



n 计算机有三个重要的功能

- | 信息的处理 CPU管理、内存管理
- | 信息的存储 文件管理
- | 信息的输入输出



# 第10章 文件系统接口

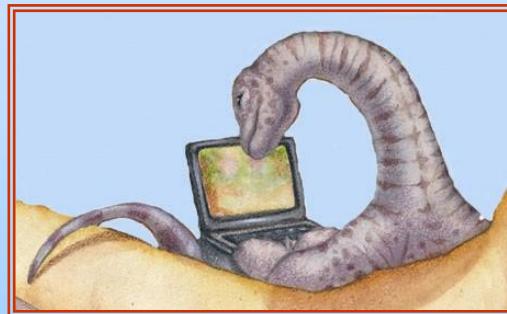




# 目录

1. 文件
2. 逻辑文件及其访问方法
3. 文件目录
4. 目录结构

# 1、文件

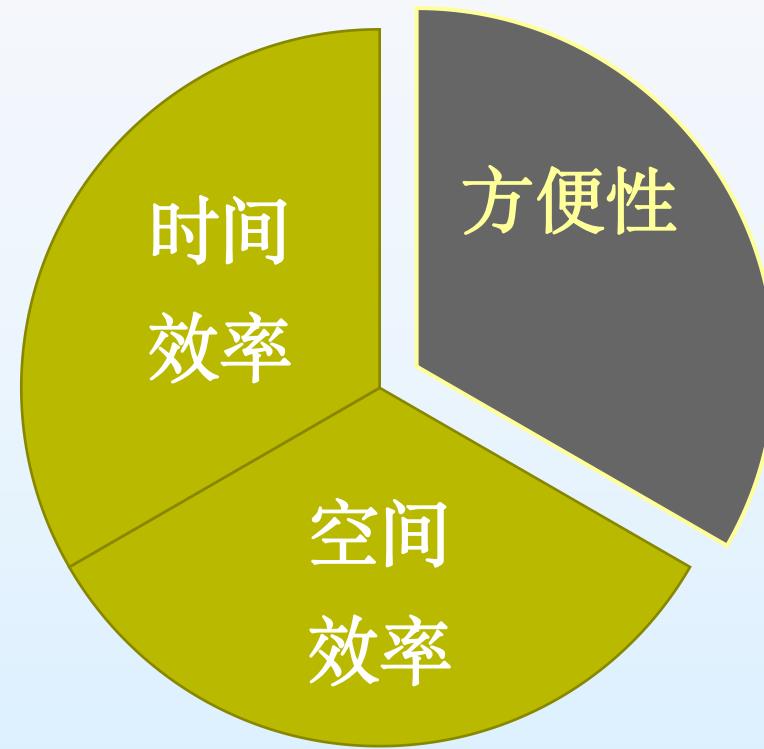




5

# 内容

- n 文件概念
- n 文件属性
- n 文件操作
- n 文件类型
- n 文件结构





# 文件概念

## n 文件

- | 计算机中信息存储的基本组织形式
- | 具有文件名的相关信息集合



## n 文件名

- | 按名存取：文件名 → 存储位置
- | 文件名由一串ASCII码或(和)汉字构成
- | 名字长度
  - 8.3规则：文件名8个字符，类型3个字符，之间有“.”分割
  - 长文件名：可以最多输入255多个字符作为文件名
- | 文件名可能大小写敏感





# 文件结构

- n 文件结构是指文件内信息的组织方式
- n 目的：便于程序理解文件内容
- n 操作系统和应用程序决定了文件的结构
  - | pdf文件的结构由pdf阅读器决定
  - | 可执行文件的结构由操作系统决定
- n 常用文件结构
  - | 无结构：文字流、字节流等
  - | 简单记录结构：线性、固定长度、可变长度等
  - | 复杂结构：格式化文档、多媒体文件等





n 在文件的各种属性中，用户最关注的是（）

- A. 文件名
- B. 文件长度
- C. 文件存放位置
- D. 文件创建时间



# 文件类型 – 扩展名

- n 文件类型一般由扩展名决定
- n 文件扩展名
  - | 也称文件后缀名
  - | 标识文件类型的一种机制
  - | 扩展名跟在主文件名后面，由一个分隔符“.”分隔

file type	usual extension	function
executable	exe, com, bin or none	read to run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rrf, doc	various word-processor formats
library	lib, a, so, dll, mpeg, mov, rm	libraries of routines for programmers
print or view	arc, zip, tar	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, rm	binary file containing audio or A/V information





# 文件属性

- n 文件名：唯一，便于记忆
- n 文件位置：设备上文件位置的指针
- n 文件类型：文件的格式
- n 文件大小：文件当前大小
- n 保护：读、写、执行等访问控制信息
- n 时间、日期和用户标识：保护、安全和使用跟踪的数据
- n .....

文件属性保存在目录中





# 文件操作

- n 创建文件
- n 写文件
- n 读文件
- n 在文件内重定位
- n 删除文件
- n 截断文件
- n 打开文件
- n 关闭文件





# 为什么要有打开文件操作?

## n 需要数据结构

- | 打开文件表: 跟踪打开文件
- | 文件指针: 指向最后一次读写的位置, 每个进程1个
- | 打开文件计数器: 打开文件次数 (调用open次数)
- | 文件存储位置: 文件存放在存储设备上的位置信息
- | 访问权限: 每个进程的访问权限

## n 优点

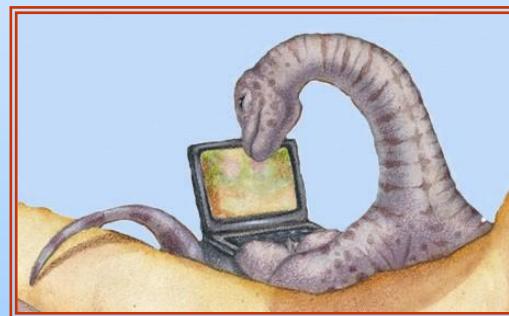
- | 方便文件共享
- | 提高文件存取效率





10.13

## 2、逻辑文件及其访问方法





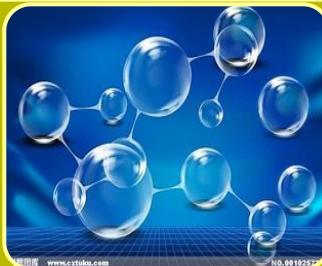
# 内容

- n 逻辑文件
- n 顺序文件
- n 直接文件
- n 索引文件





# 文件访问



## 逻辑文件

- 文件内容组织方式
- 面向用户



## 目录

- 文件组织方式
- 文件属性



## 物理文件

- 文件存储方式
- 面向系统





# 逻辑文件

- 逻辑文件是指文件呈现在用户面前的组织结构
- 又称为文件逻辑结构
- 逻辑文件决定了文件访问方法

苹果：3 元/斤  
香蕉：5 元/斤

苹果：3 元/斤|香蕉：5 元/斤

文本文件

	A	B
1	苹果	3元/斤
2	香蕉	5元/斤

	A	B
1	水果	单价 (元/斤)
2	苹果	3
3	香蕉	5

Excel文件





# 文件访问方式

## n 顺序访问

- | 最简单的访问方式
- | 文件信息按照存放顺序，一个记录一个记录地依次访问
- | 顺序文件
- | 典型存储设备：磁带



## n 直接（随机）访问

- | 可以直接定位到文件的某条记录进行访问
- | 直接文件
- | 典型设备：磁盘

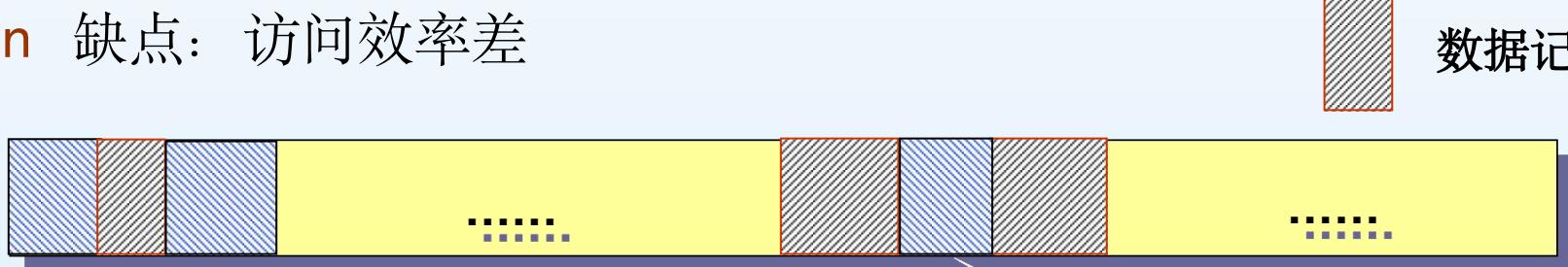




# 顺序文件

- n 顺序文件由一系列不等长记录按照某种顺序（一般是写入时间）排列形成
- n 访问方式：顺序访问
- n 依次访问数据，不能直接跳转到文件的指定位置
- n 优点：记录存储紧凑，节省存储空间
- n 缺点：访问效率差

数据记录



要读写记录  $n$ :

read record 0  
read record 1  
.....  
read record  $n-1$   
read/write record  $n$

无法直接定位  
到第 $n$ 个记录



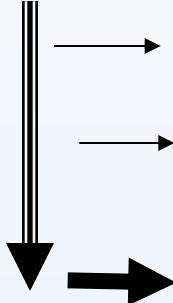


n 存放在磁盘上的顺序文件可以直接访问。



# 顺序访问例子

记录长度不等



0	王一, 男, 00计算机1班, 20, 0013001.
1	赵尔, 女, 00电子技术1班, 20, 0014005.
...	.....
i-1	李立理, 男, 00英语1班, 18, 0019034.
i	黄湖, 男, 00英语1班, 18, 0019035.
i+1	
...	.....

