

操作系统 *Operation System*

文件系统

王雷

wanglei@buaa.edu.cn

TEL: 82316284

内容提要

- 文件系统基本概念
- 文件系统实现方法
- 文件系统实例分析

- 硬件资源：

中央处理机、主存储器及各种输入输出设备。

- 软件资源：

各种系统程序、标准子程序和某些常用的应用程序。

数据的持久化存储

所有的应用程序都需要存储和检索数据，但是这些数据应该保存在什么地方呢？

- 当一个进程正在运行的时候，可以将一些数据保存在它自己的地址空间中。
- 但是，用户地址空间的大小是有限的；当进程运行结束后，这些数据就丢失了。
- 因此，有些数据必须保存在磁盘、光盘等外部存储设备上，因为这些存储设备能够长期地保存大量的数据。

计算机中的数据应该如何来存放？

- 三个基本要求
 - 能够存储大量的数据（突破地址空间限制）
 - 长期保存：进程终止以后，数据依然存在
 - 可以共享数据：有些数据可能会被多个不同的进程所访问；
- 为解决这些问题，人们提出“文件”的概念，把数据组织成文件的形式，用文件作为数据的存储和访问单位。

文件是什么？

- 所谓文件是指一组带标识（标识即为文件名）的、在逻辑上有完整意义的信息项的序列。
- 信息项：构成文件内容的基本单位（单个字节，或多个字节），各信息项之间具有顺序关系。
- 文件内容的意义：由文件建立者和使用者解释。
- 文件包括两部分：
 - 文件体：文件本身的内容；
 - 文件说明：文件存储和管理的相关信息，如：文件名、文件内部标识、文件存储地址、访问权限、访问时间等；

文件是什么？

- 文件是一种抽象机制，它提供了一种把信息保存在磁盘等存储设备上，并且便于以后访问的方法。抽象性体现在用户不必关心具体的实现细节。
- 可以视为一个单独的连续的逻辑地址空间，其大小即为文件的大小，与进程的地址空间无关。

文件是什么？

- 文件的本质是一组字节序列，源于用户程序对所输入处理的原始数据和输出结果的“长期”（持久化）保存的需求，并按一定“格式”（规范）呈现。所有“需要长期保存”的文件都按某种组织形式存放（映射）到磁盘中。
 - 所有的IO设备（字符设备，块设备以及网络设备）都可以看作为字节序列的载体，因此，所有的IO设备也可以抽象为文件进行表达。这就是现代OS中“一切皆文件”的含义。
 - 因此所有的I/O设备，包括磁盘、键盘、鼠标、显示器都可以看成是文件。
 - 所有的I/O设备都是文件这一抽象概念的具体表现。

文件管理

- 早期解决方法
 - 将程序和数据保存在纸带和卡片上，再用纸带机或卡片机输入到计算机。
- 磁盘存储器和磁带存储器出现，程序和数据才开始真正被计算机**自动管理**。
- 解决方法：把信息以一种单元，即文件的形式存储在磁盘或其他外部介质上。
- **文件是通过操作系统来管理的**，包括：
 - 文件的结构、命名、存取、使用、保护和实现方法

文件管理的需求

■ 用户视角（使用逻辑文件）

- 用户关心文件中要使用的数据，不关心具体的存放形式（和位置）。
- 关心的是文件系统所提供的对外的用户接口，包括文件如何命名、如何保护、如何访问（创建、打开、关闭、读、写等）；

■ 操作系统视角（组织和管理物理文件）

- 文件的描述和分类，关心的是如何来实现与文件有关的各个功能模块，包括如何来管理存储空间、文件系统的布局、文件的存储位置、磁盘实际运作方式（与设备管理的接口）等。

文件系统

- 文件系统是操作系统中统一管理信息资源的一种软件，管理文件的存储、检索、更新，提供安全可靠的共享和保护手段，并且方便用户使用。
- 文件系统要完成的任务：
 1. 统一管理磁盘空间，实施磁盘空间的分配与回收；
 2. 实现文件的按名存取：名字空间--映射-->磁盘空间；
 3. 实现文件信息的共享，并提供文件的保护、保密手段；
 4. 向用户提供一个方便使用、易于维护的接口，并向用户提供有关统计信息；
 5. 提高文件系统的性能；
 6. 提供与IO系统的统一接口。

文件名

- 当一个文件被创建时，必须给它指定一个名字，用户通过文件名来访问文件。
 - 命名规则：文件名是一个有限长度的字符串。一个文件名一般由两部分组成，中间用句点隔开。
- 文件名.扩展名
 - 文件名的长度一般不超过8个字符，但很多系统支持长文件名，如255个字符。文件名可以由字母、数字和特殊字符组成，有的系统区分字母的大小写（如Unix），有的系统则不区分（如Windows）。

文件类型

- 程序所处理的数据的种类繁多：如数值数据、字符数据、二进制数据，以及以此为基础的具有各种组织形式的数据。为方便使用，人们定义了各种类型的文件。
- 按性质和用途：系统文件、库文件、用户文件
- 按数据形式：源文件、目标文件、可执行文件
- 按对文件实施的保护级别分：只读文件、读写文件、执行文件、不保护文件
- 按逻辑结构分：有结构文件、无结构文件
- 按文件中物理结构分：顺序文件、链接文件、索引文件

例：UNIX的文件类型

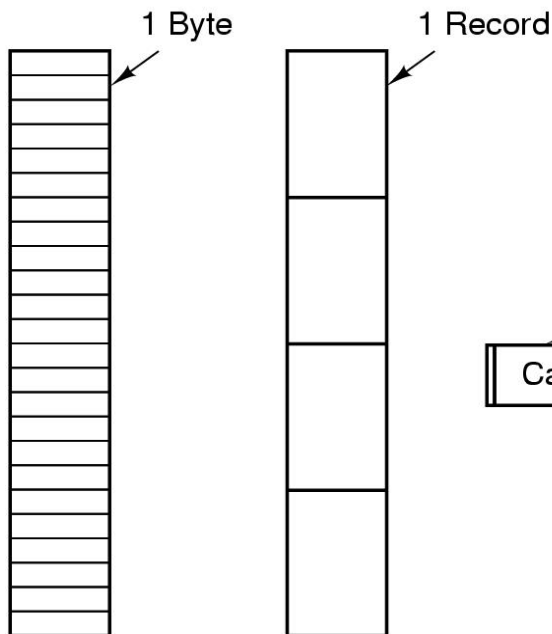
UNIX按性质和用途将文件分为：一般分为普通文件、目录文件、特殊文件（设备文件）、管道文件、套接字。

- **普通文件**：即用户自己建立的文件，包含了用户的信息，一般为ASCII或二进制文件；
- **目录文件**：管理文件系统的系统文件；
- **特殊文件**：
 - 字符设备文件：和输入输出有关，用户模仿串行IO设备，例如终端、打印机、网卡等。
 - 块设备文件：磁盘、磁带等。

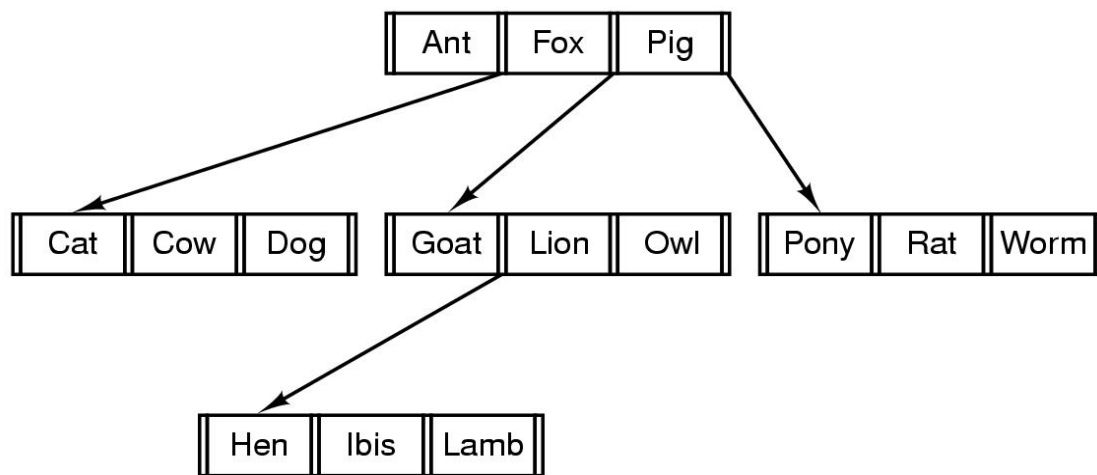
file type	usual extension	function
executable	exe, com, bin or none	read to run machine- language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual 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 com- pressed, for archiving or storage
multimedia	mpeg, mov, rm	binary file containing audio or A/V information

文件的逻辑结构

这是从用户角度看文件，由用户的访问方式确定的。这里给出了三种逻辑结构，还可以组织成堆、顺序、索引、散列等结构。第一种是以字节为单位的流式结构，第二种是一种记录式文件结构，最后一种是树形结构。



(a) 字节序列 (b) 记录序列



(c) 树

典型的文件结构、文件存取方式及存储介质

- 流式文件：构成文件的基本单位是字符。文件是有逻辑意义、无结构的一串字符的集合；
- 记录式文件：文件由若干记录组成，可以按记录进行读写、查找等操作。每条记录有其内部结构
- 文件存取方式：
 - 顺序存取（访问）；
 - 随机存取：提供读写位置（当前位置）。
- 文件的存储介质
 - 典型的存储介质：磁盘(包括SSD)、磁带、光盘等；
 - 物理块（块block、簇cluster）：数据存储、传输和分配的单位，存储设备通常划分为大小相等的物理块，统一编号。

文件基本操作

- 创建文件
 - 删除文件
 - 打开文件
 - 关闭文件
 - 读文件
 - 写文件
 - 修改文件名
 - 设置文件的读写位置
1. Create
 2. Delete
 3. Open
 4. Close
 5. Read
 6. Write
 7. Append
 8. Seek
 9. Get attributes
 10. Set Attributes
 11. Rename

“目录”的提出

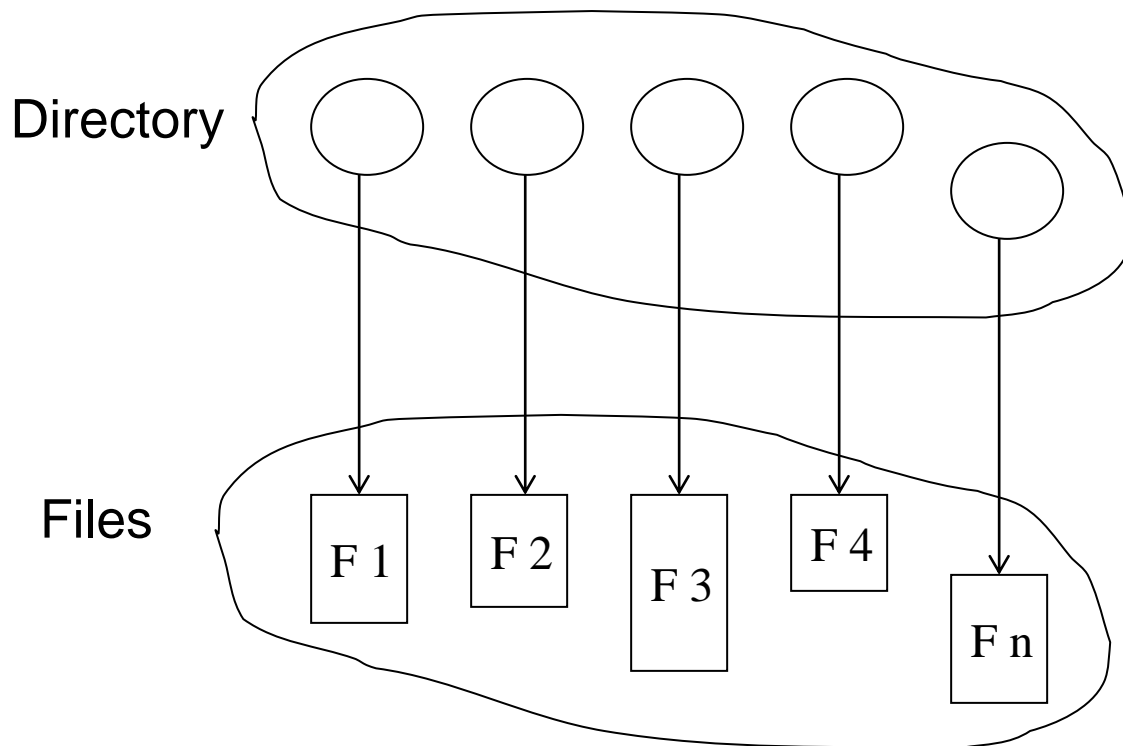
- 文件太多了怎么办？不同的应用程序有不同类型的文件，不同的用户有不同的文件，如何对它们进行组织、分类？
- 如何对文件进行管理？当用户需要访问某个文件时，如何根据这个文件名迅速地定位到相应的文件，从而对文件的属性和内容进行各种操作？

“目录”的提出

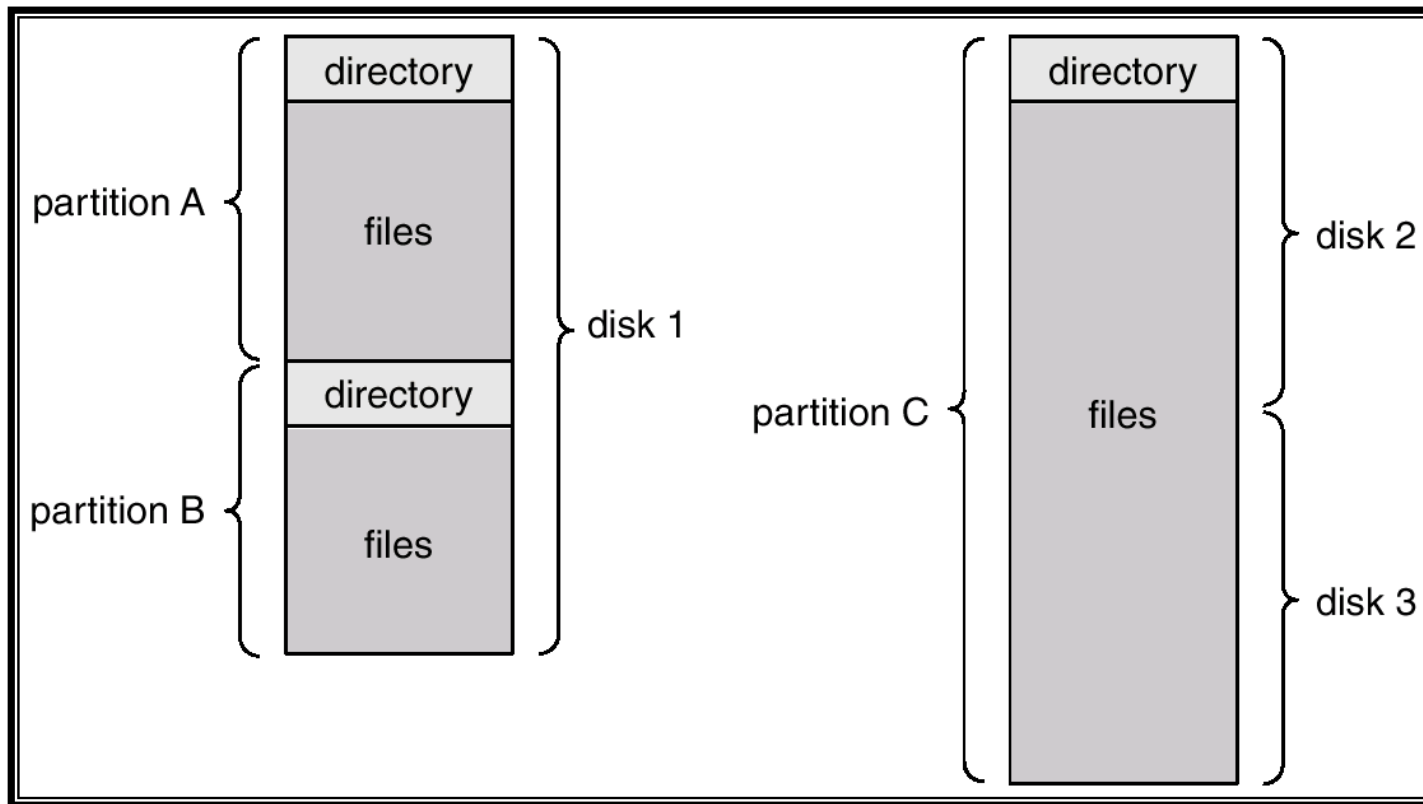
- **效率**：要能迅速定位一个文件；
- **命名**：要方便用户；
 - 重名：对不同的用户文件，可使用相同名字；
 - 别名：对同一个文件，可有多多个不同的名字；
- **分组功能**：能够把不同的文件按照某种属性进行分组（如某门课程的所有文件可分为讲义、作业、参考资料等不同的组）。

