Blocks 块大小
  - Device 设备号码



# inode大小

---

- 格式化
  - OS将硬盘分成两个区域：数据区； inode区（表）
- inode大小:128-256B
  - 假定在一块1GB的硬盘中，每个inode节点的大小为128字节，每1KB就设置一个inode，那么inode table的大小就会达到128MB，占整块硬盘的12.8%
- inode的数量：
  - 当创建众多小文件时，用完inode区空间
  - 显示磁盘无空间，而实际磁盘空间尚剩余很多



# inode号

---

- 每个inode都有一个号码
  - OS用inode号码来识别不同的文件
  - Unix/Linux系统内部不使用文件名，而使用inode号码来识别文件
  - 每个目录项：文件名+该文件名对应的inode号
- 文件查找
  - 系统根据文件名找到对应的inode号；
  - 通过inode号码，获取inode信息；
  - 根据inode信息找到文件数据的block



# inode的好处是什么？

- 引入索引结点

- 按文件名检索目录文件时，只用到了文件名。当找到该文件名时，才需要它的其它描述信息。
- 所以在把存放该目录文件的盘块从外存调入内存进行比较时，应使一个盘块中包含尽量多的文件名，以**减少启动磁盘次数， 加快按名存取的速度。**





# i-node的好处是什么?

- 例：设物理块大小为512B，某目录下有128个文件。
  - 若采用简单FCB，设FCB占64B
    - 则每物理块能容纳  $512/64=8$  个FCB
    - 该目录文件需占  $128/8 = 16$  块
    - 查找一个文件的平均访盘次数为：  
 $(1+16) / 2 = 8.5$  次。
  - 若采用inode，目录项占16B(文件名+inode号)，inode部分64B
    - 每物理块能容纳  $512/16=32$ 个目录项
    - 每个物理块容纳  $512/64=8$ 个inode
    - 该目录文件需占  $128/32 = 4$  块，inode部分需占  $128/8=16$  块
    - 查找一个文件的平均访盘次数为：  
 $(1+4) / 2 + 1 = 3.5$  次。