记录平均长度: 40B      1M条记录长度: **40MB**

访问第1万条记录: 必须把0到9999个记录读入后才知道第1万条记录首址

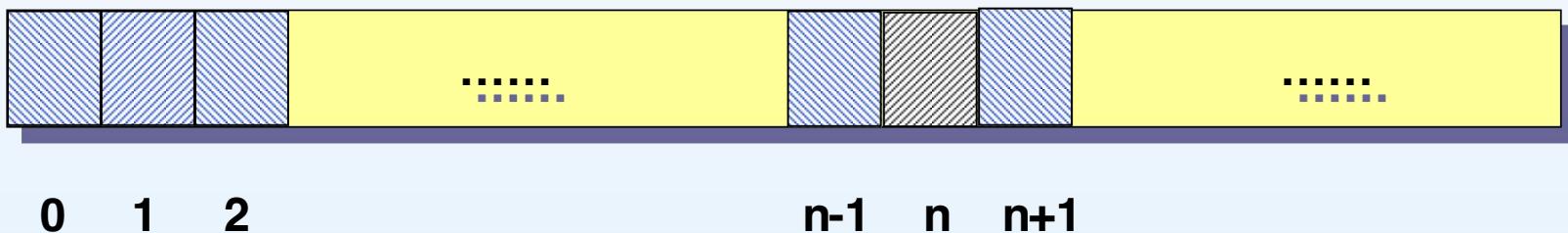
需要读入数据:  $40 * 10, 001 = \text{400040B}$





# 直接 (随机) 文件

- 直接文件一般组织为记录等长文件
  - 访问方式：直接（随机）访问
  - 直接通过计算得到需要读写记录的位置，直接跳转进行文件读写



- | Record Length=R
  - | Read record n :  $\text{ptr}=n*R$ , read
  - | Read record j :  $\text{ptr}=j*R$ , read

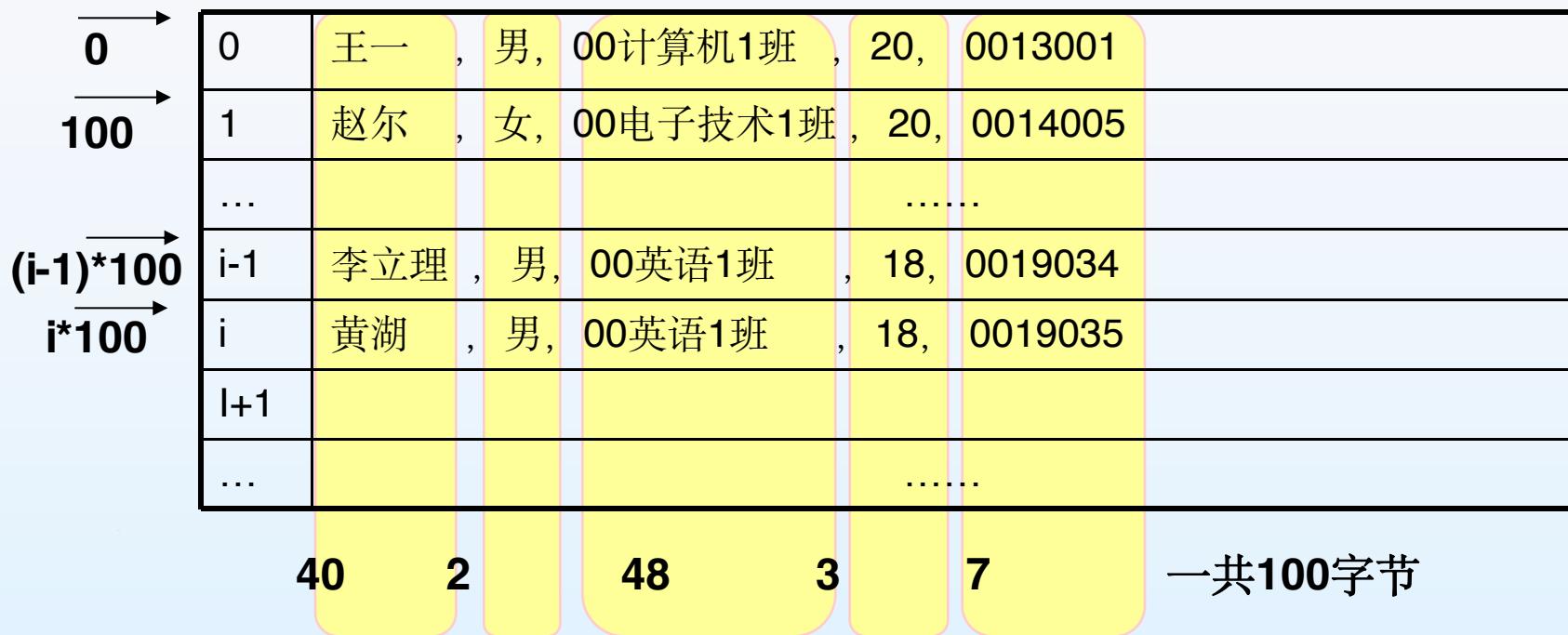
n 优点：访问效率高，可以直接定位到某条记录

n 缺点：浪费存储空间





# 直接访问例子



记录长度: 100B 1M条记录: **100MB**

访问第1万记录:

计算得到第1万条记录的首址为 $10000 * 100 = 1000000$

从1000000处开始读入100B

需要读入数据: **100B**





# 索引文件

## n 顺序文件

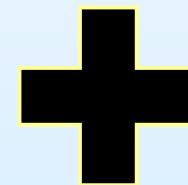
- | 检索效率低
- | 存储空间省

## n 直接文件

- | 检索效率高
- | 存储空间费

是否可以具有顺序文件和直接文件的优点?