目录管理

- 目录是由文件说明索引组成的用于文件检索的特殊文件。文件目录的内容主要是文件访问和控制的信息（不包括文件内容）。



典型的文件系统的组织



目录内容

- 目录的内容是文件属性信息(properties)，其中的一部分是用户可获取的。
- 基本信息
 - 文件名：字符串，通常在不同系统中允许不同的最大长度，可以修改。有些系统允许同一个文件有多个别名(alias)；
 - 别名的数目；
- 文件类型：可有多种不同的划分方法，如：
 - 有/无结构（记录文件，流式文件）；内容（二进制，文本）用途（源代码，目标代码，可执行文件，数据）属性attribute（如系统，隐含等）文件组织（如顺序，索引等）

目录内容（续）

■ 地址信息

- 存放位置：哪个设备或文件卷volume，及各存储块位置；
- 文件长度（当前和上限）：以字节、字或存储块为单位。

■ 访问控制信息

- 文件所有者（属主）：通常是创建文件的用户，也可改变；
- 访问权限（用户访问方式）：读、写、执行等。

■ 使用信息

- 创建时间；
- 最后一次读访问的时间和用户。
- 最后一次写访问的时间和用户。

目录操作

1. Create
2. Delete
3. Opendir
4. Closedir

1. Readdir
2. Rename
3. Link
4. Unlink

文件目录分类

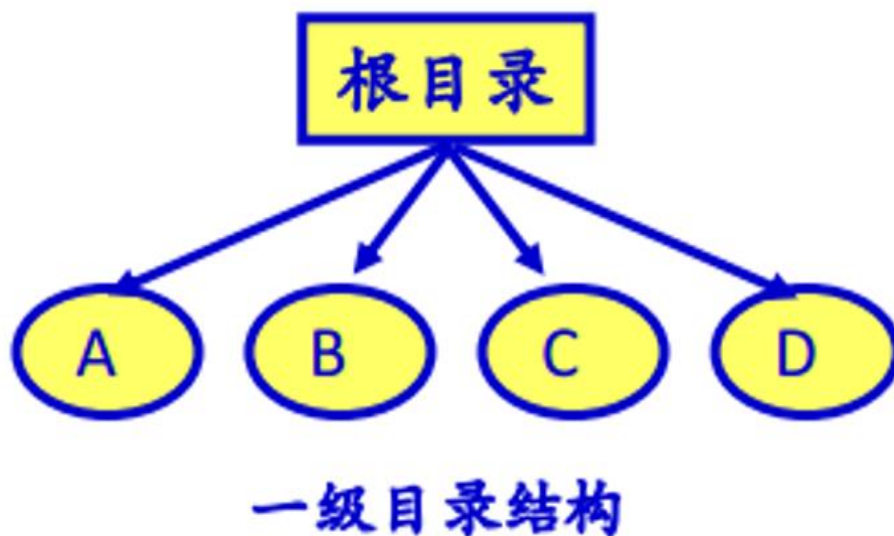
■ 目录分类

- 单级文件目录
- 二级文件目录
- 多级文件目录

单级文件目录

- 文件目录的每个表目应包含：

1. 文件的符号名
2. 文件所在物理地址
3. 文件结构信息
4. 存取控制信息
5. 管理信息

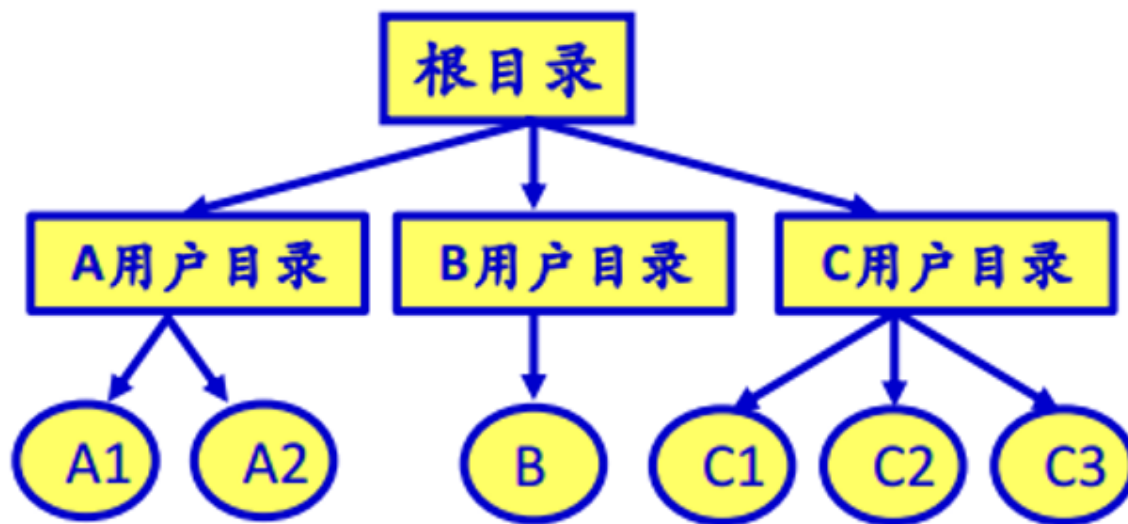


特点

- 结构简单（只在早期的个人计算机上使用过）；
- 文件多时，目录检索时间长；
- 有命名冲突：如多个文件有相同的文件名（不同用户的相同作用的文件）或一个文件有多个不同的文件名（不同用户对同一文件的命名）；
- 不便于实现共享。

两级目录

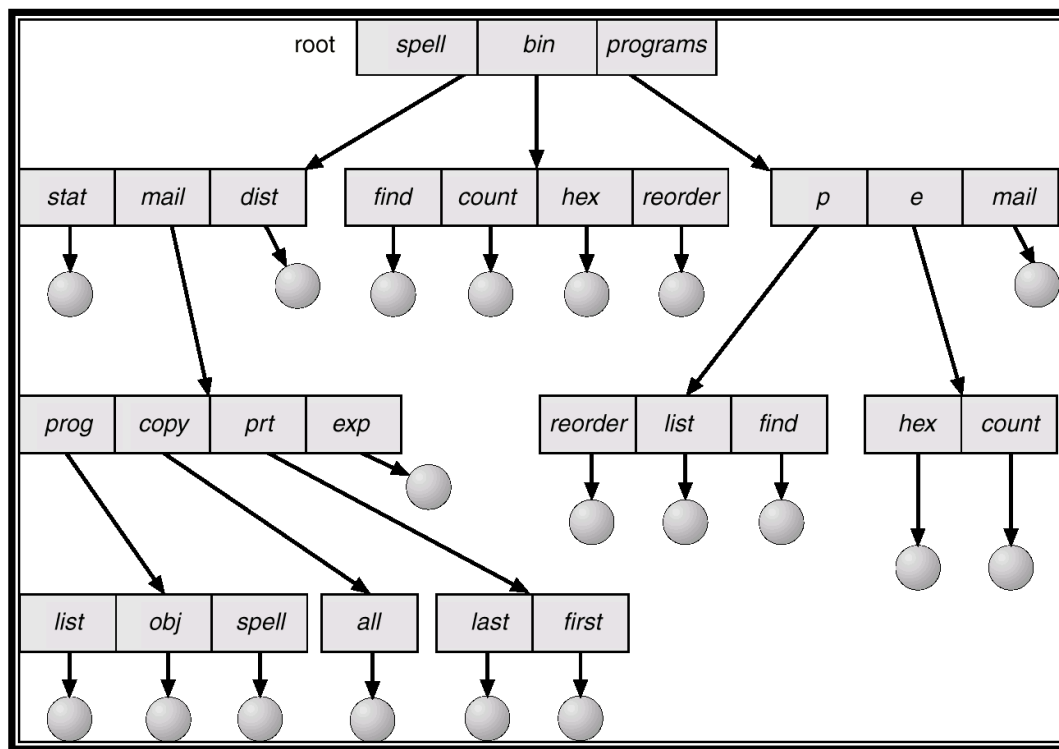
- 在根目录（第一级目录）下，每个用户对应一个目录（第二级目录），在用户目录下是该用户的文件，而不再有下级目录。适用于多用户系统，各用户可有自己的专用目录。



二级目录结构

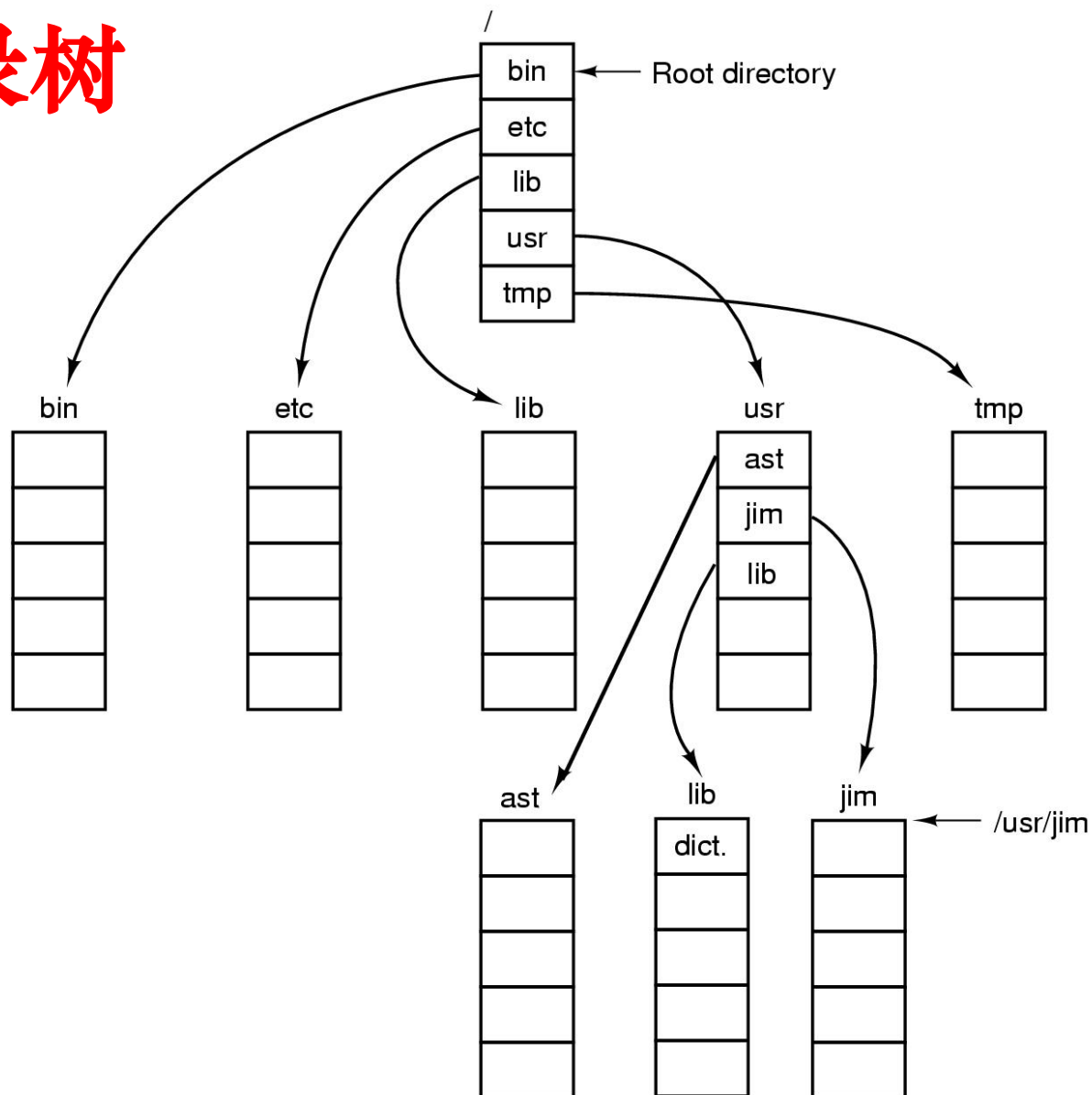
多级目录（层次目录）

- 在较高的目录级，其目录表目为下一级目录名以及一个指向其目录的指针。
- 在最后一级目录，这个指针指向文件的物理地址。
- 几乎所有现代文件系统都采用这种方案。



- 在多级目录结构中，如何来指定需要访问的文件？
 - **绝对路径名**：对于每一个文件或目录，可以用从根目录开始依次经由的各级目录名，再加上最终的文件名或目录名来表示（之间用分隔符隔开）**一个文件或目录的绝对路径名是唯一的**。例如：\spell\mail\copy\all
 - **相对路径**：结合当前路径进行使用，比如上个例子中，如果当前目录为 /usr 那么 ast/mailbox 指的就是根目录下的 usr 文件夹中的 ast 文件中的 mailbox 文件
 - **当前目录**：也叫工作目录，用户可以指定一个目录作为当前的工作目录；
 - **上一级目录**：使用 “..” 表示上一级目录，读作 dotdot, 在根目录下 “..” 表示它本身，还是根目录。

Unix 目录树



多级目录特点

■ 层次清楚

- 不同性质、不同用户的文件可以构成不同子树，便于管理；
- 不同用户的文件可以被赋予不同的存取权限，有利于文件的保护；
- 数目较多时，便于系统和用户将文件分散管理，使得文件和目录的层次结构较为清晰。适用于较大的文件系统管理；