# Linux中基于inode的共享

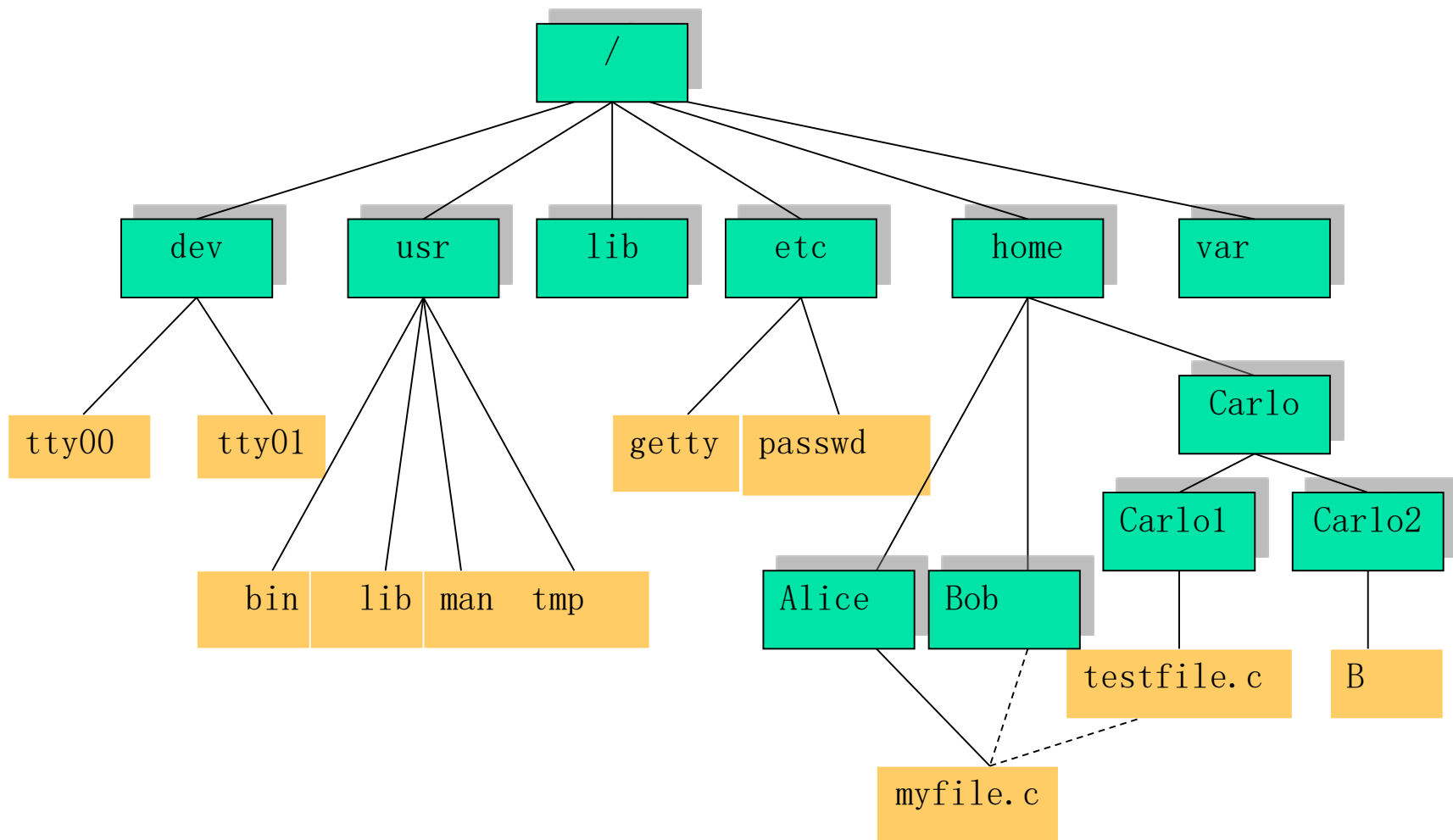
- 系统调用形式为：

`char * oldnamep, * newnamep;`

`link (oldnamep, newnamep);`

- ① 检索目录找到oldnamep所指向文件的索引节点inode编号。
- ② 再次检索目录找到newnamep所指文件的父目录文件，并把已存在文件的索引节点inode编号与别名构成一个目录项，记入到该目录中去。
- ③ 把已存在文件索引节点inode的连接计数i\_nlink加“1”。

# Linux层次目录结构





# Linux中基于inode的共享

- 链接实际上是共享已存在文件的索引节点inode，完成链接的系统调用：
  - `link( "/home/Alice/myfile.c" , " /home/Bob/myfile.c" );`
  - `link( "/home/Alice/myfile.c" , " /home/Carlo/Carlo1/testfile.c" );`
  - 执行后，三个路径名指的是同一个文件：
    - `/ home/Alice/myfile.c`
    - `/ home/Bob/myfile.c`
    - `/ home/Carlo/Carlo1/testfile.c`



# Linux中基于inode的共享

- 文件解除链接调用形式为：  
    `unlink (namep)`
  - 解除链接与文件删除执行的是同一系统调用代码。
  - 删除文件，是从文件所有者角度讲的
  - 解除文件链接，是从共享文件的其他用户角度讲的。
  - 都要删去目录项，把`i_nlink`减“1”，不过，只有当`i_nlink`减为“0”时，才真正删除文件。