顺序文件  
的存储形式



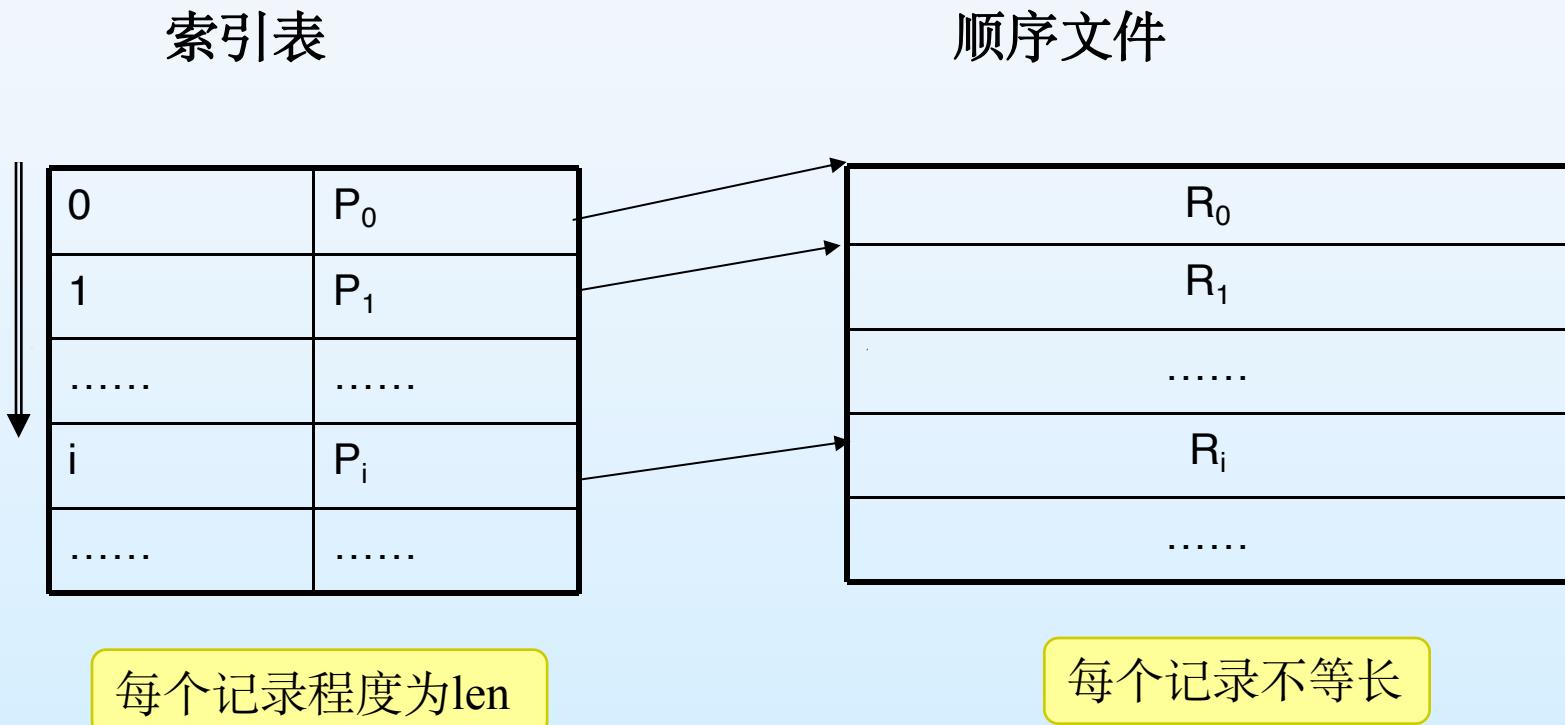
直接文件  
的检索方式





# 索引文件

n 基本方法：为顺序文件建立索引表

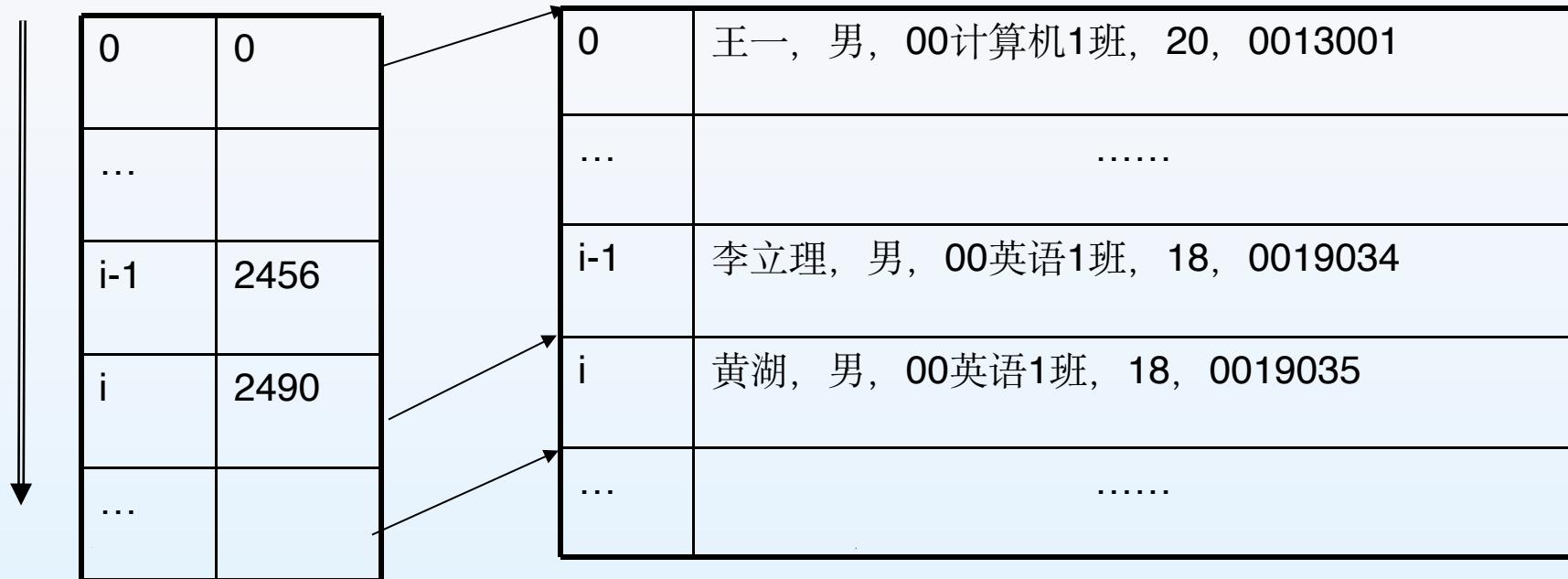




n 索引文件中，索引表的表项一般等长



# 例子



记录平均长度: 40B 索引表项大小: 4B 1M条记录长度: **44MB**

访问第1万条记录:

- 1) 计算得到第1万条记录的索引项在索引表中首址:  $10000 * 4 = 40000$
- 2) 从索引表地址40000处读入4个字节, 内容为第1万条记录在顺序文件中的首址P
- 3) 从顺序文件地址P处读入40个字节 (假如第1万条记录长度为40B)

合计读入: **4+40=44B**



# 三种逻辑文件性能对比

文件访问方式，即逻辑文件是影响文件访问效率的因素之一！！！

	存储容量	访问数据
顺序文件	40MB	400040
直接文件	100MB	100B
索引文件	<b>44MB</b>	<b>44B</b>

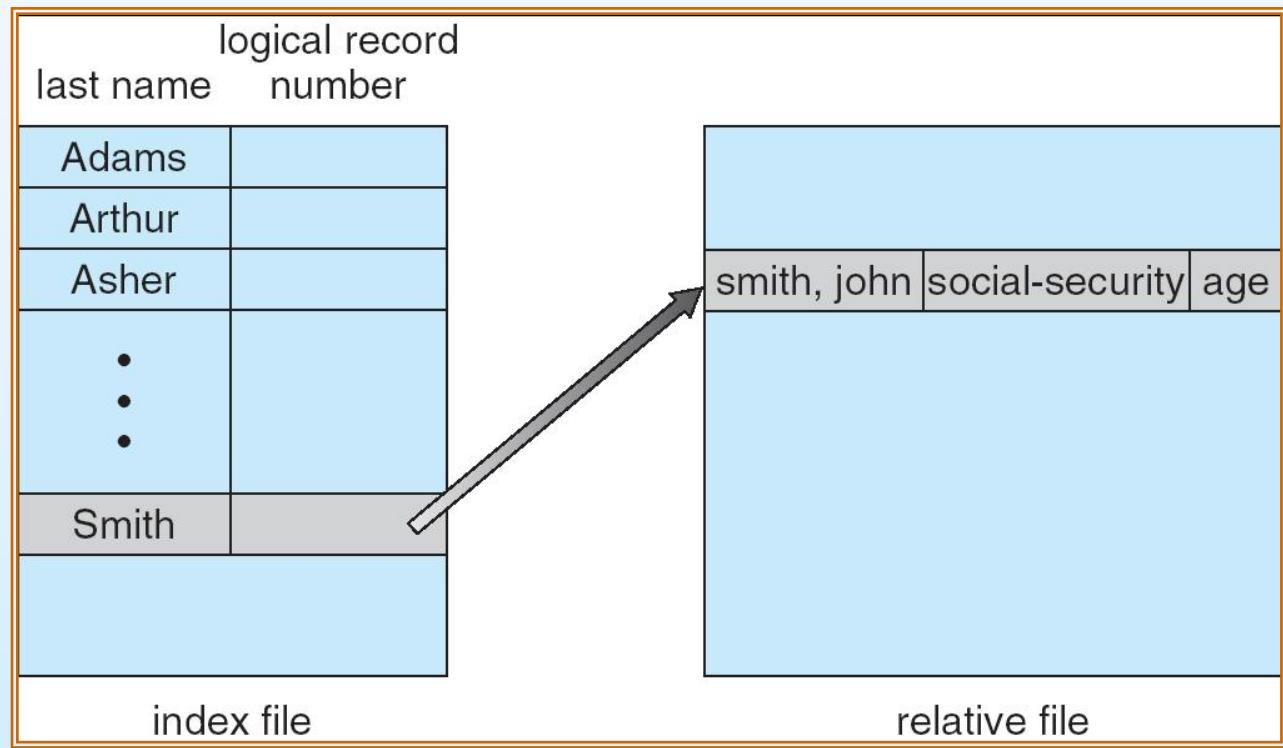
索引文件存储容量接近  
顺序文件，访问速度接  
近直接文件





# 另外一种索引文件

- n 根据某个检索项作索引
- n 如： last name



- 
- n 最节省存储空间的文件组织方式是 ( ) 。
- A. 直接文件
  - B. 索引文件
  - C. 顺序文件
  - D. 哈希文件



10.31



### 3、文件目录





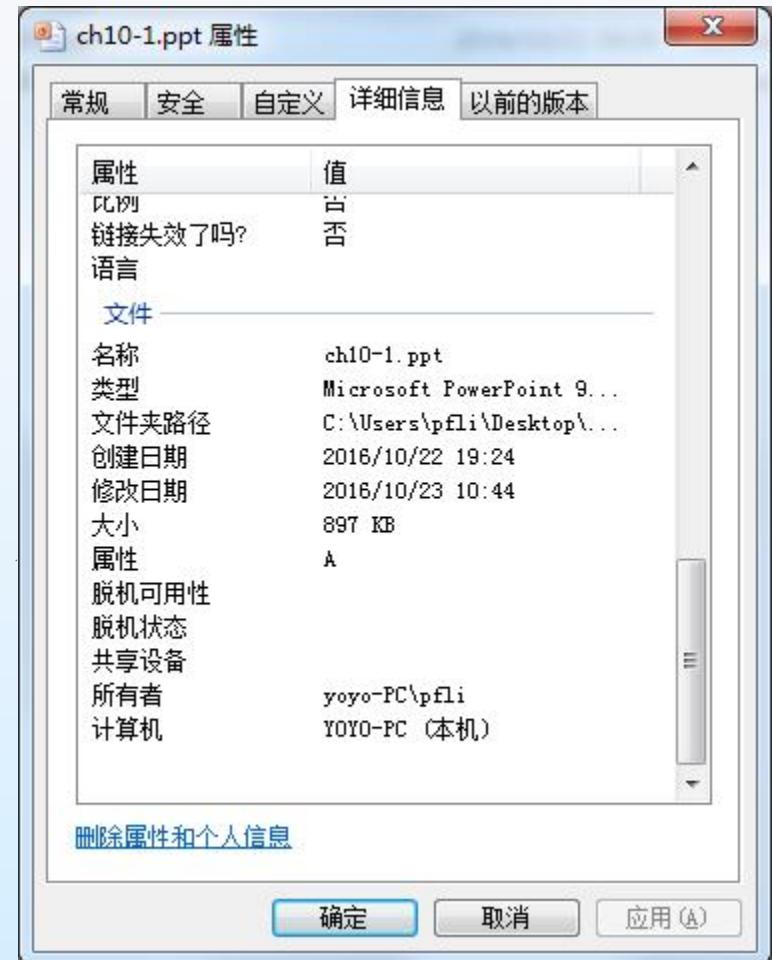
# 内容

- n 文件控制块
- n 目录项和目录
- n 目录组织
- n 目录访问
- n 目录性能
- n 目录保护



# 文件控制块 (FCB)

- n File Control Block (FCB)
- n 存放操控文件所需的各类文件属性信息
  - | 文件名
  - | 文件长度
  - | 创建时间
  - | 存放位置
  - | 访问控制权限
  - | .....
- n 类似一个索引项