■ 可解决文件重名问题

- 文件在系统中的搜索路径时从根开始到文件名为止的各文件名组成，只要在同一目录下的文件名不发生重复就不会由文件重名而引发混乱；

■ 查找速度快

- 可为每类文件建立一个子目录，由于对多级目录的查找每次只查找目录的一个子集，所以搜索速度快于一级和二级目录；

■ 目录级别太多时，会增加路径检索时间。

文件系统的定义和目的

定义：

- 操作系统中与文件管理有关的那部分软件和被管理的文件以及实施管理所需要的数据结构的总体。

目的：

- 为系统管理者和用户提供了对文件的透明存取（**按名存取**）：不必了解文件存放的物理机制和查找方法，只需给定一个代表某段程序或数据的文件名称，文件系统就会自动地完成对给定文件名称相对应的文件的有关操作

文件系统的任务目标

- 方便的文件访问：以符号名称作为文件标识，便于用户使用；
- 并发文件访问和控制：在多道程序系统中支持对文件的并发访问和控制；
- 统一的用户接口：为不同设备提供统一接口，方便用户操作和编程；
- 多种文件访问权限：在多用户系统中的不同用户对同一文件会有不同的访问权限；
- 执行效率：存储效率、检索性能、读写性能；
- 差错恢复：能够验证文件的正确性，并具有一定的差错恢复能力；

文件系统必须解决的几个主要问题

- 如何有效的分配文件存储器的存储空间
- 提供合适的存取方法
- 命名的冲突和文件的共享
- 文件系统的执行效率
 - 文件系统在操作系统接口中占的比例最大,用户使用操作系统的感觉在很大程度上取决于对文件系统的使用效果.
- 提供与I/O的统一接口

理想文件系统应具有的特性

- 有效的分配文件存储器的存储空间；
- 文件结构和存取的灵活性和多样性；
- 具有对用户来说尽可能透明的机制；
- 尽可能达到对文件存储装置的独立性；
- 存储在文件中的信息的安全性；
- 能方便的共享公用的文件；
- 有效的实现各种文件操作的命令。

文件系统模型



文件系统模型的三个层次

一、文件系统的接口(最接近用户的)

- 为了方便用户使用操作系统，文件系统通过向用户提供两种类型的接口
- 命令行接口：这是指的是作为用户何文件系统的交互的接口。用户可以通过键盘终端键入命令，取得文件系统的服务。
- 程序接口：这是作为用户程序何文件系统的接口。用户程序可以通过系统调用的形式来取得文件系统的服务。

文件系统模型的三个层次

二、对象操作管理的软件集合

- 这是文件管理系统的核心部分。文件系统的功能大多数是在这一层实现的，其中包括：
 - 对文件存储空间的管理，
 - 对文件目录的管理、
 - 用户将文件的逻辑地址转换伟物理地址的机制、
 - 对文件读和写的管理、
 - 以及对文件的共享和保护的功能。

文件系统模型的三个层次

三、对象及其属性

■ 文件系统管理的对象有

(1) 文件，它作为文件管理的直接对象。

(2) 目录。为了方便用户对文件的存取何检索，在文件系统中必须配置目录，每个目录项中，必须包含文件名及文件所在的物理地址（或指针）。对目录的组织和管理是方便用户何提高对文件存取速度的关键。

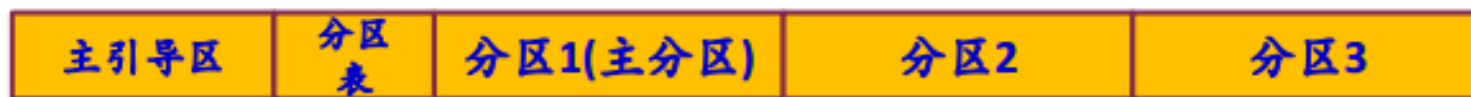
(3) 磁盘存储空间。文件和目录必定占用存储空间，对这部分空间的高效管理，不仅能够提高外存的利用率，而且能够提高对文件的检索速度。

内容提要

- 文件系统基本概念
- 文件系统实现方法
- 文件系统实例分析

磁盘上文件系统的布局

整块磁盘



UNIX文件系统布局



文件卷(逻辑分区)



Windows的FAT文件系统布局

文件的实现需解决以下两个问题

- 如何来描述一个文件，如何来记录文件的各种管理信息？
- 如何来存放文件，即如何把文件的各个连续的逻辑块存放在磁盘上的空闲物理块当中？如何来记录逻辑块与物理块之间的映射关系？

文件控制块与文件属性

文件控制块(FCB, File Control Block):

- 为管理文件而设置的数据结构，保存管理文件所需的所有有关信息（文件属性或元数据）

文件常用属性:

- 文件名，文件号，文件大小，文件地址，创建时间，最后修改时间，最后访问时间，保护，口令，创建者，当前拥有者，文件类型，共享计数，各种标志（只读、隐藏、系统、归档、ASCII/二进制、顺序/随机访问、临时文件、锁）

文件控制块(FCB, File Control Block)

■ 基本信息

- **文件名**：字符串，通常在不同系统中允许不同的最大长度。可以修改。
- **物理位置**；
- **文件逻辑结构**：有/无结构（记录文件，流式文件）
- **文件物理结构**：（如顺序，索引等）

■ 访问控制信息

- **文件所有者（属主）**：通常是创建文件的用户，或者改变已有文件的属主；
- **访问权限**（控制各用户可使用的访问方式）：如读、写、执行、删除等；

■ 使用信息

- **创建时间，上一次修改时间**，当前使用信息等。

一个典型的文件说明

文件名	
文件所在的物理地址	
记录长度	记录个数
文件占有者的存取权限	
其它用户的存取权限	
.....	
.....	
文件建立时间	
上次存取时间	
暂存文件/永久文件	

文件逻辑结构和物理结构

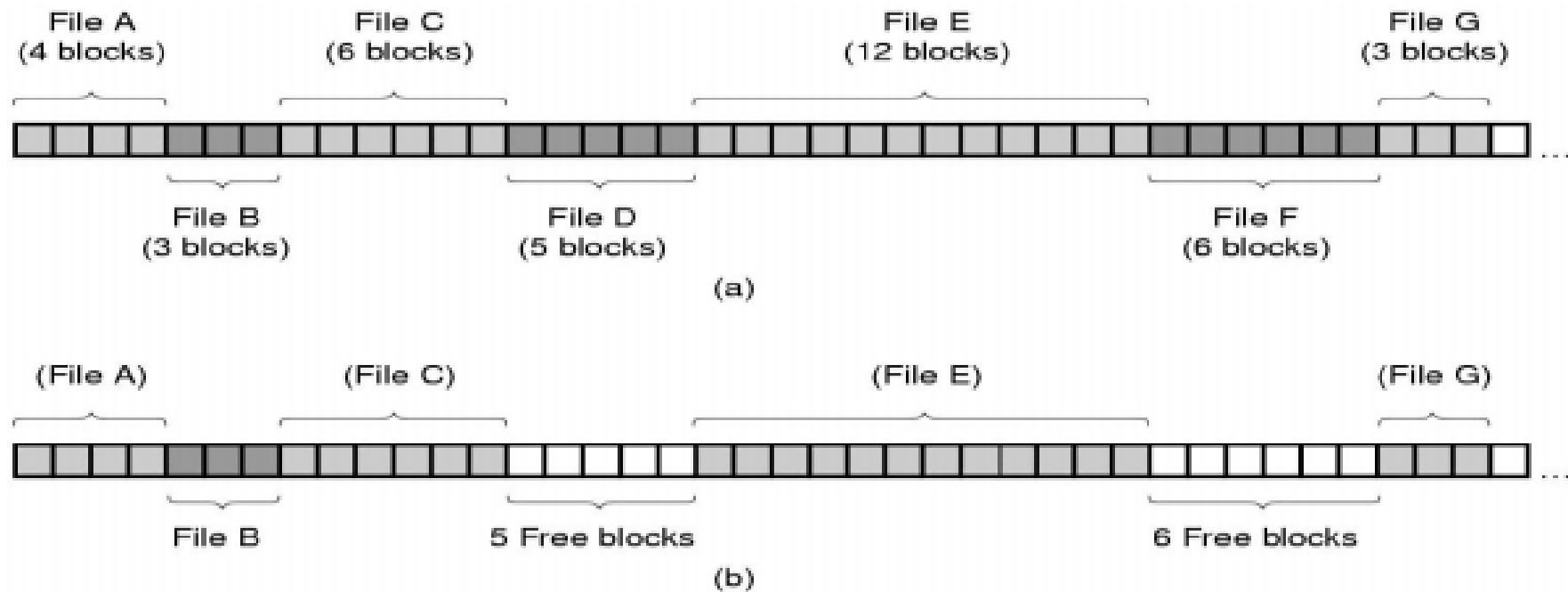
- 文件逻辑结构（文件组织）

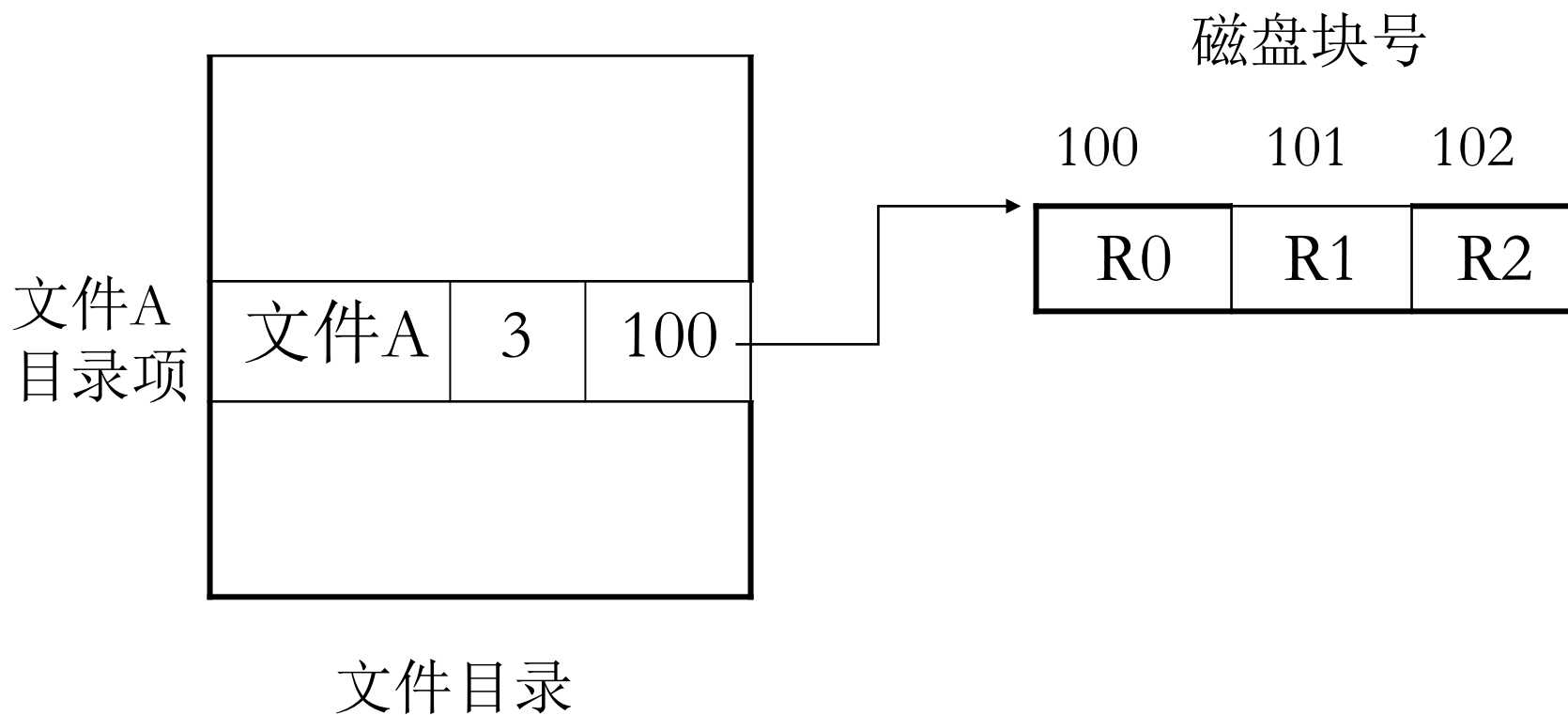
- 提高检索效率；便于修改；降低文件存储费用

- 文件物理结构

- 文件在存储介质上的存放方式，表示了一个文件在文件存储介质上的位置、链接和编目的方法。
 - 主要解决两个问题：
 - 假设一个文件被划分成N块，这N块在磁盘上是怎么存放的？
 - 其地址（块号或簇号）在FCB中是怎样记录的？
 - 主要结构：连续结构、索引结构、串联结构