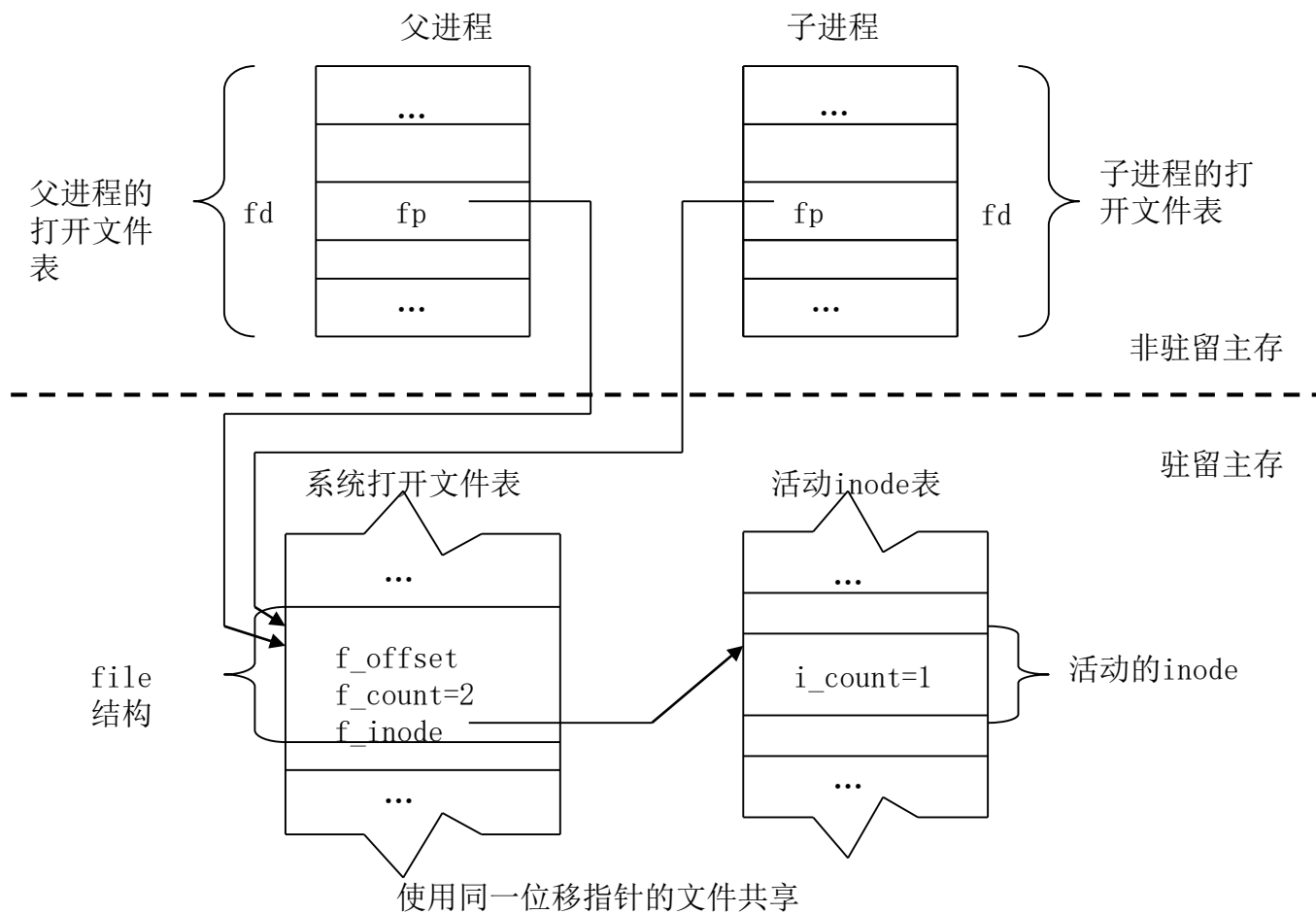


## (2)文件的动态共享

- 文件动态共享是系统中不同的用户进程或同一用户的不同进程并发访问同一文件。
- 这种共享关系只有当用户进程存在时才可能出现，一旦用户的进程消亡，其共享关系也就自动消失。
- 文件的每次读写由一个读/写位移指针指出要读写的位置。现在的问题是：**应让多个进程共用同一个读/写位移，还是各个进程具有各自的读写位移呢？**