# 目录项和目录

## n 目录

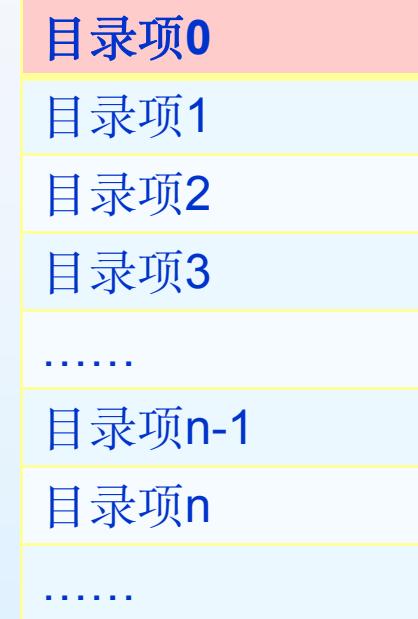
- | 包含着所有文件信息的节点集合
- | 根据文件名检索文件的桥梁
- | 目录项的有序集合

## n 目录项

- | 存放一个文件的各类属性
- | 有的系统等同于文件控制块

## n 目录文件

- | 目录组织形式
- | 目录作为一个文件存在于文件系统



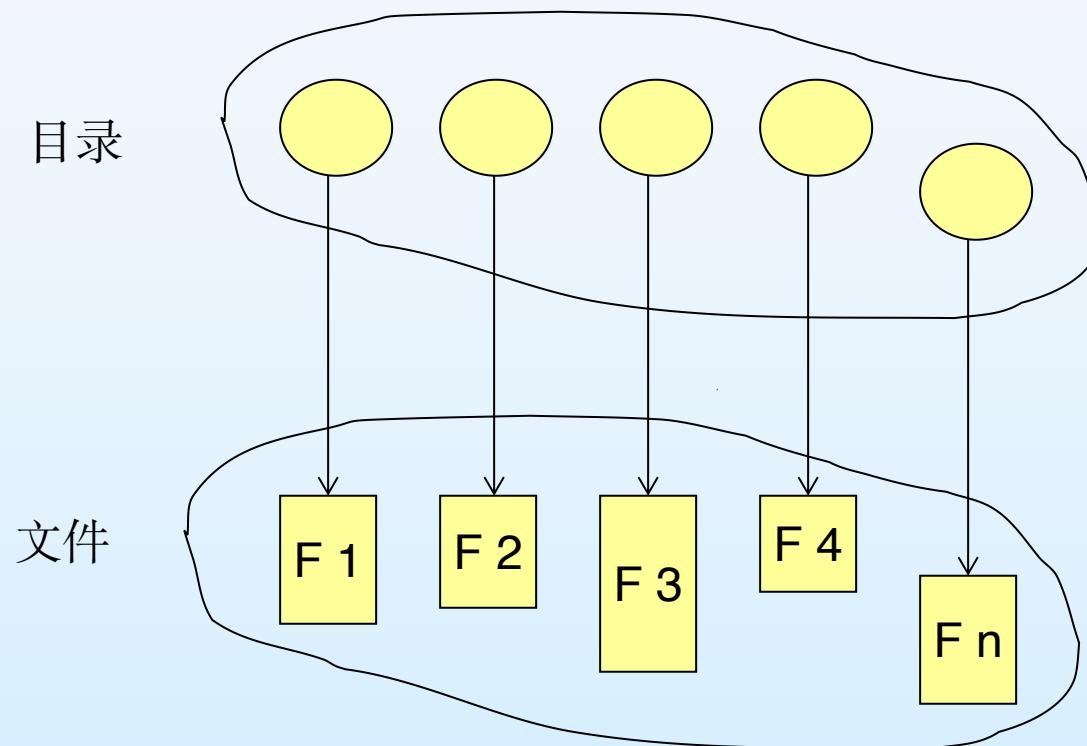
目录文件





# 目录结构

- n 目录和文件都驻留在存储设备（如磁盘）
- n 每个目录项中存放了文件在存储设备的存放地址





# 目录

- 是用户和文件之间的一个桥梁
- 用户根据文件名访问文件，而文件是存储在存储设备上的
- 负责把文件名转换为文件在存储设备上的位置，这个根据文件名找文件存储位置信息的过程，就是文件检索。





# 文件检索过程

n 文件检索是一个遍历目录项的过程

1. 根据文件所在目录打开目录文件;
2. 从磁盘读入该目录文件的1个(物理)块，该块包含若干个目录项；
3. 根据文件名遍历内存中的该块，如找到对应文件名的目录项则结束；
4. 判断该目录文件是否还有物理块没有读入，如果是转2；否则，结束。表示该目录中没有此文件名的文件。

n 目录项由于经常变化，一般不排序

n 平均遍历目录项数： **(1+n) /2**

不包括文件查不到的情况

目录项0
目录项1
目录项2
....
目录项I
目录项i+1
....
目录项n-1





# 目录项的检索性能

- n **(物理) 块 (Block)** : 内存和存储设备数据交换基本单位。一个物理块一般为4KB、8KB和16KB等。每次读写文件都是以块为单位，至少1块。
  - | 例如，当前物理块大小是4KB，则一个读入文件的1个字节的操作也要从存储设备读入4KB
- n 目录性能：为了提高文件的检索效率，需要读入尽可能少的物理块（耗时少）





# 目录项的检索性能

n 如果

- | 目录项大小 =  $ds$  bytes
- | 目录中最多文件数 =  $n$
- | 物理块大小 =  $b$

n 那么

- | 目录文件大小 =  $ds * n$  bytes
- | 目录文件需要的物理块数目 =  $ds * n / b$
- | 检索一个文件需要平均读入的块数 =  $(ds * n / b + 1) / 2$

n 因此 (由于物理块大小固定, 想要降低读入物理块数有两个途径)

- | 降低  $ds$ , 降低目录项大小 → 降低物理块数
- | 降低  $n$ , 降低文件数 → 降低物理块数





n 一般而言，一个目录中文件越多则文件访问的性能越差。

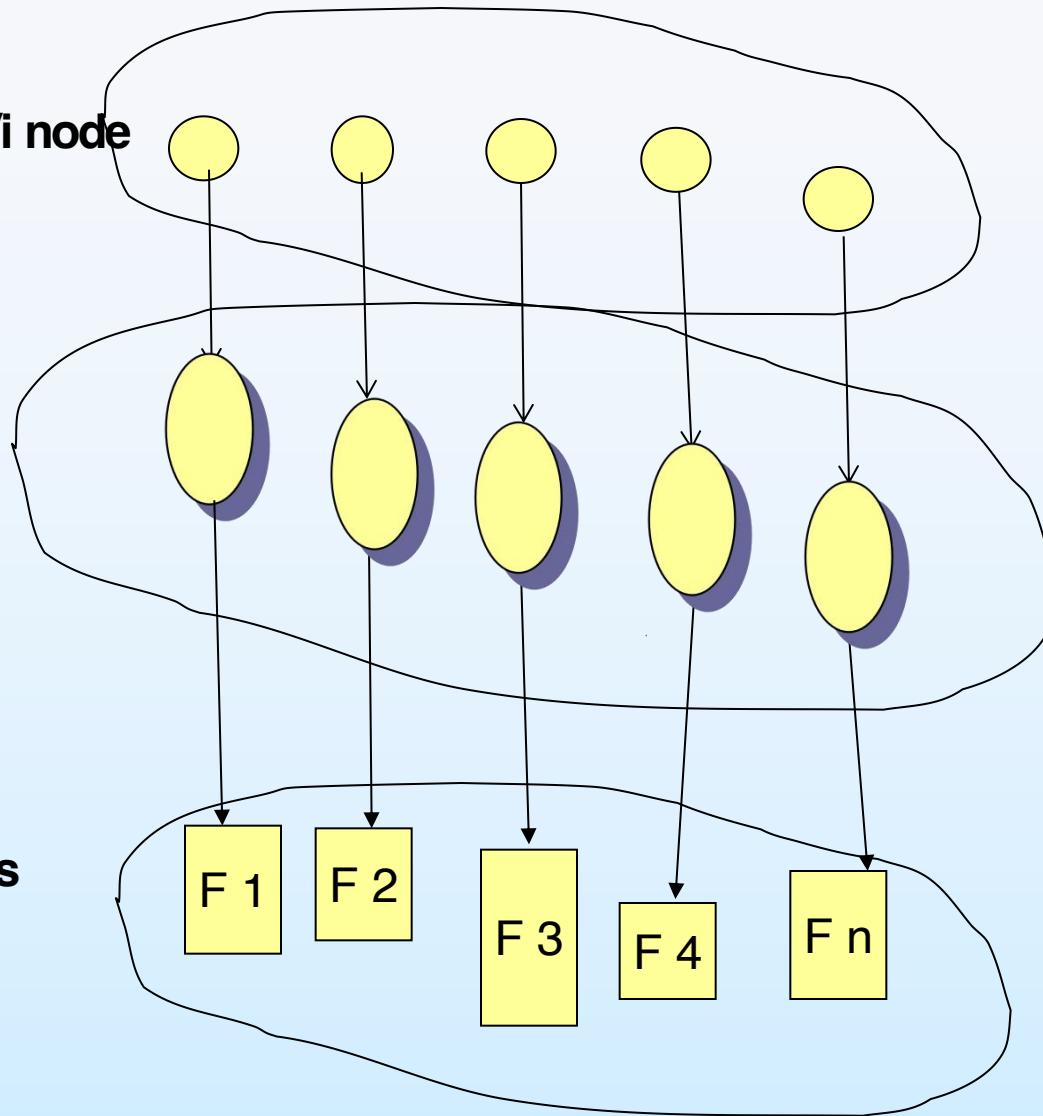


# i Node

Directory item/i node

FCB

Files





- n 一种降低目录项大小的方法就是Linux采用的iNode
- n 一个完整的目录项就是一个文件控制块，里面包含了大量的文件属性，但是这些属性绝大多数在文件检索的时候是用不到的
- n 如果不读入这些无关的信息，降低目录项的大小，可以提高文件访问的效率
  