连续（顺序）结构





在连续文件结构下，当要存取 R_i 记录时，应如何操作？

连续结构的特点

■ 优点：

- 结构简单，实现容易，不需要额外的空间开销
- 支持顺序存取和随机存取，顺序存取速度快
- 连续存取时速度较快，

■ 缺点：

- 文件长度一经固定便不易改变；
- 不利于文件的动态增加和修改；

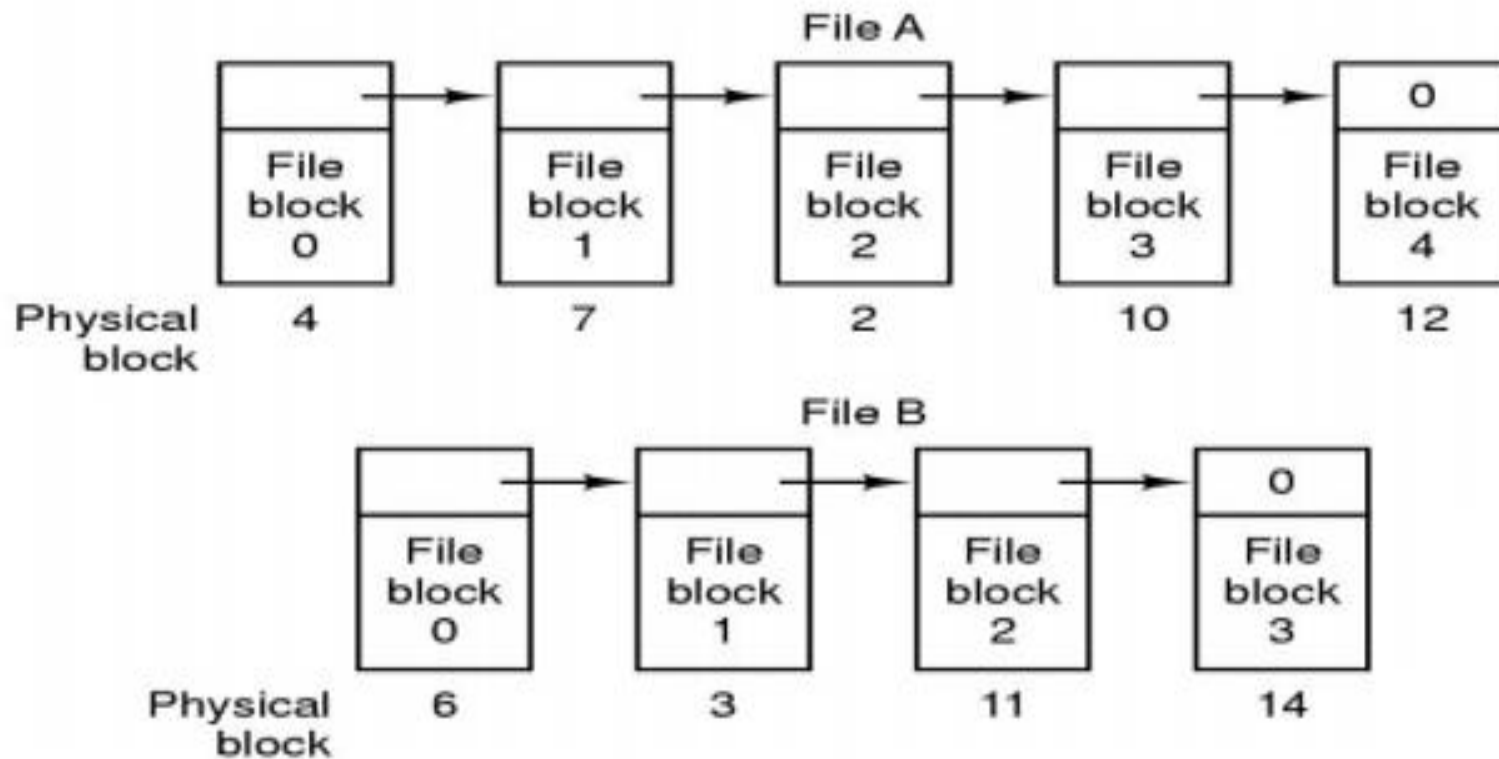
适用于变化不大的顺序访问的文件

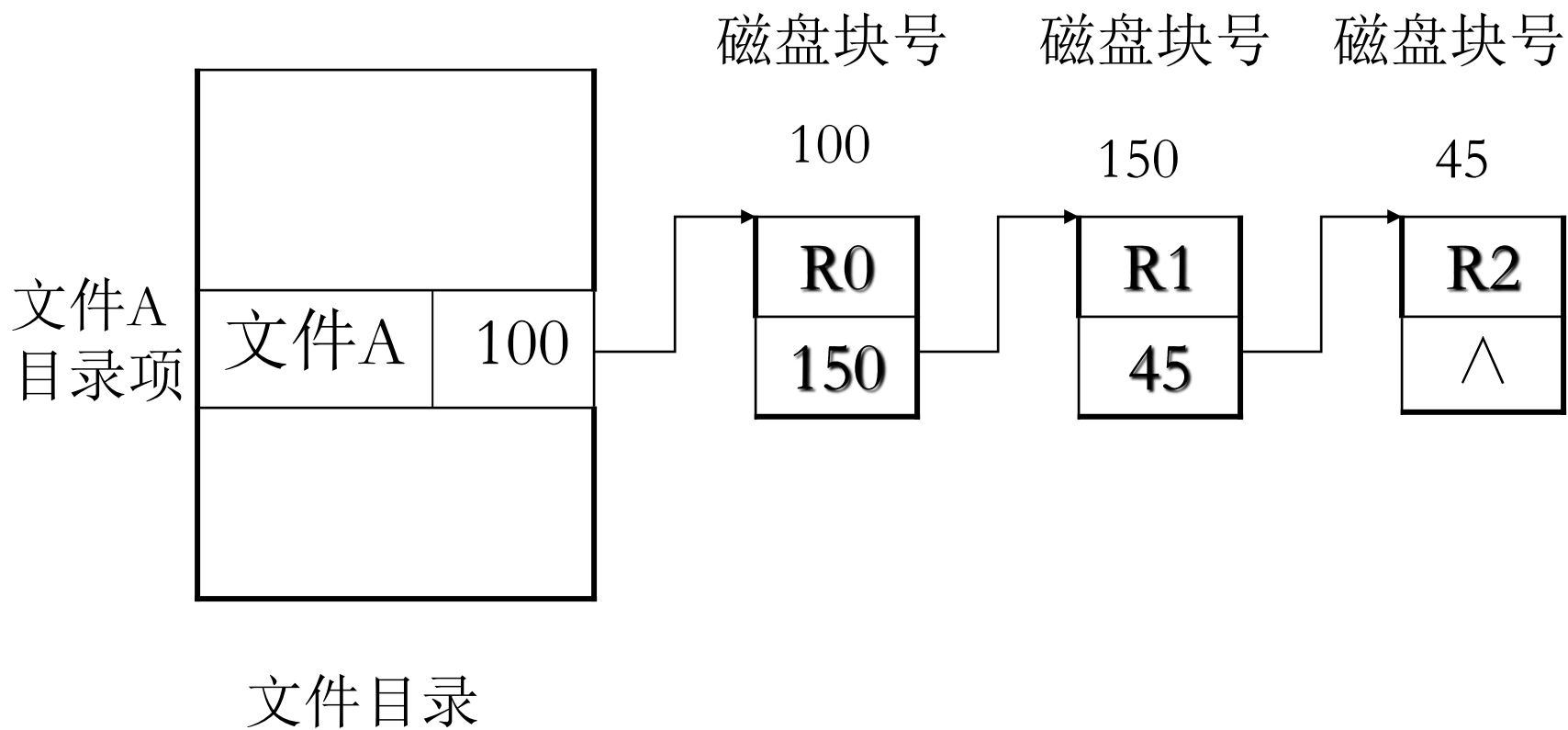
串联/链接文件结构

什么是串联结构

- 串联文件结构是按顺序由串联的块组成的，即文件的信息按存储介质的物理特性存于若干块中。
- 每个物理块的最末一个字(或第一个字)作为链接字，它指出后继块的物理地址。链首指针存放在该文件目录中。文件的结尾块的指针为“^”，表示文件至本块结束。
- 对于记录式文件一块中可包含一个逻辑记录或多个逻辑记录，也可以若干物理块包含一个逻辑记录。

串联文件的结构





在串联文件结构下，当要存取 R_i 记录时，应如何操作？

串联文件的特点

■ 优点：

- 空间利用率高；能较好的利用辅存空间。
- 文件动态扩充和修改容易。
- 顺序存取效率高

■ 缺点：

- 随机存取效率太低，如果访问文件的最后的内容，实际上是要访问整个文件。
- 可靠性问题，如指针出错；
- 链接指针占用一定的空间。

索引结构

■ 索引结构：

- 一个文件的信息存放在若干个不连续物理块中
- 系统为每个文件建立一个专用数据结构：索引表，并将这些物理块的块号存放在该索引中。
- 索引表就是磁盘块地址数组，其中第 i 个目录指向文件的第 i 块。

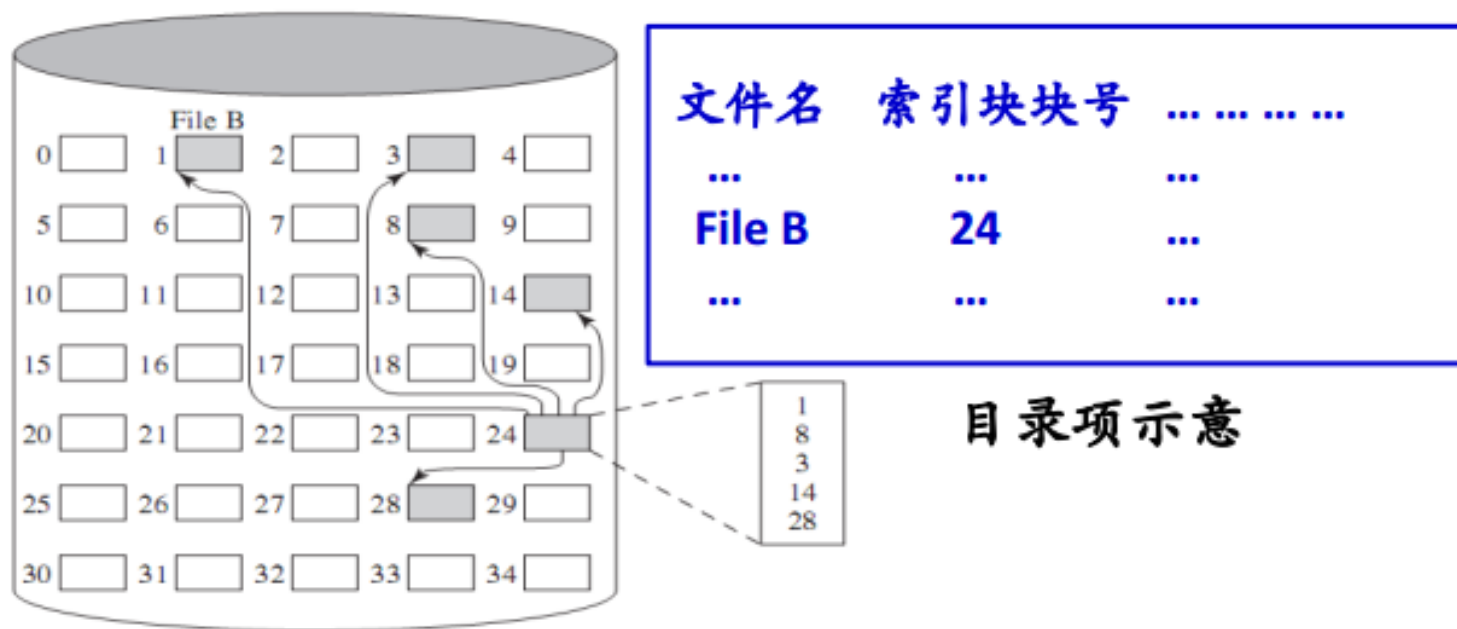
■ 索引表应该存放在何处？

- 这里必须知道每个文件的索引表长度是不一样的，不能存放在FCB中，因此FCB中只记录索引表的地址。

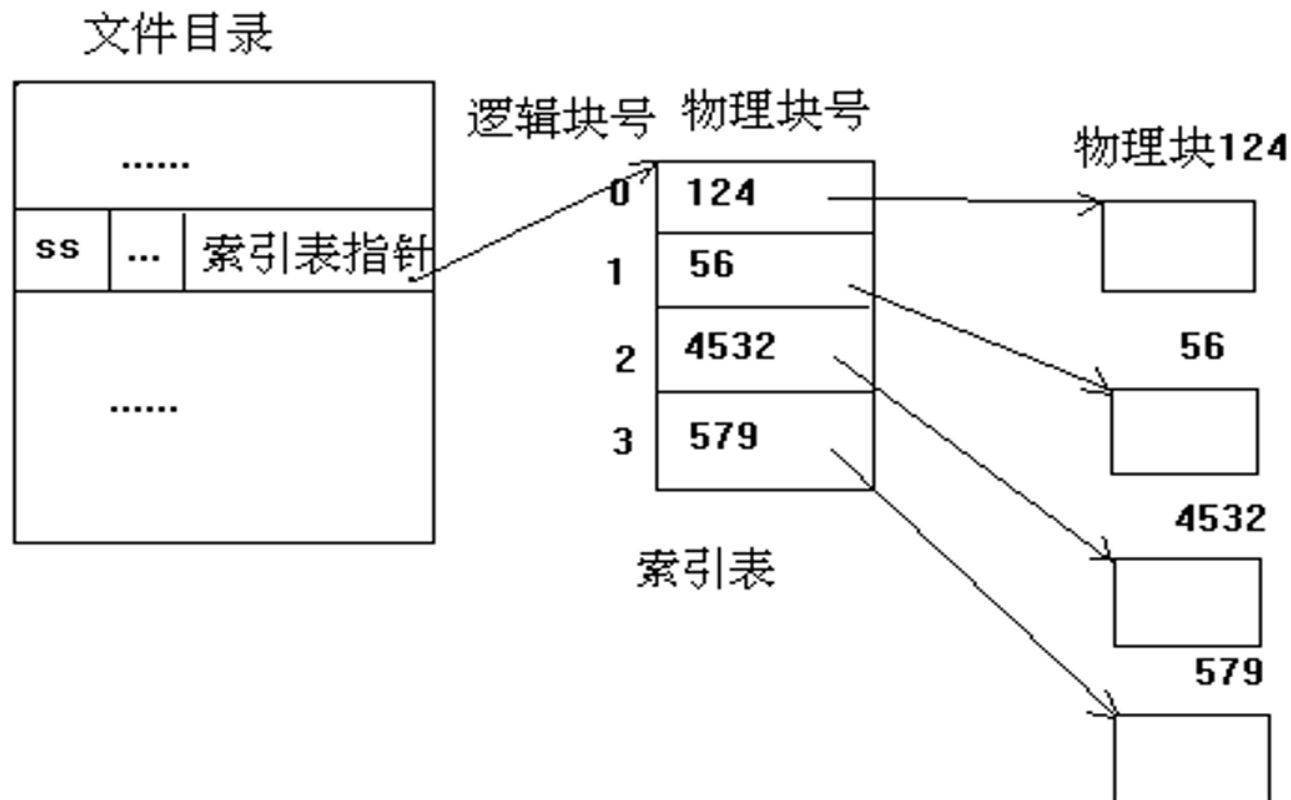
索引文件

系统为每个文件建立逻辑块号与物理块号的对照表，称为文件的索引表。文件由数据文件和索引表构成。这种文件称为索引文件。

- 索引表位置：文件目录中，文件的开头等。
- 索引表大小：固定大小，非固定大小。



索引文件结构



索引文件的操作

- 索引文件在存储区中占两个区：索引区和数据区。索引区存放索引表，数据区存放数据文件本身。
- 访问索引文件需要两步操作：
 - 查文件索引号，由逻辑块号查得物理块号
 - 由此磁盘物理块号而获得所要求的信息

索引文件的特点

■ 优点：

保持了链接结构的优点，又回避了其缺点：

- 即能顺序存取，又能随机存取
- 满足了文件动态增长、插入删除的要求
- 能充分利用外存空间

■ 缺点：

索引表本身带来了系统开销

如：内外存空间，存取时间

索引表的组织

- 链接模式：
 - 一个盘块一个索引表，多个索引表链接起来
- 多级索引：
 - 将一个大文件的所有索引表（二级索引）的地址放在另一个索引表（一级索引）中
- 综合模式：
 - 直接索引方式与间接索引方式结合

	...
	1890
0	124
1	56
2	7981
3	226
4	null
5	null
6	null
7	null
8	null
9	null
10	null
11	null
12	null

文件大小

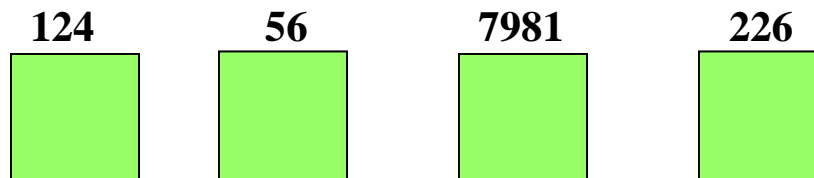
文件目录

File 1	1235
...	

假设：

磁盘块大小：512字节

磁盘块号：4字节



直接索引

File1 i 结点 1235

北京航空航天大学

计算机学院

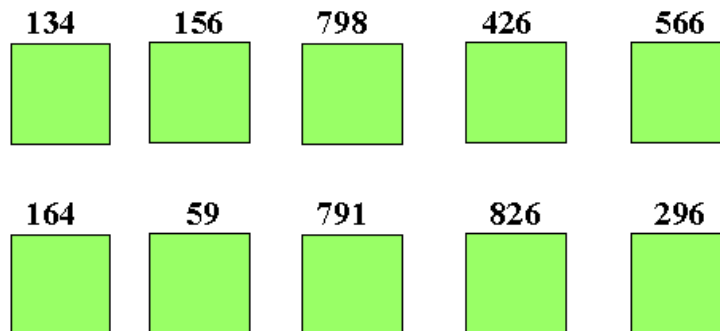
OS教学组

文件目录

File 1	1235
File 2	896
...	