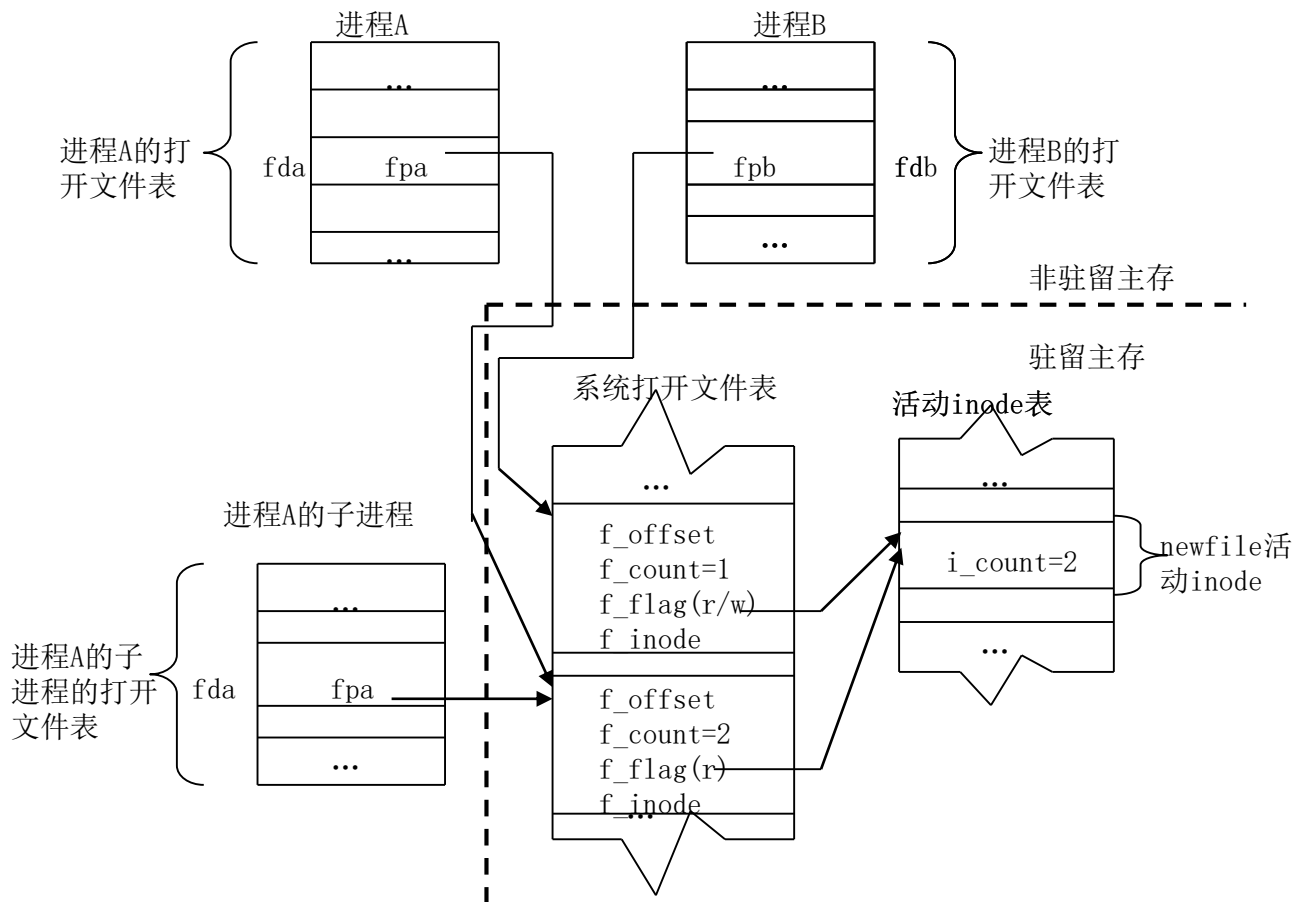
# 共享读写位移

- **同一用户父、子进程**协同完成任务，使用同一读/写位移，同步地对文件进行操作。



# 不共享位移

- **多用户共享文件**，每个希望独立地读、写文件，这时不能只设置一个读写位移指针，须为每个用户进程分别设置一个读、写位移指针。

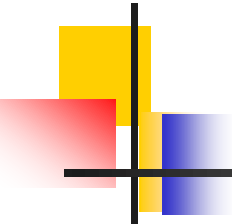


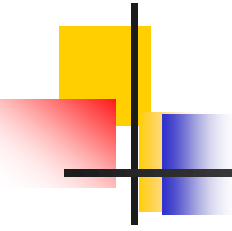


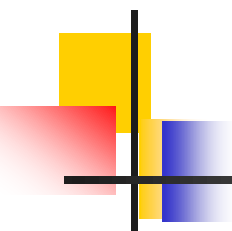


## (3)利用符号链接实现文件共享

- 操作系统可支持多个物理磁盘或多个逻辑磁盘(分区), 那么, 文件系统是建立一棵目录树还是多棵目录树呢?
  - Windows采用将盘符或卷标分配给磁盘或分区, 并将其名字作为文件路径名的一部分。
  - UNIX/Linux的每个分区有自己的文件目录树, 当有多个文件系统时, 可通过安装的办法整合成一棵更大的文件目录树。

- 
- 问题：Linux系统中每个文件对应一个inode，编号是惟一的，但两个不同的磁盘或分区可能都含有相同inode号对应的文件，也就是说，整合的目录树中，inode号并不惟一地标识一个文件，
  - 办法：对于link这样系统调用，拒绝创建跨越文件系统的硬链接。