- n 文件检索过程中需要用到的主要内容是文件名，为此Unix为每个文件控制块建立了一个索引项，内容为文件名和指向文件控制块的指针，类似于索引文件，称为inode索引节点
- n 目录文件只包含一个inode索引节点，每个索引节点就是一个目录项，在索引节点中有一个指针，指向该文件的文件控制块，文件控制块再指向对应的文件，这样遍历目录项就是遍历索引节点
- n 由于索引节点比传统的目录项小了很多，就可以提高检索效率





# 例子

n 物理块大小为4KB，某个目录中有1万个文件，每个文件的FCB（目录项）大小为2KB，则

- | 目录文件大小: 20000KB
- | 目录文件需要的物理块数量:  $20000\text{KB}/4\text{KB}=5000$ 块
- | 检索文件平均需要访问的物理块数:  $(5000+1)/2=\mathbf{2500.5}$

n i node: 64B

- | 目录文件大小:  $640000\text{B}=625\text{KB}$
- | 目录文件需要的物理块数量: 157块
- | 检索文件平均需要访问的物理块数:  $(157+1)/2=\mathbf{79}$

1/32





# 目录相关操作

- n 打开目录
- n 创建目录
- n 删除目录
- n 遍历目录
- n 读目录
- n 切换目录
- n .....
- n 搜索文件
- n 创建文件
- n 删除文件
- n 列出目录
- n 重命名文件
- n 跟踪文件系统
- n 这些操作和文件本身无关

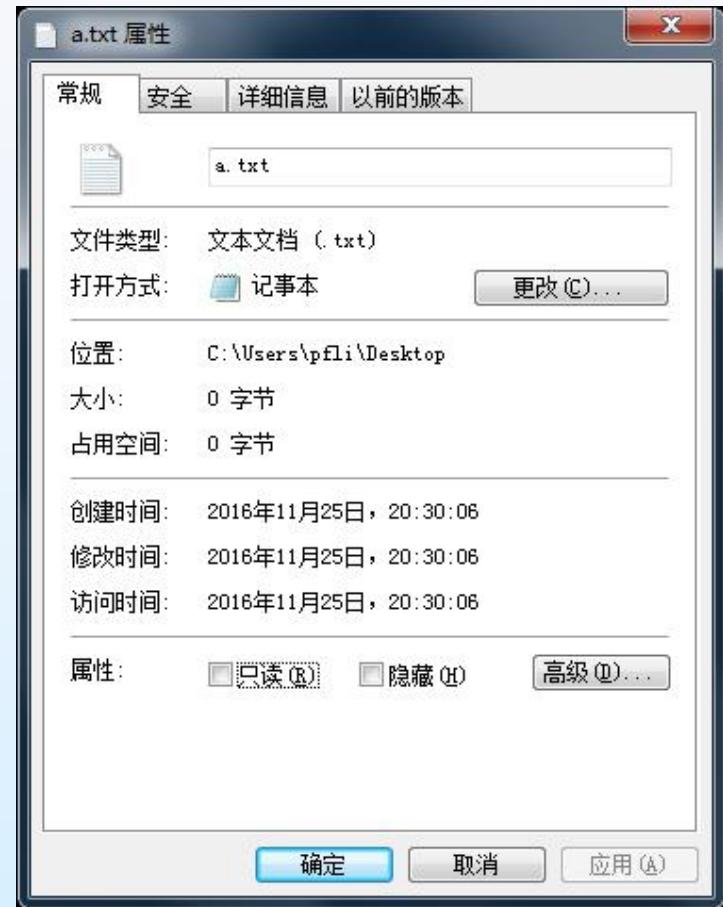




# 目录相关操作

## n 创建文件

只是为文件创建了一个目录项，  
只为这个文件分配了一个FCB  
用来存储各类信息





- n 以下文件操作中，需要访问文件内容的是（ ）。
- A. 读文件
  - B. 打开文件
  - C. 文件改名
  - D. 删 除文件





# 文件保护

- n 文件的所有者/创建者应该有权控制：
  - | 能做什么
  - | 由谁来做
- n 文件存取类型
  - | 读
  - | 写
  - | 执行
  - | 添加
  - | 删除
  - | 列表清单





# 访问控制列表和分组

- n 访问模式：读/写/执行，用R/W/X来表示
- n 三种类型的用户

		RWX
a) 所有者	7	1 1 1 RWX
b) 同组用户	6	1 1 0 RWX
c) 公共用户	1	0 0 1

- n 建立一个组，加入一些用户
- n 对特定的文件或目录(*game*)，定义适当的访问权限

owner group public

\ \ \ | / /

chmod 761 game





# UNIX 访问控制

-rw-rw-r--	1	pbg	staff	31200	Sep 3 08:30	intro.ps
drwx-----	5	pbg	staff	512	Jul 8 09:33	private/
drwxrwxr-x	2	pbg	staff	512	Jul 8 09:35	doc/
drwxrwx---	2	pbg	student	512	Aug 3 14:13	student-proj/
-rw-r--r--	1	pbg	staff	9423	Feb 24 2003	program.c
-rwxr-xr-x	1	pbg	staff	20471	Feb 24 2003	program
drwx--x--x	4	pbg	faculty	512	Jul 31 10:31	lib/
drwx-----	3	pbg	staff	1024	Aug 29 06:52	mail/
drwxrwxrwx	3	pbg	staff	512	Jul 8 09:35	test/





# Windows 7 访问控制列表管理



- 
- n Unix中执行chmod 321 mydoc, 会把（ ）权限授权给同组用户。
- A. 运行
  - B. 读
  - C. 写
  - D. 所有



10.53



## 4、目录结构





# 内容

- n 单层目录
- n 双层目录
- n 树形目录
- n 无环图目录
- n 通用图目录



# 目录结构的设计目标

## n 效率

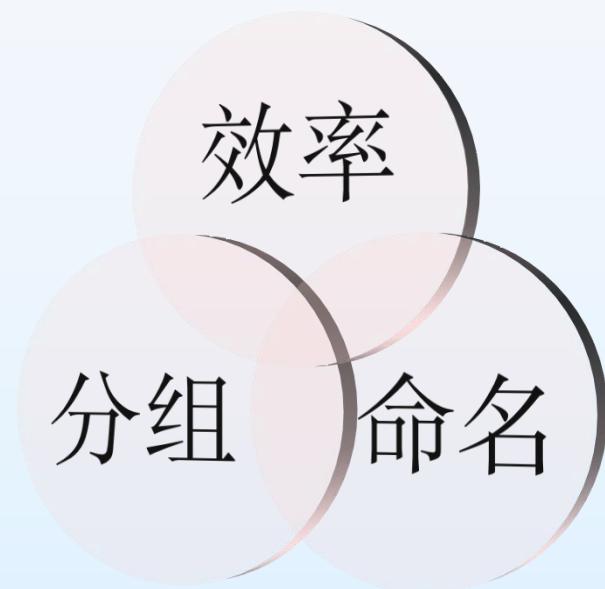
- | 快速定位文件位置
- | 提高文件访问效率

## n 命名

- | 方便用户使用
- | 同名的不同文件
- | 不同名的相同文件

## n 分组

- | 文件分组 (子目录)
- | 兼顾效率和方便性





n 目录设计的目标中不包括 () 。

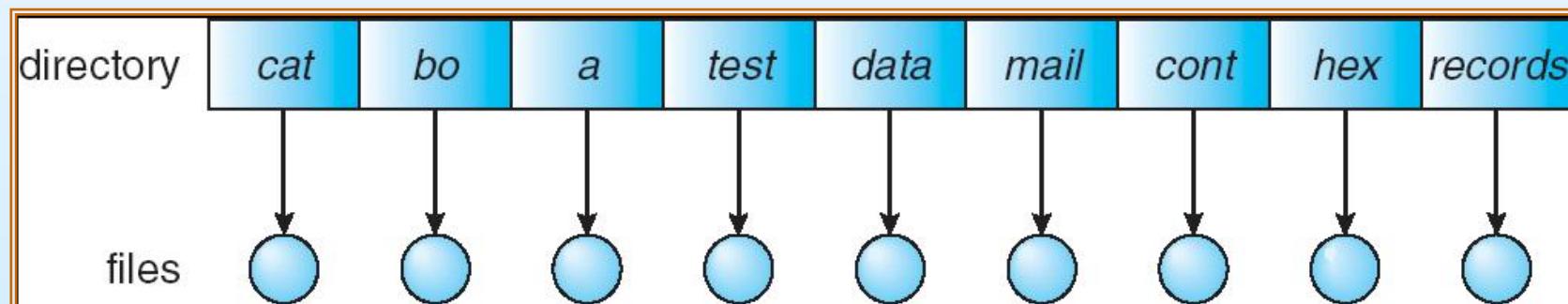
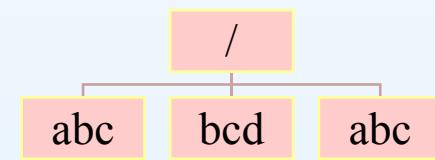
- | A.文件分组
- | B.同名文件
- | C.访问效率
- | D.节省空间