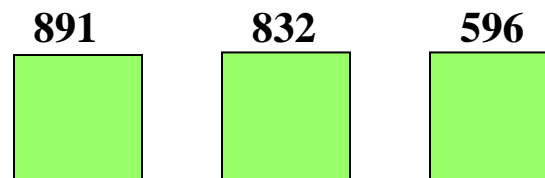
	...
0	6250
1	134
2	156
3	798
4	426
5	566
6	164
7	59
8	791
9	826
10	296
11	999
12	null

文件大小



直接索引

	999
10	891
11	832
12	596
	null
	...
	null
	null



一次间接索引

一次索引块

Flile 1	1235
Flile 3	1896



直接索引

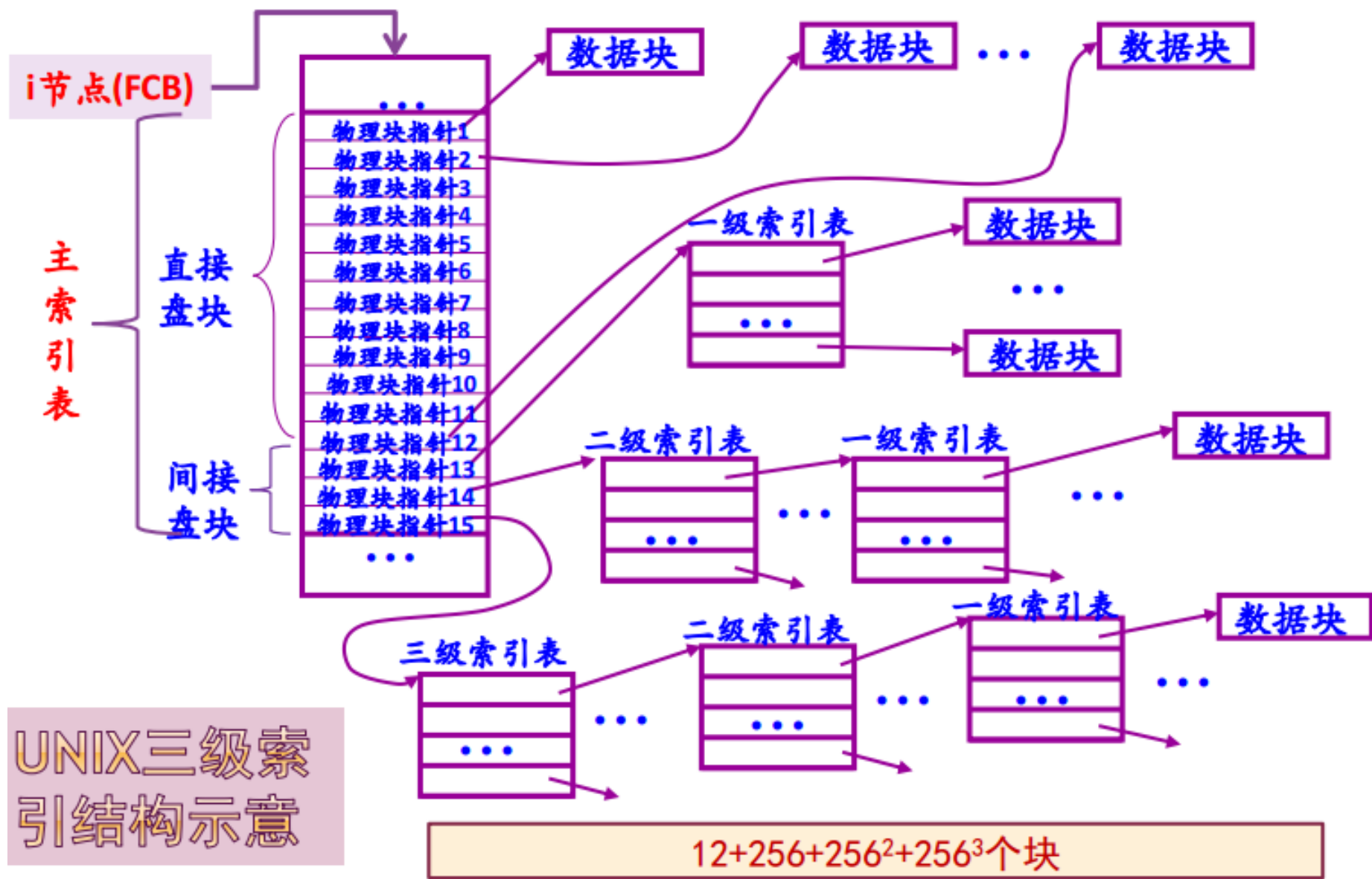
一次间接索引



二次索引块引

计算机学





UNIX三级索引结构示意

文件物理结构的比较

- **连续文件**：优点是不需要额外的空间开销，只要在文件目录中指出文件的大小和首块的块号即可，对顺序的访问效率很高。适应于顺序存取。缺点是动态地增长和缩小系统开销很大；文件创建时要求用户提供文件的大小；存储空间浪费较大。
- **串联文件**：克服了连续文件的不足之处，但文件的随机访问系统开销较大。适应于顺序访问的文件。DOS系统中改造了串联文件的结构，使其克服了串联文件的不足，但增加了系统的危险性。
- **索引文件**：既适应于顺序存访问，也适应于随机访问，是一种比较好的文件物理结构，但要有用于索引表的空间开销和文件索引的时间开销。UNIX系统是使用索引结构成功的例子。

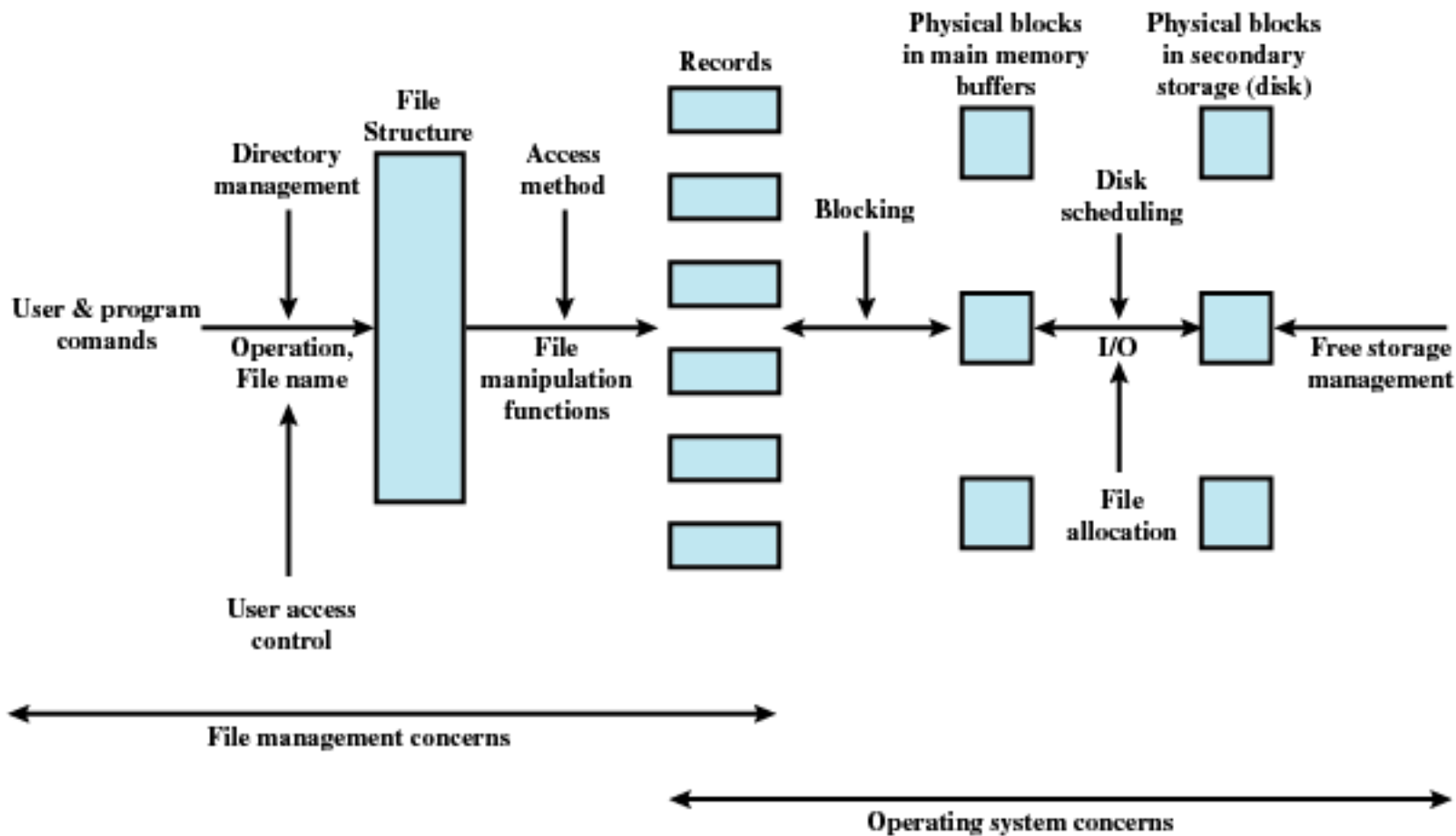
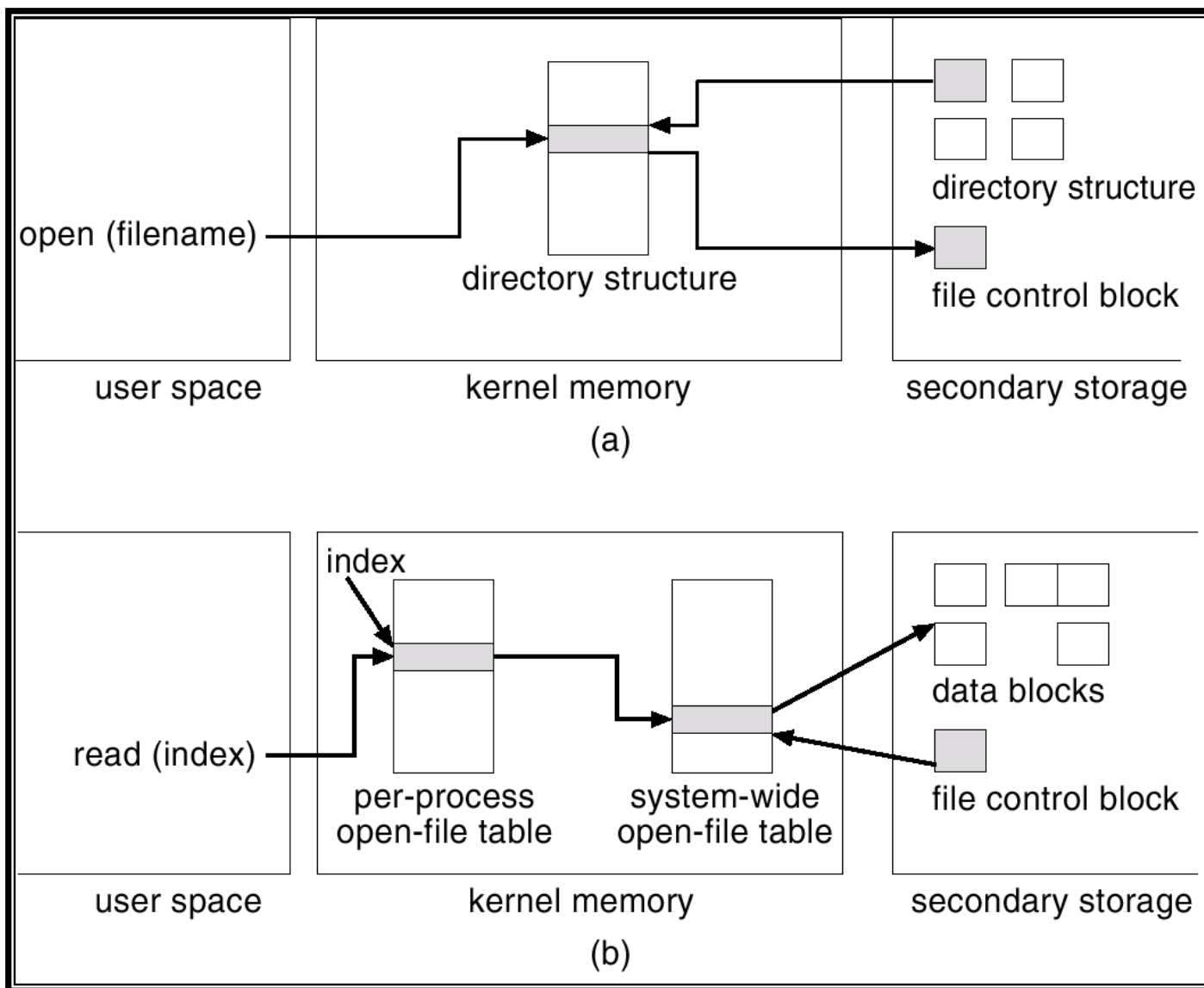
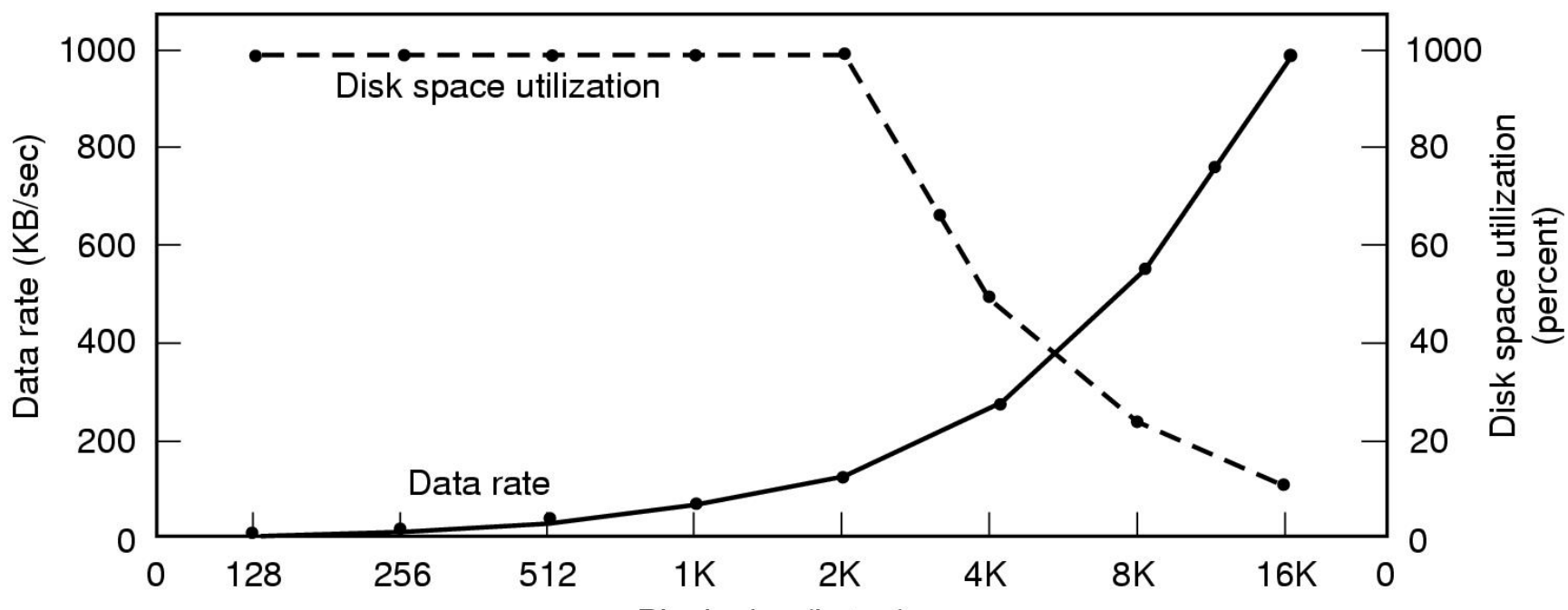