- 
- 基于符号的链接，又称软链接，符号链接是一种只有文件名，不指向inode的文件
  - 符号链接共享文件的实现思想：
    - 用户A目录中建立形如 $afile \rightarrow bfile$ ，实现A的目录与B的文件的链接。其中只包含被链接文件bfile的路径名而不是它的inode号。

- 
- 当用户A要访问被符号链接的用户B的文件bfile，且要读“符号链接”类文件时，被操作系统截获，它将**依据符号链接中的路径名**去读文件，于是就能实现用户A使用文件名afile对用户B的文件bfile的共享。
  - 优点：能用于链接计算机系统中不同文件系统中的文件，可链接计算机网络中不同机器上的文件，此时，仅提供文件所在机器地址和该机器中文件的路径名。
  - 缺点：搜索文件路径开销大，需要额外的空间查找存储路径。

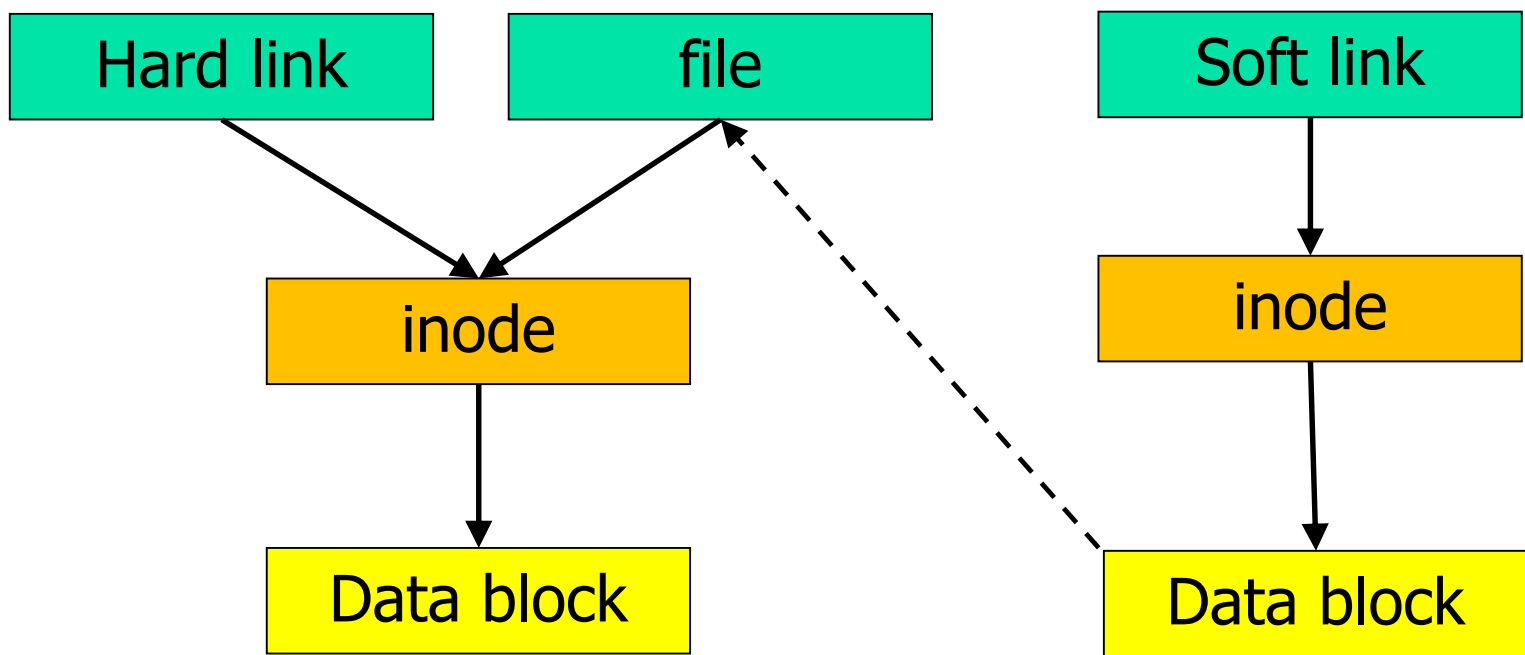


# Linux中的软链接

- 软链接是个普通文件
  - 文件内容特殊，存放另一个文件的路径名的指向
  - 有自己的inode以及数据块
  - 有自己的文件属性和权限
  - 创建软链接时，原文件的i\_nlink不变化
  - 删除软链接并不影响被指向的文件，但若被指向的  
原文件被删除，则相关软连接被称为死链接  
(dangling link)