# 单层目录

- n 所有文件在同一目录中，只有一级目录：根目录
- n 根目录 (/)：一个文件系统最顶层的目录
- n 优点：结构简单
- n 缺点
  - | 检索效率差（目录下文件过多）
  - | 不能有同名文件，一个文件只能有一个名称
  - | 不能分组



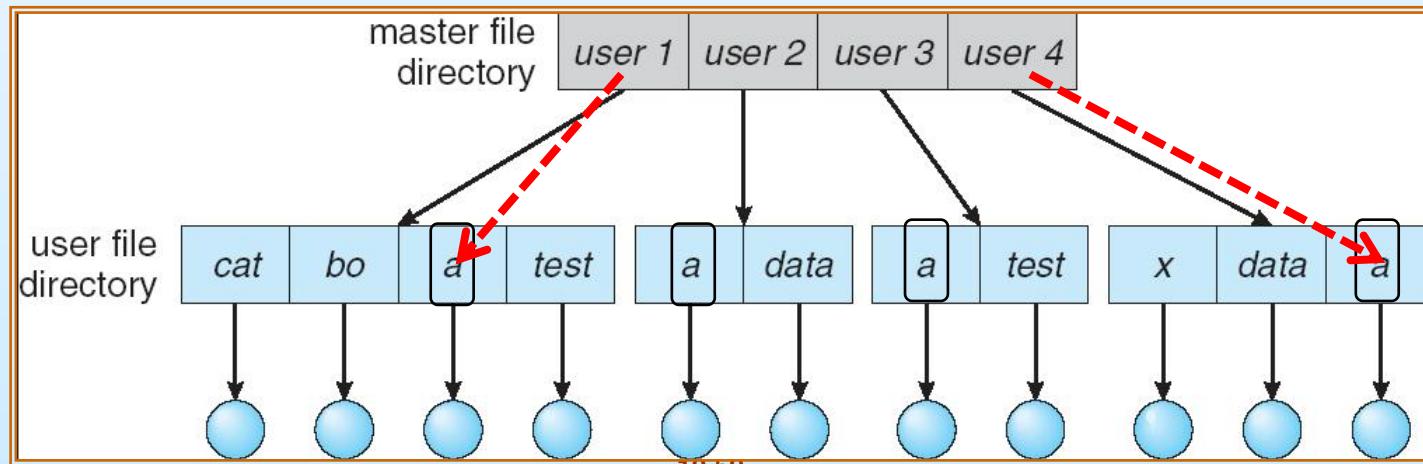


# 双层目录

- n 每个用户有自己的目录结构
- n 目录下的目录
- n 优点: 1) 不同用户可有相同文件名的文件; 2) 比单层目录提高检索效率 (文件分布在多个用户目录中)
- n 缺点: 1) 同一用户无法分组; 2) 同一用户不能有相同文件名的文件

## n 路径名

- | 不同目录存在同名文件, 为了区分引入了路径
- | 从根目录开始到文件的路径, 称为路径名
- | 路径名=目录名+分隔符+文件名
  - ▶ /user4/a
  - ▶ /user1/a





# 树型目录

## n 特点

- | 检索高效 (子目录增多导致每个目录下文件减少)
- | 可以分组, 用户可以自由建立子目录
- | 允许重名, 不同子目录可以有同名文件

## n 当前目录: 工作目录

- | **cd** /spell/mail/prog

## n 绝对路径

- | 从根开始的路径名

## n 相对路径

- | 从当前目录开始的路径名
- | 提高检索效率

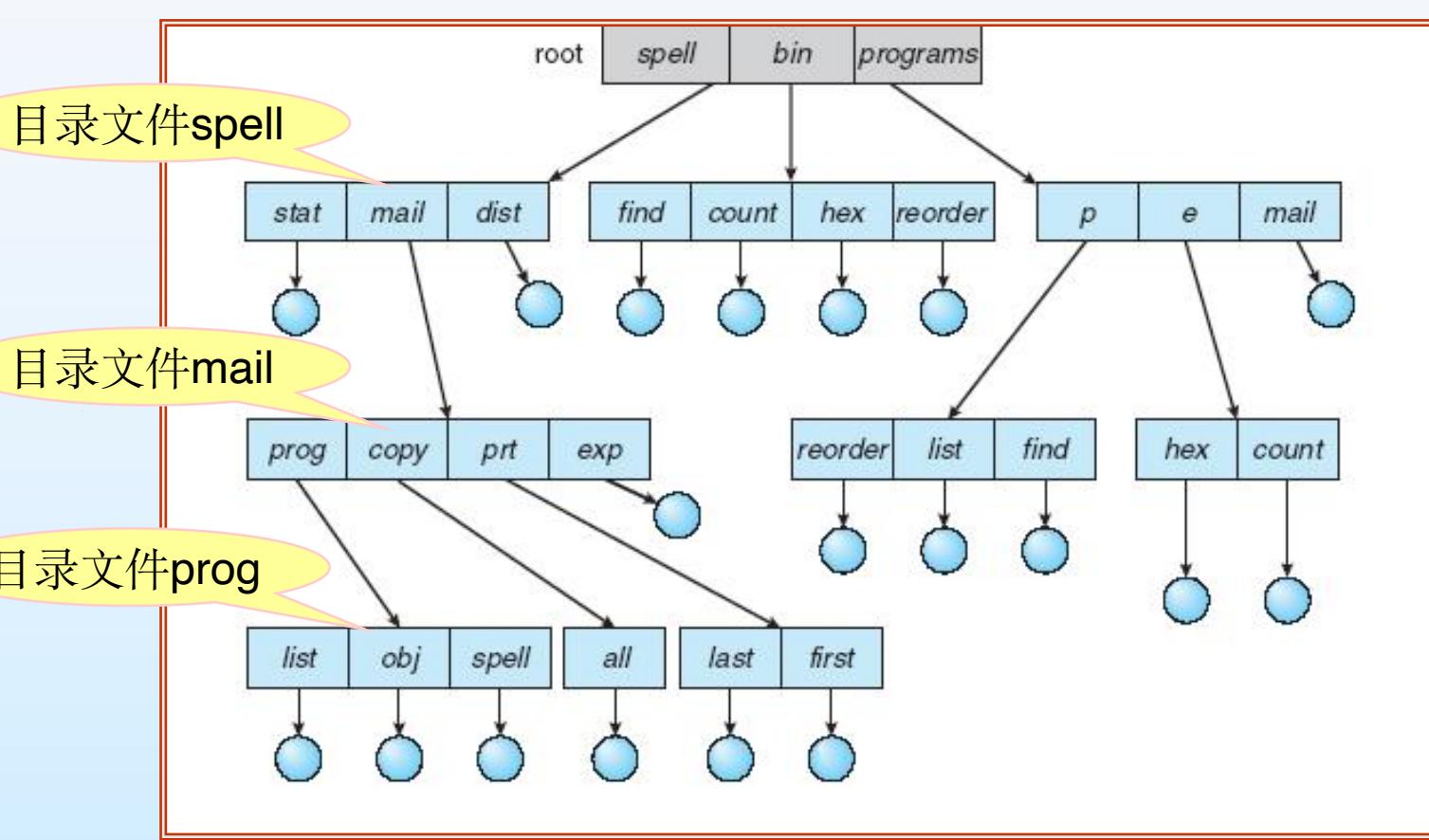




# 树型目录

n 双层目录的扩展: 2层->N层

cd /spell/mail/proc





# 性能对比 (查找文件first为例)

n 目录项：占1个物理块

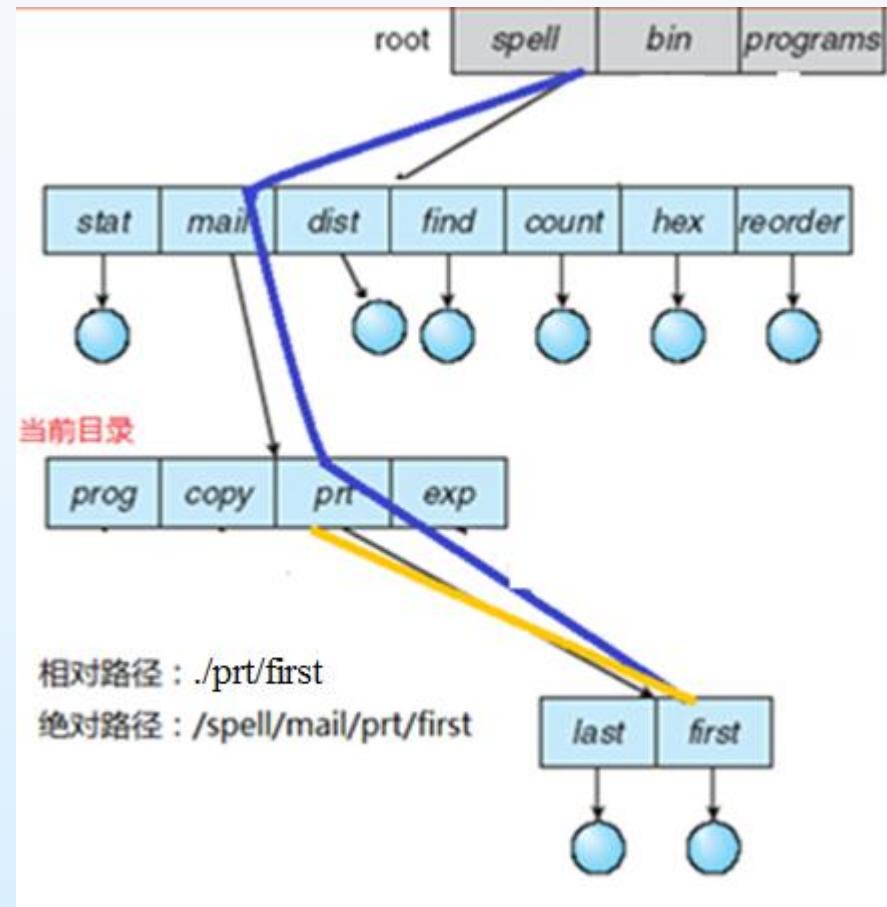
目录文件	目录项	平均读入块数
/	spell	0
spell	mail	$(1+7)/2=4$
mail	prt	$(1+4)/2=2.5$
prt	first	$(1+2)/2=1.5$