Figure 12.2 Elements of File Management



文件存储器存储空间的管理

- 空白文件目录
- 空白物理块链
- 位视图



目录的主要功能和实现

目录的主要功能：

- 根据用户给出的ASCII形式的文件名（路径名），迅速地定位到相应的文件控制块。目录的实现需解决以下三个问题：
 - 目录项的内容；
 - 长文件名问题；
 - 目录的搜索方法。

目录的实现

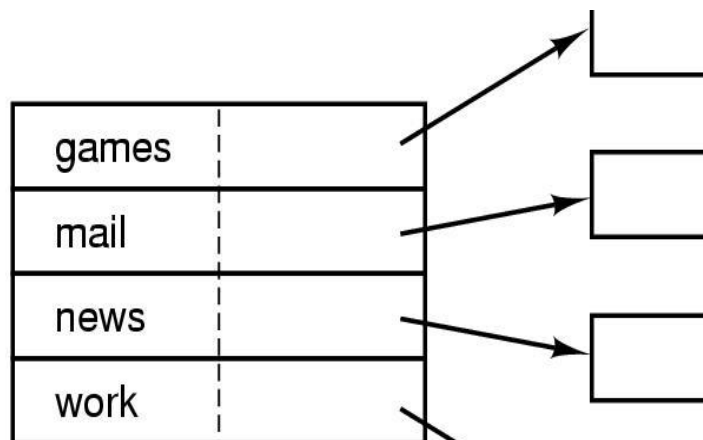
- 不同的系统采用不同的实现方法，一般分为两类：
 - 直接法：目录项=文件名+FCB（属性信息、在外存上的存放位置）。如MS-DOS/Windows；
 - 间接法：目录项=文件名+FCB的地址（索引号）。如Unix（inode）；
- 不管是何种方法，给定一个文件名，即可返回相应的FCB。

两种实现目录的方法

games	attributes
mail	attributes
news	attributes
work	attributes

(a)

直接法



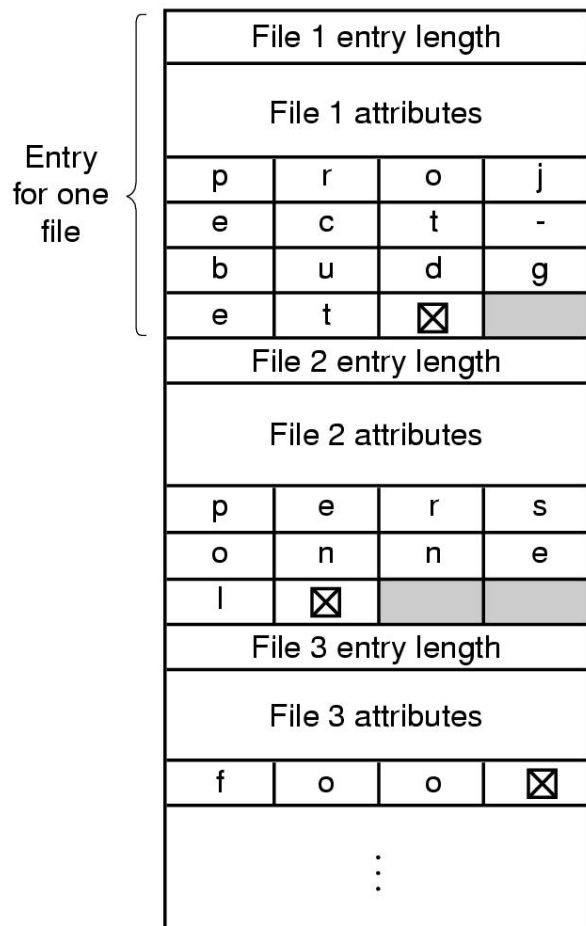
(b)

间接法

长文件名

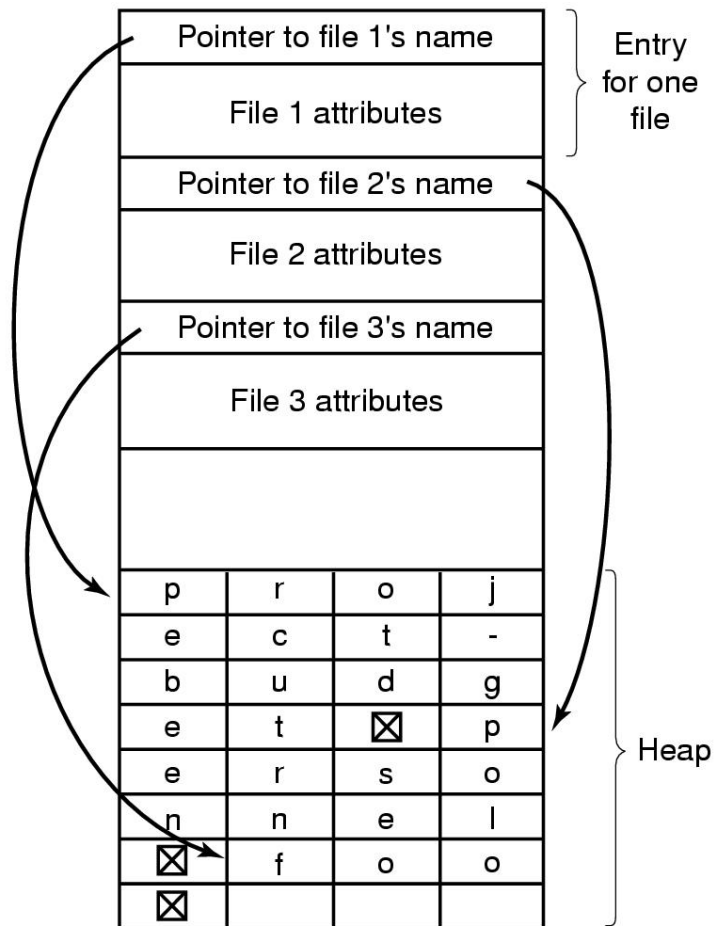
- MS-DOS：文件名的最大长度为8个字符，扩展名的最大长度为3个字符；Unix V7：文件名的最大长度为14个字符。
- 但在现代OS中，一般都支持更长的、可变长度的文件名。实现方法有：
 - **方法1**：在目录项中，将文件名的长度固定为255个字符。缺点：浪费空间，很少文件用很长的名字；
 - **方法2**：每个目录项的长度可变，分为三部分：目录项长度、文件的属性信息（此两项长度固定）、文件名（长度可变）。缺点：文件被删除后，该目录项所占用的空间不太好回收利用；
 - **方法3**：目录项本身的长度固定，把长度可变的文件名统一放在目录文件的末尾。

两种处理长文件名的方法



(a)

方法 2



(b)

方法 3

符号文件目录的查询技术

- 顺序查寻法：

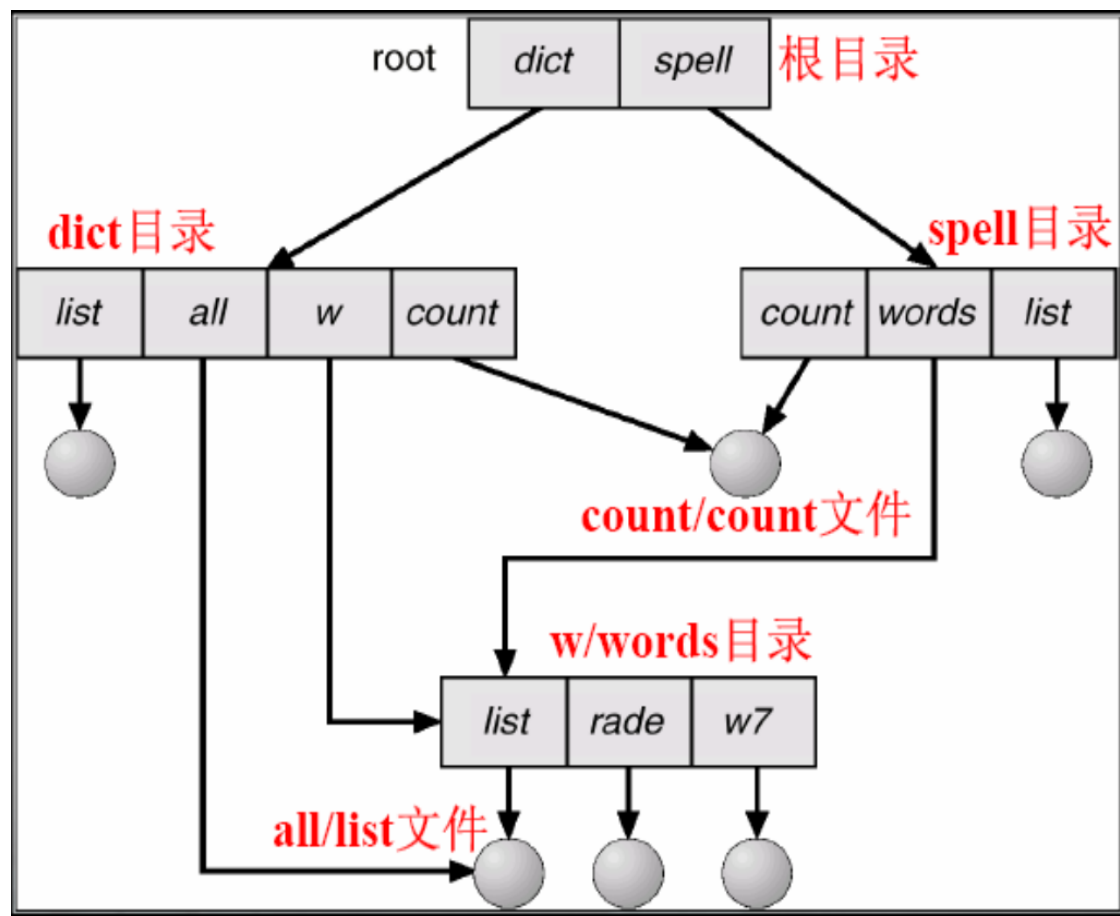
- 依次扫描符号文件目录中的表目，将表目中的名字字段与查找的符号名NAME进行比较（只在表目不多时适用）。

- Hash方法：

- 即“散列法”或成“杂凑法”，是一种构造符号表、查询符号表常用的技术。其基本思想是：利用一个易于实现的变换函数（即Hash函数），把每个符号名唯一的变换成符号表中的表目索引。

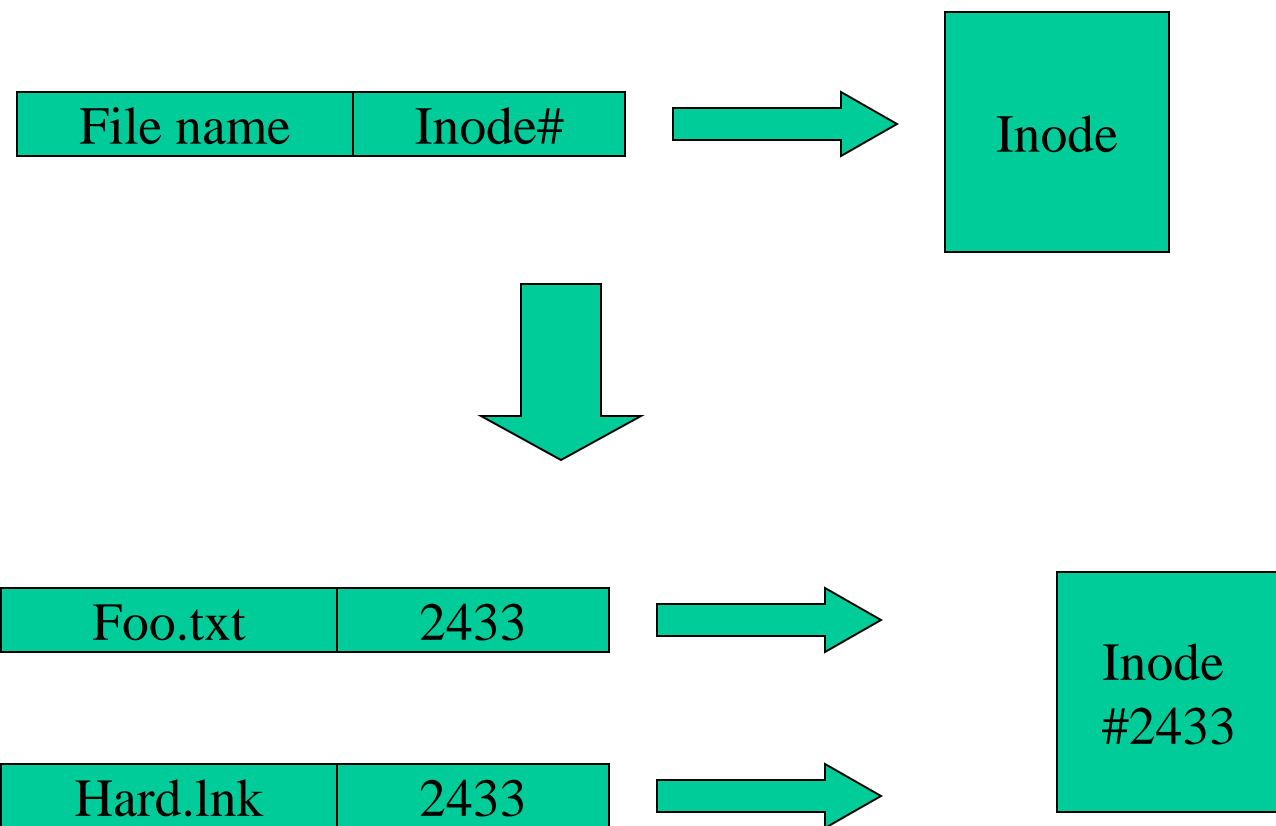
便于共享的目录组织

文件的共享，要求系统能提供某种手段，使存储空间内保存一份副本，而所有要共享该文件的用户可用相同的或不同的文件名来访问它。

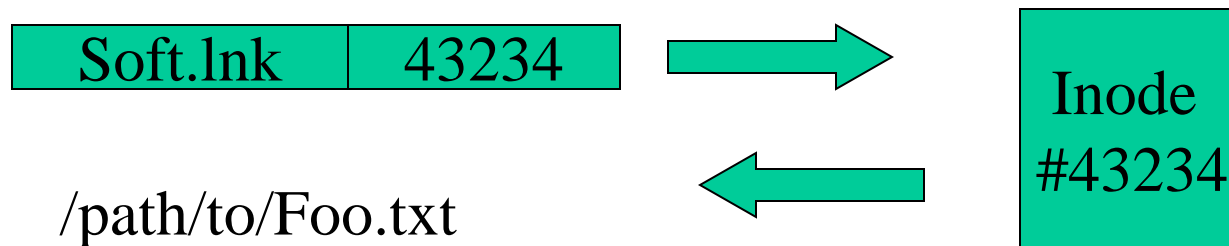


有向无环图

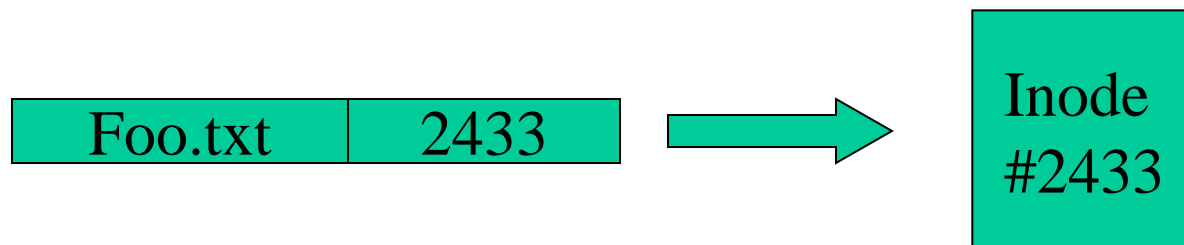
硬连接 v.s. 软连接



硬连接 v.s. 软连接



当调用open()时，重定向到 Inode #2433



保护文件的方法

- 建立副本
- 定时转储
- 规定文件的权限

建立副本

- 把同一个文件保存到多个存储介质上（同类或不同类）
 - 出故障时，可备用副本替换
- 优点
 - 方法简单
- 缺点
 - 设备费用和系统开销增大
- 适用于短小且极为重要的文件