# 硬链接与软链接

- 硬链接是通过索引节点来链接，其结果是多个文件名指向同一索引节点，即一个文件拥有多个有效路径名
- 软链接类似于快捷方式，实际上只包含要链接的文件路径和文件名





# 硬链接与软链接

- 软硬链接对引用数的影响
  - 创建硬链接时，引用数+1
  - 创建软连接时，由于只拷贝文件路径和文件名，引用数不变
- 若要删除一个共享文件，必须判别是否有多个用户共享该文件
  - 删除原文件，对硬链接无影响，符号链接失效
- 文件真正删除的条件是与之相关的硬链接文件均删除

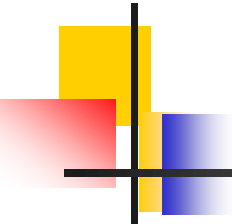


# 考研真题解析

---

- 设文件F1的当前引用计数值为1，先建立F1的符号链接（软链接）文件F2，再建立F1的硬链接文件F3，然后删除F1。此时，F2和F3的引用计数值分别是
- A. 0、1    B. 1、1    C. 1、2    D. 2、1





---

第9章 文件系统的实现

## 9.8 文件的备份转储和恢复



# 文件的备份转储和恢复

---

- 为了能在各种意外情况下减少或避免文件系统遭到破坏时的损失，常用的方法是定期转储。
- 备份转储的方法有两种：
  - 全量转储
  - 增量转储



# 全量转储

---

- 全量转储：
  - 定期将文件存储器中的**所有文件备份转储**到某存储介质上
  - 一旦系统出现故障破坏了文件信息，便可以将最近一次转储的内容复制到文件系统中去，使系统恢复到上次转储时的状态。
- 全量转储的不足：
  - 转储期间应停止对文件系统进行其他操作，转储时间长。



# 增量转储

---

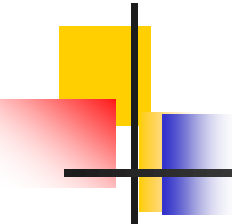
- 增量转储：
  - 将上次转储以来**修改过的文件和新增加的文件**转储到某存储介质上。
  - 增量转储能使系统遭到破坏后，恢复到数小时前文件系统的状态，从而使得所造成的损失减到最小。
- 在实际工作中，两种方法要配合使用，根据实际情况，确定全量转储的周期和增量转储的时间间隔。



# 文件系统的恢复过程

---

- 一旦系统发生故障，文件系统的恢复过程大致如下：
  - 从最近一次全量转储中装入全部系统文件
  - 从近到远从增量转储盘上恢复文件。同一个文件只恢复最近一次转储的副本。



---

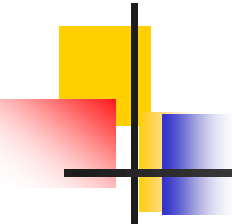
第9章 文件系统的实现

## 9.9 基于日志结构的文件系统



# 基于日志结构的文件系统

- 基于日志结构的文件系统
  - Log structured (or journaling) file systems
  - 以事务形式记录每一个对文件系统metadata的更新
- 所有的事务都被写入日志
  - 事务：一旦写入日志，即认为被提交
  - 文件系统中各种操作可能因故障被中断
- 事务被异步写入文件系统结构
  - 当文件系统结构被修改后，事务被从日志中删除
  - 如果文件系统崩溃，保留的事务需要再次执行
  - 对于没有提交的事务，即事务提交中被中断，这类修改必须撤回。