n 绝对路径

- | 从根目录开始到文件
- | 根目录一般在内存
- | 读入 $4+2.5+1.5=8$ 个物理块

n 相对路径

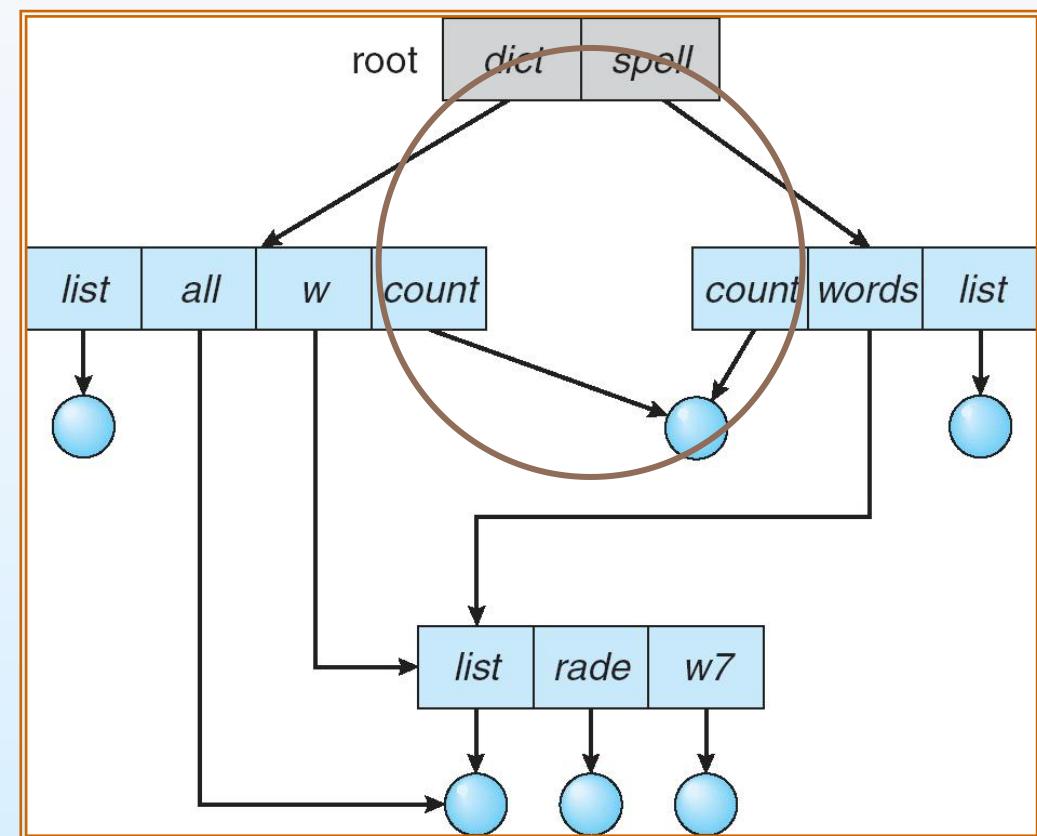
- | 当前目录在内存
- | 读入1.5个物理块





# (有向) 无环图目录

- n 文件共享：不同目录中的文件指向同一个物理文件，也就是它们内容相同
- n 树型目录不能实现文件共享
- n 解决方法：图型目录
  - | 无环图目录
  - | 通用图目录（有环图）
- n 无环图：有向边无环



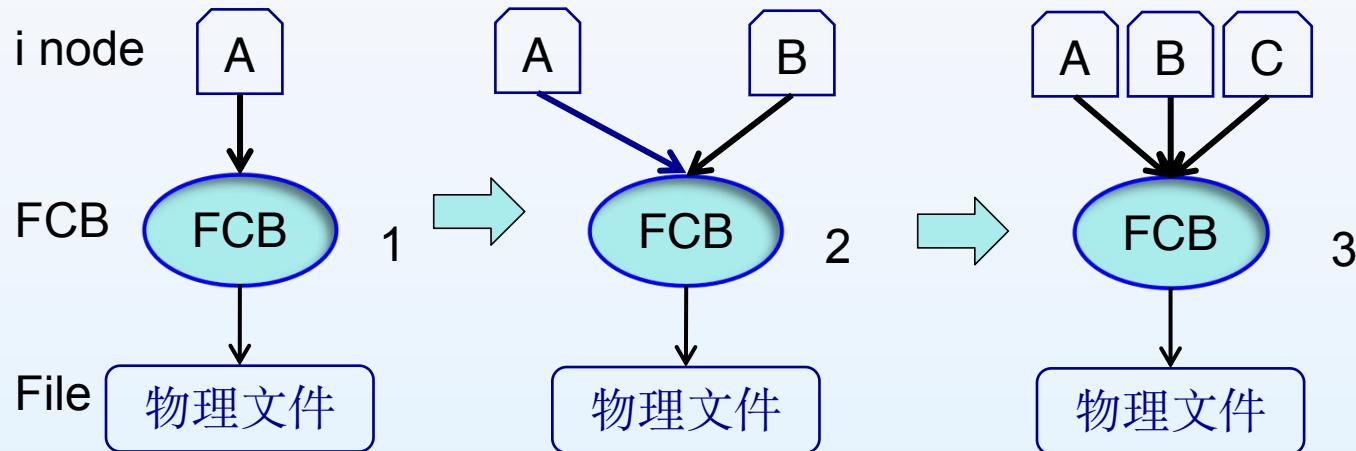


# 文件共享

- n 两种共享文件方式：
  - | 符号链接（软链接）
  - | 硬链接（Hard Link）：一个物理文件可以有几个文件名，但是只有一个FCB，每个文件名有自己的目录项，也就是i node，但共享一个FCB
    - ▶ UNIX



# 硬链接



[Linux]\$ touch f1

[Linux]\$ ln f1 f2

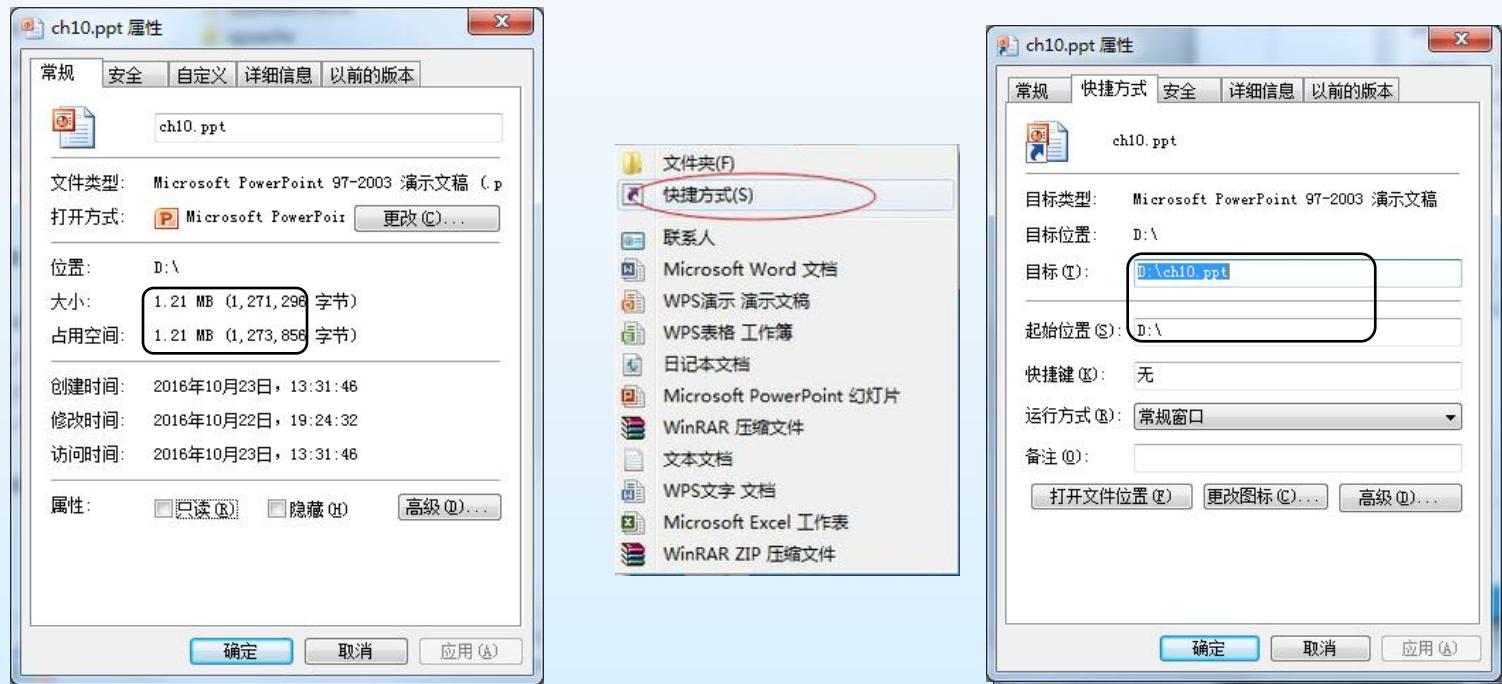
#创建一个测试文件f1

#创建f1的一个硬连接文件f2



# 符号链接例子 (Windows)

## | 符号链接 (Symbolic Link) ▶ Windows/Linux





```
n [Linux]$ touch f1          #创建一个测试文件f1  
[Linux]$ ln f1 f2           #创建f1的一个硬连接文件f2  
[Linux]$ ln -s f1 f3         #创建f1的一个符号连接文件f3  
[Linux]$ ls -li              # -i参数显示文件的inode节点信息  
total 0  
9797648 -rw-r--r--    2 oracle oinstall 0 Apr 21 08:11 f1  
9797648 -rw-r--r--    2 oracle oinstall 0 Apr 21 08:11 f2  
9797649 lrwxrwxrwx  1 oracle oinstall 2 Apr 21 08:11 f3 -> f1
```