定时转储

- 定时转储

- 每隔一定时间把文件转储到其他存储介质上，当文件发生故障，就用转储的文件来复原，把有故障的文件恢复到转储时刻文件的状态

- UNIX采用定时转储办法保护文件，可以提高文件的可靠性

文件的一致性检查

- Windows: scandisk程序
- Unix: fsck程序
- 两类一致性
 - 磁盘块的一致性：每个磁盘块设置两个计数器，一个记录在文件中出现的次数，另一个记录在空闲块中出现的次数，最终检查两个计数器是否存在不一致问题。
 - 文件的一致性：每个文件设置两个计数器，一个记录其i节点被引用的次数，另一个记录文件目录中引用它的次数，最终检查两个计数器是否存在不一致问题。

磁盘块不一致的例子

- (a) 一致状态
- (b) 丢失块2: add it to free list
- (c) 块4在空闲表中重复出现: rebuild free list
- (d) 数据块5重复出现: copy block and add it to one file

BLOCK NUMBER															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	0	1	0	1	1	1	1	0	0	1	1	1	0	0
Blocks in use															
0	0	1	0	1	0	0	0	0	1	1	0	0	0	1	1
Free blocks															

(a)

BLOCK NUMBER															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	0	1	0	1	1	1	1	0	0	1	1	1	0	0
Blocks in use															
0	0	0	0	1	0	0	0	0	1	1	0	0	0	1	1
Free blocks															

(b)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	0	1	0	1	1	1	1	0	0	1	1	1	0	0
Blocks in use															
0	0	1	0	2	0	0	0	0	1	1	0	0	0	1	1
Free blocks															

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	0	1	0	2	1	1	1	0	0	1	1	1	0	0
Blocks in use															
0	0	1	0	1	0	0	0	0	1	1	0	0	0	1	1
Free blocks															

文件的存取控制

■ 文件保护机制

- 防止未被核准的用户存取文件；
- 防止一个用户冒充另一个用户来存取；
- 防止核准用户（包括文件主）误用文件；

■ 存取权限验证步骤

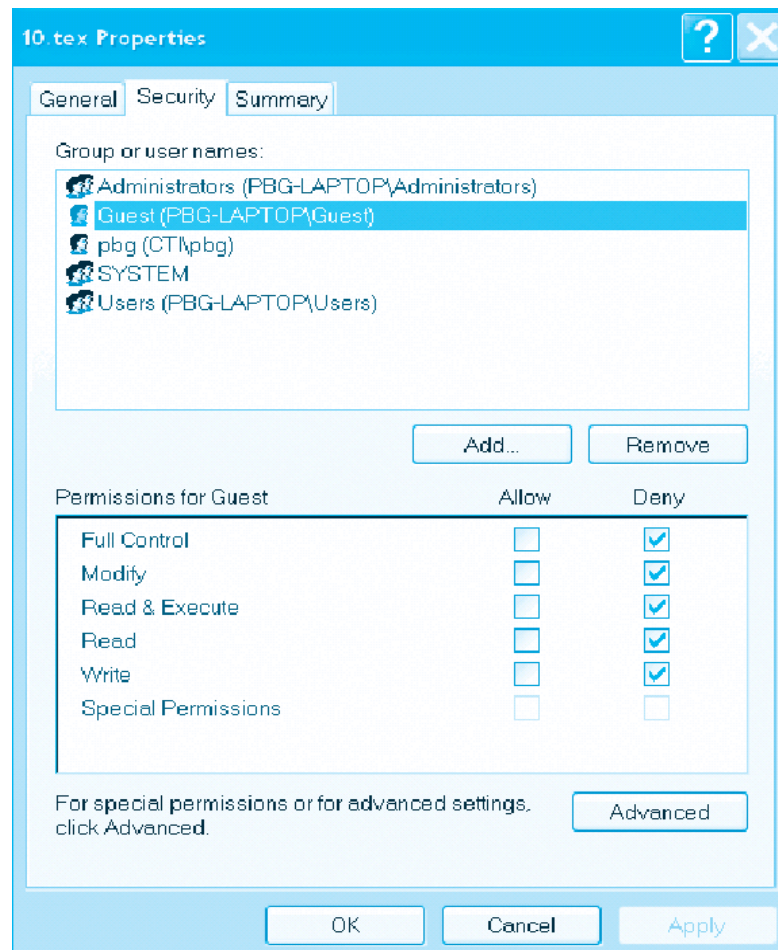
- 审定用户的权限；
- 比较用户的权限与本次存取要求是否一致；
- 将存取要求和被访问文件的保密性比较，看是否有冲突。

存取控制的实现方案

- 存取控制矩阵
- 存取控制表
- 用户权限表
- 口令

Mode of access: read, write, execute
Three classes of users

- a) owner access RWX
 $7 \Rightarrow 1\ 1\ 1$
- b) group access RWX
 $6 \Rightarrow 1\ 1\ 0$
- c) public access RWX
 $1 \Rightarrow 0\ 0\ 1$



著名的漏洞

- 中文Windows 2000 登录时，启动中文输入法看其帮助，就能够访问文件系统，从而绕过登录机制
- Unix 下，用lpr命令（lpr命令是脱机打印命令,该命令将打印作业放到打印缓冲队列中）可以打印任何文件，并有选项指定是否打印完后删除。打印口令文件并删除口令文件后，就不需要口令了。
- Finger name@site显示计算机用户的身份信息。Finger程序接受参数的缓冲区是自动变量，而且不检查溢出。将精心设计的长字符串（可以包括攻击指令等）作为参数传递给finger,会导致缓冲区溢出。缓冲区溢出后破坏栈的内容，将栈中的函数调用返回地址改写到攻击代码，那么函数调用返回时就会执行攻击代码。

安全性的设计原则

- 系统设计必须公开
- 缺省属性应该不可访问
- 检查当前权限
- 给每个进程赋予一个最小的可能权限
- 保护机制应简单一致，嵌入到系统底层
- 采取的方案必须可接受

文件的并发访问

- 文件并发访问控制的目的是提供多个进程并发访问同一文件的机制。
- 访问文件之前，必须先打开文件：如果文件的目录内容不在内存，则将其从外存读入，否则，仍使用已在内存的目录内容。这样，多个进程访问同一个文件都使用内存中同一个目录内容，保证了文件系统的一致性。
- 文件锁定(file lock)：可以协调对文件指定区域的互斥访问
 - Solaris 8中 “`int lockf(int fildes, int function, off_t size);`” 的锁定方式：
 - F_UNLOCK：取消锁定；
 - F_LOCK：锁定；如果已被锁定，则阻塞；
 - F_TLOCK：锁定；如果已被锁定，则失败返回
 - F_TEST：锁定测试；
- 利用进程间通信，协调对文件的访问

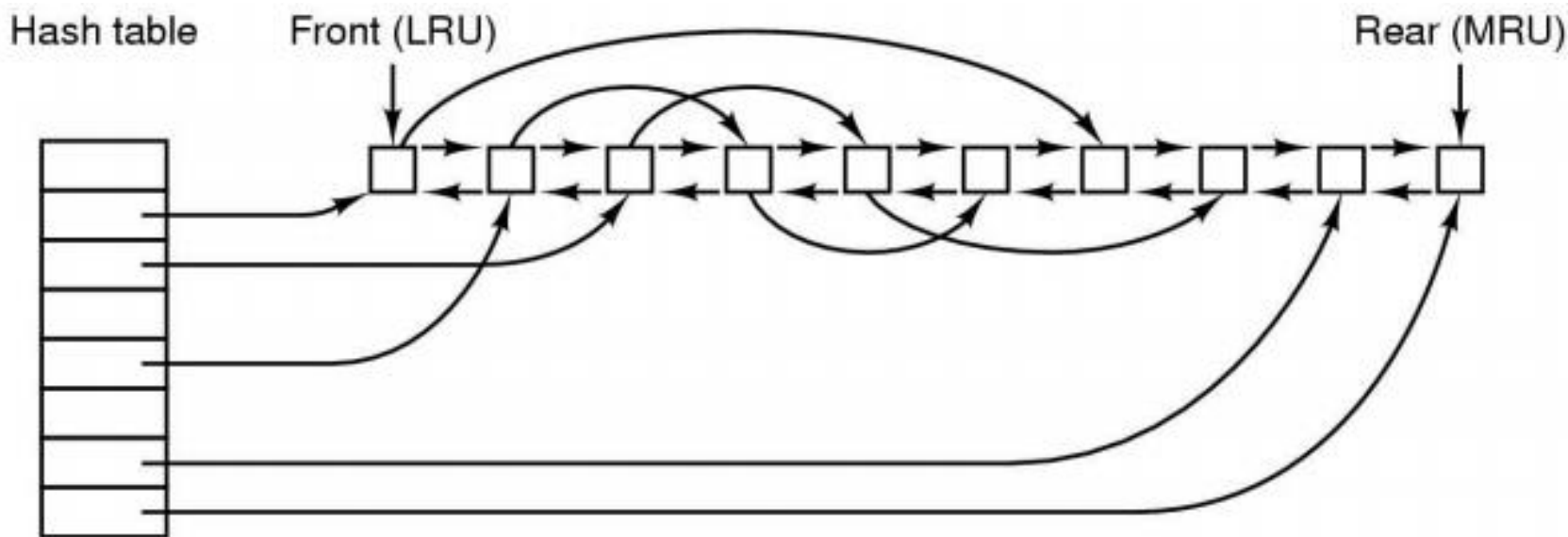
文件系统的性能问题

- 磁盘服务：速度成为系统性能的主要瓶颈之一。因此，在设计文件系统时应尽可能减少磁盘访问次数。
- 提高文件系统性能的方法：
 - 目录项（FCB）分解、当前目录、磁盘碎片整理、块高速缓存、磁盘调度、提前读取、合理分配磁盘空间、信息的优化分布、RAID技术等。

提高文件系统性能：块高速缓存

- 块高速缓存（**BLOCK CACHE**）又称为文件缓存、磁盘高速缓存、缓冲区高速缓存。是指在内存中为磁盘块设置的一个缓冲区，保存了磁盘中某些块的副本。当对文件系统进行操作的时候：
 - 检查所有的读请求，看所需块是否在块高速缓存中
 - 如果在，则可直接进行读操作；否则，先将数据块读入块高速缓存，再拷贝到所需的地方。
 - 由于访问的局部性原理，当一数据块被读入块高速缓存以满足一个IO请求时，和可能将来还会再次访问到这一数据块。

块高速缓存的组织方式



- 在块高速缓存中有若干个数据块，首先将这些块使用一个双向链表组织起来，当要访问这个链的时候就将其从此链中拿出来，然后挂接到链尾，而我们对某个文件使用的块要检查其是否在高速缓存中，所以这里又使用块号进行散列以提高检查速度。

块高速缓冲的管理

■ 块高速缓存的置换

- 此缓存的空间不会很大，当其满时我们需要对其进行置换。对于以后可能会再次使用的块我们将其放在链尾，而对于使用概率很小的块可能就需要将其剔除。

■ 块高速缓存的写入策略

- 在文件系统中，我们需要考虑该块是否会影响文件系统的一致性（高速缓存中的数据与相应磁盘块中的数据是否一致）。

基于日志结构的文件系统

Mendel Rosenblum and John K. Ousterhout. 1992. The design and implementation of a log-structured file system. *ACM Trans. Comput. Syst.* 10, 1 (February 1992), 26-52.

提出背景

- 大多文件系统都使用缓存来存储最近访问的数据块
 - 大多读操作都可以基于缓存数据来处理，因而无需读取硬盘
- 从磁盘的角度来看，主要以写操作为主
 - 为了提高磁盘的I/O速度，需要提高写数据的速度
- 然而，磁盘的性能主要受到磁头的移动所带来的开销
- 当前文件系统中，写一个数据块需要几次写操作（写数据、修改元数据），因而需要多次移动磁头来进行寻道

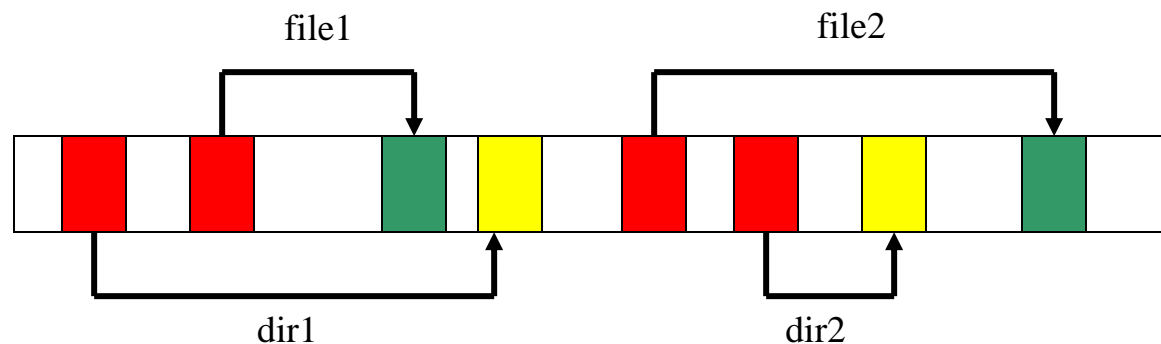
LFS: 基本思路

- 将磁盘看作一个日志系统: *log*
- 日志log是一个数据结构, 所有的写操作仅在日志头来完成, 也就是对日志的追加(*append*)操作
- 如果将磁盘按照日志进行管理, 就能够避免为寻道而进行的磁头移动
- “文件”总是顺序地添加到磁盘上
- 新的数据块和元数据 (*inode*、目录)先放在缓存中, 然后 一次性写入到硬盘 (e.g., segments of .5M or 1M)
- 这种方式能够有效提高磁盘的吞吐量