---

第9章 文件系统的实现

## 9.10 文件系统举例

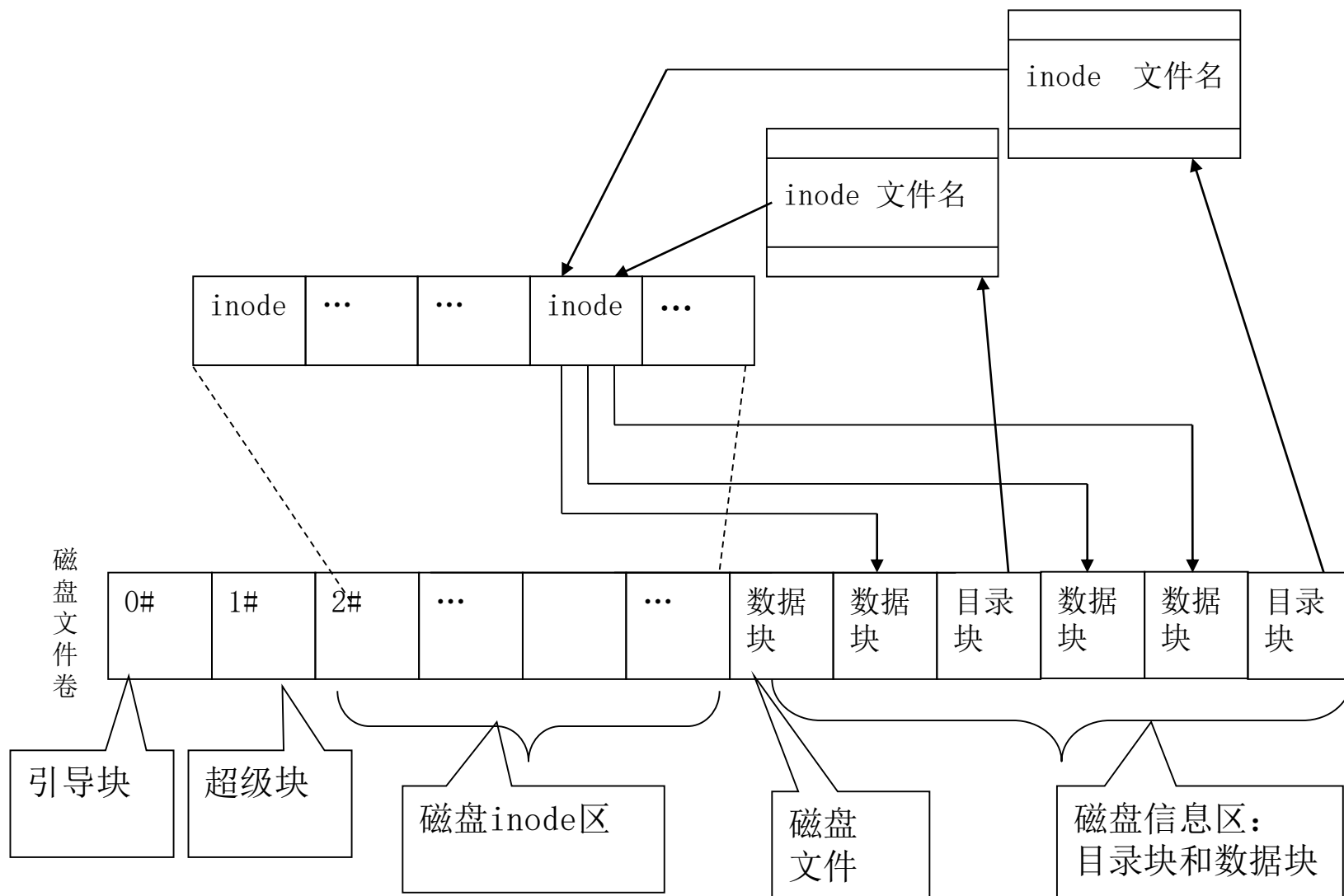




# 文件系统举例

- 超级块：占用1#号块
  - 存储了文件系统的结构和管理信息，如inode表所占盘块数，空闲盘块数等等信息
- 索引节点区：2# ~ k+1#块
  - 存放inode表，每个文件都有一个inode
- 数据区：k+2# ~ n#为数据块
- 两个重要数据结构：
  - 用户打开文件表：PCB中维护
  - 系统打开文件表：系统管理，用于共享

# 目录项、inode和数据块的关系



# 文件系统内部结构

文件描述符fd

file的指针fp

...

用户打开文件表  
files\_struct

系统打开文件表  
file\_struct

主存活动  
inode表

一个打开  
文件的file

f\_flag  
f\_count  
...  
f\_inode

i\_number  
i\_count  
...  
i\_addr[40]

活动inode

i\_number  
i\_count  
...  
i\_addr[40]

活动inode

主存

磁盘

磁盘文件卷

0#

1#

2#

...

...

...

...

引导块

超级块

磁盘inode区

磁盘文件

磁盘信息区：  
目录块和数据块