# Linux的硬链接和软链接

## n 硬链接(hard link):

- | A是B的硬链接 (A和B都是文件名) , 则A的目录项中的inode节点号与B的目录项中的inode节点号相同, 即一个inode节点对应**两个不同的文件名, 两个文件名指向同一个文件**。如果删除了其中一个, 对另外一个没有影响。每增加一个文件名, inode节点上的链接数增加一, 每删除一个对应的文件名, inode节点上的链接数减一, 直到为0, inode节点和对应的数据块被回收。注: 文件和文件名是不同的东西, rm A删除的只是A这个文件名, 而A对应的数据块(文件) 只有在inode节点链接数减少为0的时候才会被系统回收。

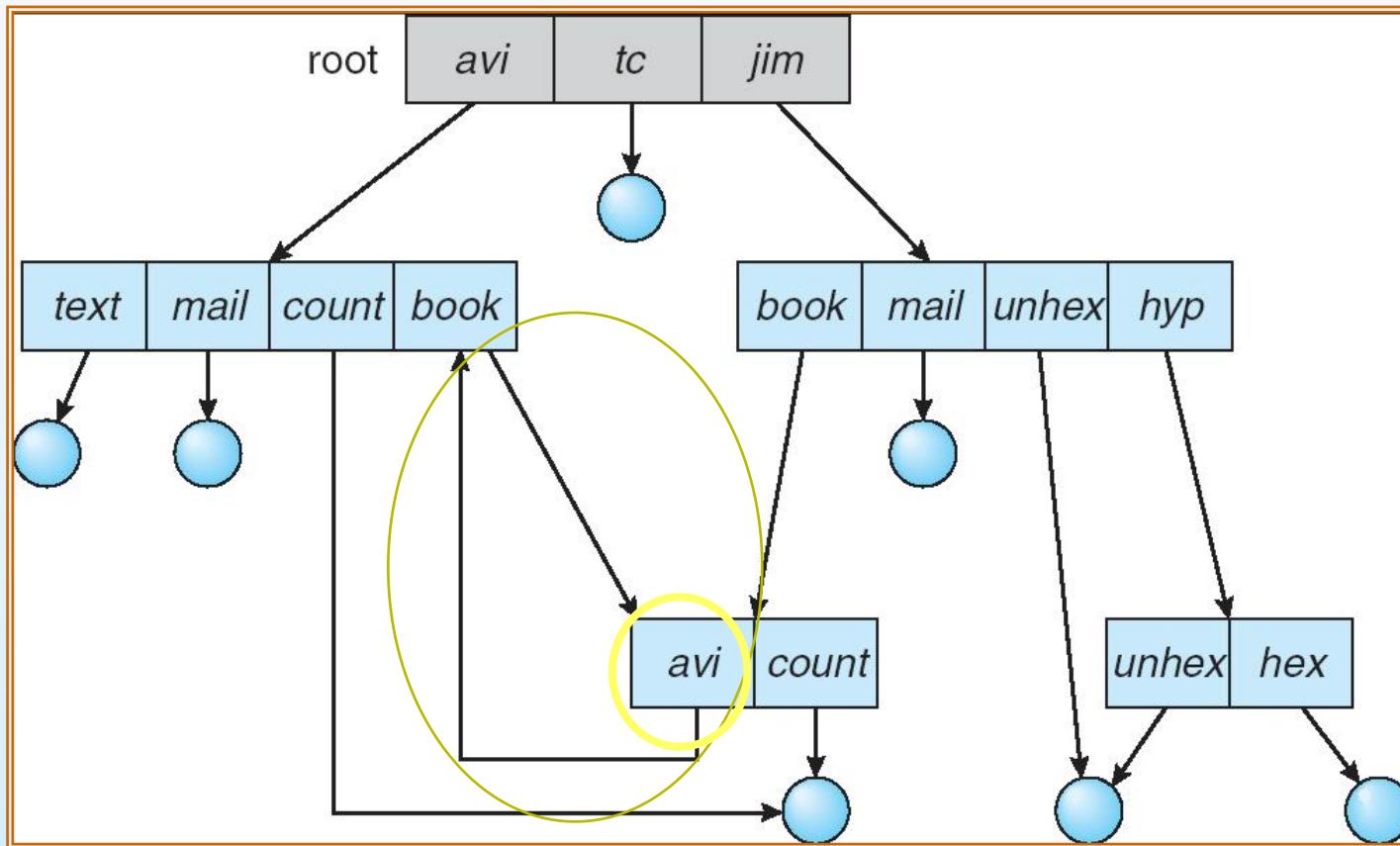
## n 软链接(soft link):

- | A是B的软链接 (A和B都是文件名) , A的目录项中的inode节点号与B的目录项中的inode节点号不相同, A和B指向的是两个不同的inode, 继而指向两块不同的数据块。但是A的数据块中存放的只是B的路径名(可以根据这个找到B的目录项)。A和B之间是“主从”关系, 如果B被删除了, A仍然存在(因为两个是不同的文件), 但指向的是一个无效的链接。



# 通用图目录

n 图中有环





# 通用图目录

## n 如何保证无环?

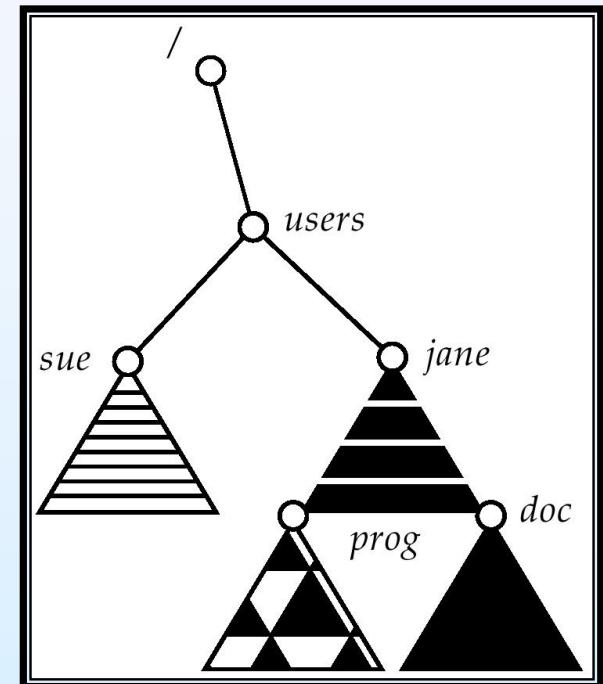
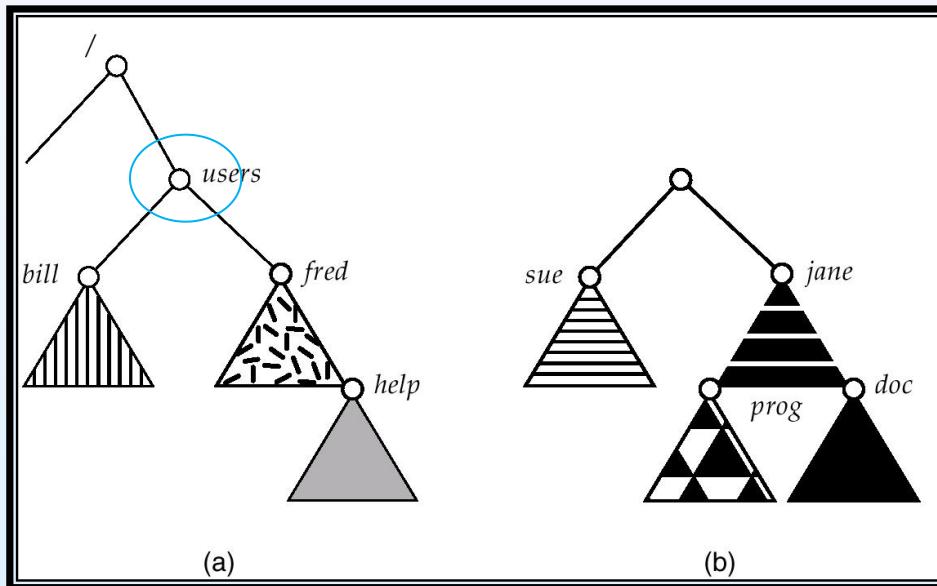
- | 仅允许指向文件的链接，不允许指向子目录的链接
- | 垃圾回收，发现环就消除
- | 每当加入新链接时，使用环路检测算法判断是否正确
- | 优化遍历目录算法，避免对环的重复搜索





# 文件系统安装

- 要访问一个文件系统，必须先安装它
- 一个未安装的文件系统将被安装在一个安装点(**mount point**)上





10.72





10.73