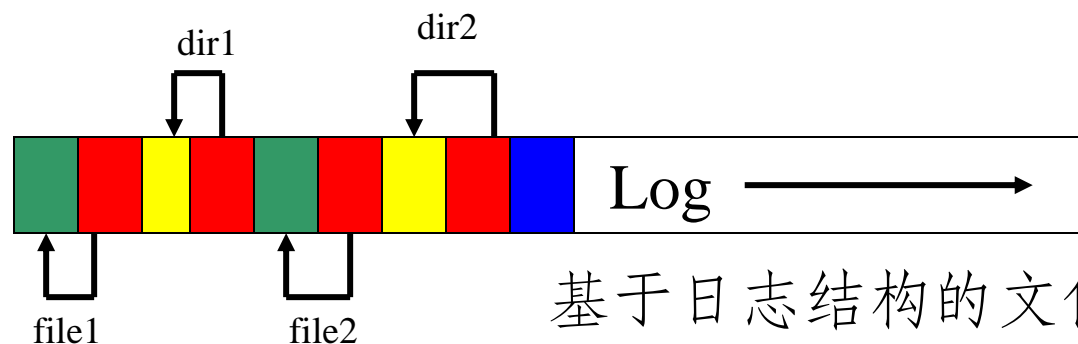
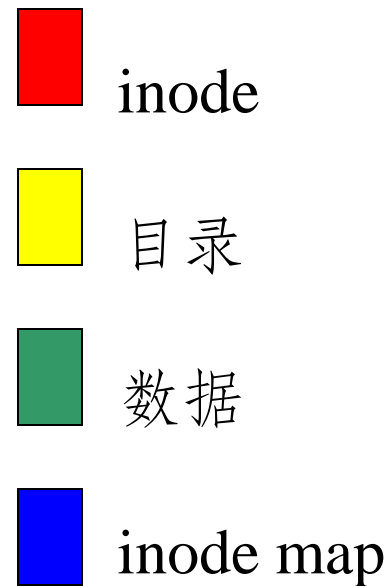
LFS的数据结构

- 段 (Segment) : 包含数据块和元数据的日志
- inode: 与Unix的inode一样, 包含文件的物理块指针
- inode map: 一个存放inode节点在磁盘上的位置的表
 - inode map blocks are written as part of the segment; a table in a fixed checkpoint region on disk points to those blocks
- 段摘要: 段中每一个数据块的信息
- 段使用情况表: 段数据块中的有效数据的量

LFS vs. UFS



Unix 文件系统



基于日志结构的文件系统

在Unix 文件系统和LFS
下创建两个包含一个
数据块文件：
dir1/file1 和 dir2/file2

LFS: 读写操作

- 每个写操作的执行都会在内存的segment缓冲区加入新的数据块；当segment写满了之后,其中的数据会写到磁盘
- 读操作与Unix文件系统基本相同，只是在寻找inode时需要用到缓存在内存中的inode map
- 随着不断的读写操作，由于新的数据块会不断替换文件中原来的数据块，结果segment会逐渐碎片化
- 问题：在出现碎片的情况下，如何确保写操作所需要的连续的空闲磁盘空间？

LFS: 失效恢复

- 检查点和向前滚动: Checkpoint , roll-forward
- 恢复的速度很快:
 - 不需要 fsck 程序来检查整个磁盘
 - 恢复最后一个检查点及其之后的数据
 - 最后一个检查点之后的数据可能部分丢失
 - 只要数秒的时间, 而Unix文件系统的恢复可能需要几个小时

LFS: 清理

- LFS的主要问题就是清理: *cleaning*, 其目的是为了提供连续的空闲磁盘空间
- 清理进程对segment进行清理, 也就是把没有填满的segment进行紧缩, 从而腾出更多的空闲空间
- 清理进程对磁盘中的segment进行选择的依据是:
 - 利用率: 清理一个segment是否能够有效提高磁盘的利用率
 - 生命周期: 一个segment是否很快就会被改变
- 清理进程会区分segment的冷热程度 (被访问的频度): 对于冷的segment, 其利用率达到75%时会被清理; 对于热的segment, 其利用率达到15%时即被清理。
(because it's worth waiting on "hot" segments for blocks to be rewritten by current activity)

LFS:小结

- 基本思路：利用缓存处理读操作，通过向日志中添加大的数据段来处理写操作
- 大幅度提高了磁盘的性能：写、创建文件、删除文件等
- 不能由缓存处理的读操作，其性能与通常的文件系统是一样的
- 需要清理进程来产生空闲磁盘空间，会由此带来附加的CPU开销

内容提要

- 文件系统基本概念
- 文件系统实现方法
- 文件系统实例分析

MS DOS的文件系统

■ 主要特点

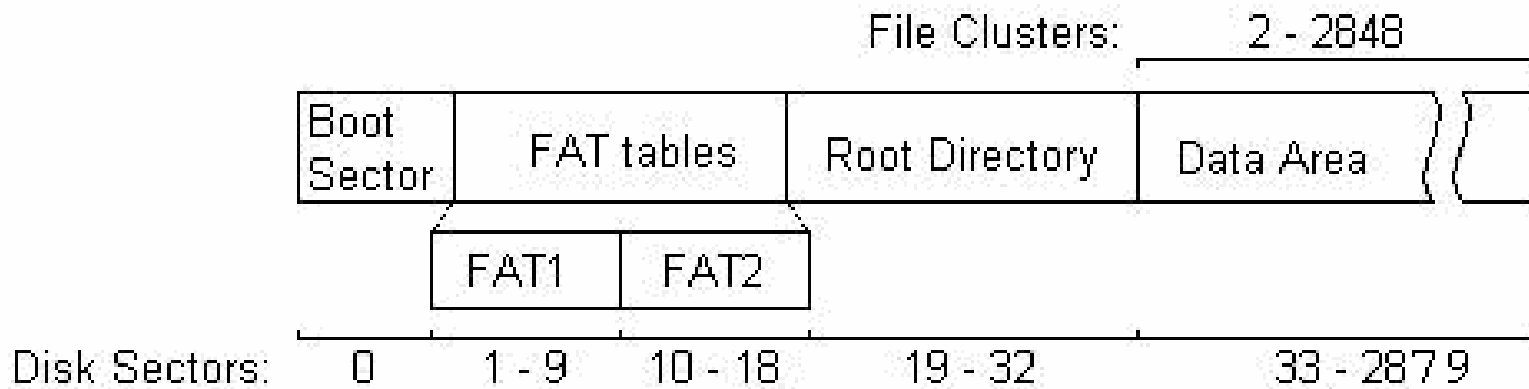
- 多级目录
- 无文件别名
- 无用户访问权限控制

■ 发展与分类

- FAT12: 最大支持32MB的磁盘空间管理
- FAT16: 最大支持2GB
- FAT32: 最大支持2TB



FAT文件系统分区结构



<http://www.eit.lth.se/fileadmin/eit/courses/eitn50/Projekt1/FAT12Description.pdf>

FAT文件系统

- 簇（块）大小：1、2、3、8、16、32或64扇区；
- 文件系统的数据记录在“引导扇区”中；
- 文件分配表（FAT，File Allocation Table）的作用：
：描述簇的分配状态、标注下一簇的簇号等；
- FAT表项：2字节（16位）；
- 目录项：32字节；
- 根目录大小固定。




FAT文件系统：主引导记录MBR

字节偏移量 (16进制)	域长	含义
00 – 1BD	446字节	引导代码
1BE – 1CD	16字节	分区表项1
1CE – 1DD	16字节	分区表项2
1DE – 1ED	16字节	分区表项3
1EE – 1FD	16字节	分区表项4
1FE – 1FF	2字节	扇区结束标记 (55AA)

MBR	分区表	分区1(主分区)	分区2	分区3
-----	-----	----------	-----	-----

FAT文件系统：分区引导扇区DBR

引导区	文件分配表1	文件分配表2	根目录	其他目录和文件
-----	--------	--------	-----	---------



字节偏移量 (16进制)	域长	样值 (16进制)	含义
00	3字节	EB 3C 90	转移指令
03	8字节	MSDOS5.0	文件系统标志(ASCII码)
0B	25字节		BIOS参数块 (BIOS Parameter Block, BPB)
24	54字节		扩展BIOS参数块(Extended BIOS Parameter Block, EBPB)
5A	410字节		引导代码
1FE	2字节	55 AA	扇区结束标记

引导扇区 (BIOS参数块)

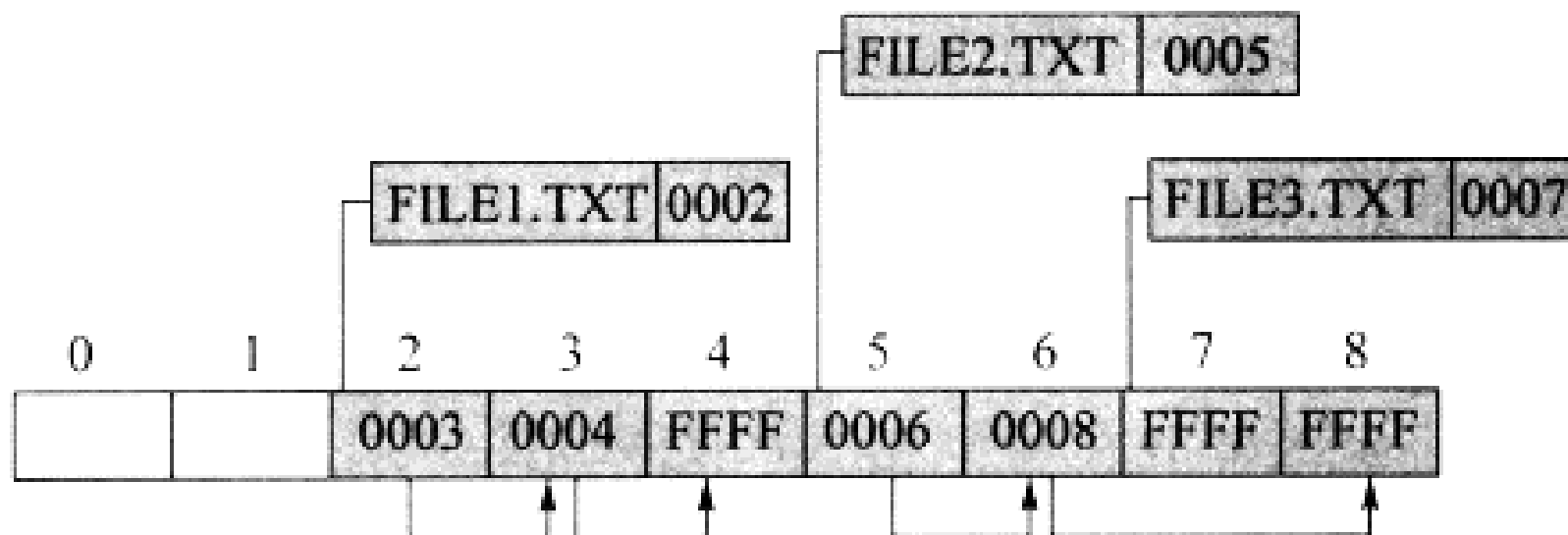
字节偏移量 (16进制)	域长	样值(16进制)	含义
0B	2字节	00 02	每扇区字节数
→ 0D	1字节	08	每簇扇区数
0E	2字节	01 00	保留扇区数: 从分区引导扇区到第一个文件分配表开始的扇区数
10	1字节	02	文件分配表个数
→ 11	2字节	00 02	根目录项数
13	2字节	00 00	扇区数(小): 卷上的扇区数, 如果该数适合于16位(65535)的话
15	1字节	F8	介质类型: F8表明为硬盘, F0表明为软盘
16	2字节	C9 00	每个文件分配表的扇区数(FAT32不用)
18	2字节	3F 00	每磁道扇区数
1A	2字节	10 00	磁头数
1C	4字节	3F 00 00 00	隐藏扇区数
20	4字节	51 42 06 00	扇区数(大): 如果小扇区数域的取值为0, 该域包含的是卷中的扇区总数

引导扇区（扩展BIOS参数块EBPB）

字节偏移量(16进制)	域长	含义
→ 24	4字节	每个文件分配表的扇区数(FAT32用)
28	2字节	标记: bit7为1, 表示只有一个FAT; 否则, 两个FAT互为镜像
2A	2字节	版本号
→ 2C	2字节	根目录起始簇号, 通常为2
30	2字节	FSINFO所在扇区, 通常1扇区
32	2字节	引导扇区备份, 通常是6号扇区
34	12字节	未用
40	1字节	BIOS Int 13H设备号
41	1字节	未用
42	1字节	扩展引导标识, 如果后面三个值有效, 设为0x29
43	4字节	卷序列号: 当格式化卷时创建的一个唯一的数字
47	11字节	卷标(ASCII码), 建立文件系统时由用户指定
52	8字节	系统ID: 根据磁盘的格式, 该域的取值为FAT12或FAT16

文件分配表FAT

- 可以把文件分配表看成是一个整数数组，每个整数代表磁盘分区的一个簇号。
- 状态：未使用、坏簇、系统保留、被文件占用（下一簇簇号）、最后一簇（0xFFFF）
- 簇号从0开始编号，簇0和簇1是保留的。



- 簇(cluster): 由若干个扇区组成。在一个文件卷中从0开始对每个簇编号。
- 对簇大小的讨论
 - 文件卷容量越大, 若簇的总数保持不变即簇编号所需位数保持不变, 则簇越大。缺点: 簇内碎片浪费越多
 - 文件卷容量越大, 若簇大小不变, 则簇总数越多, 相应簇编号所需位数越多, 可以是12, 16, 32二进制位即FAT12, FAT16, FAT32。

- FAT表：两个镜像，互为备份。文件卷中的每个簇均对应一个FAT表项，文件分配采用链式分配方法。每个FAT表项所占位数是簇编号的位数，其值是（以FAT12为例）：
 - 0：表示该簇空闲
 - FF7h：物理坏扇区
 - FF8h~FFFh：表示该簇是文件的最后一个簇
 - 其他值：表示该簇被文件占用，而且表项中的值是文件下一个簇的编号。
- FAT表大小占文件卷容量的比例： $\frac{\text{簇编号位数}}{(8 \times 512 \times \text{每个簇的扇区数})}$

FAT16目录项

偏移	域长	含义
00h	8	文件名
08h	3	文件扩展名
0Bh	1	文件属性字节
0Ch	10	保留
16h	2	最后一次修改的时间
18h	2	最后一次修改的日期
1Ah	2	起始簇号
1Ch	4	文件大小



位	7-6	5	4	3	2	1	0
	保留	归档	目录	卷标	系统	隐藏	只读

FAT32目录项

7	6	5	4	3	2	1	0
		归档	目录	卷标	系统	隐藏	只读

偏移	域长(字节)	含义
00h	8	文件名
08h	3	文件扩展名
0Bh	1	文件属性字节
0Ch	1	保留
0Dh	1	→ 创建时间，精确到1/10秒
0Eh	2	→ 文件创建时间
10h	2	→ 文件创建日期
12h	2	→ 文件最后访问日期
14h	2	→ 起始簇号的高16位
16h	2	最后一次修改的时间
18h	2	最后一次修改的日期
1Ah	2	起始簇号的低16位
1Ch	4	文件长度（子目录为0）



举例

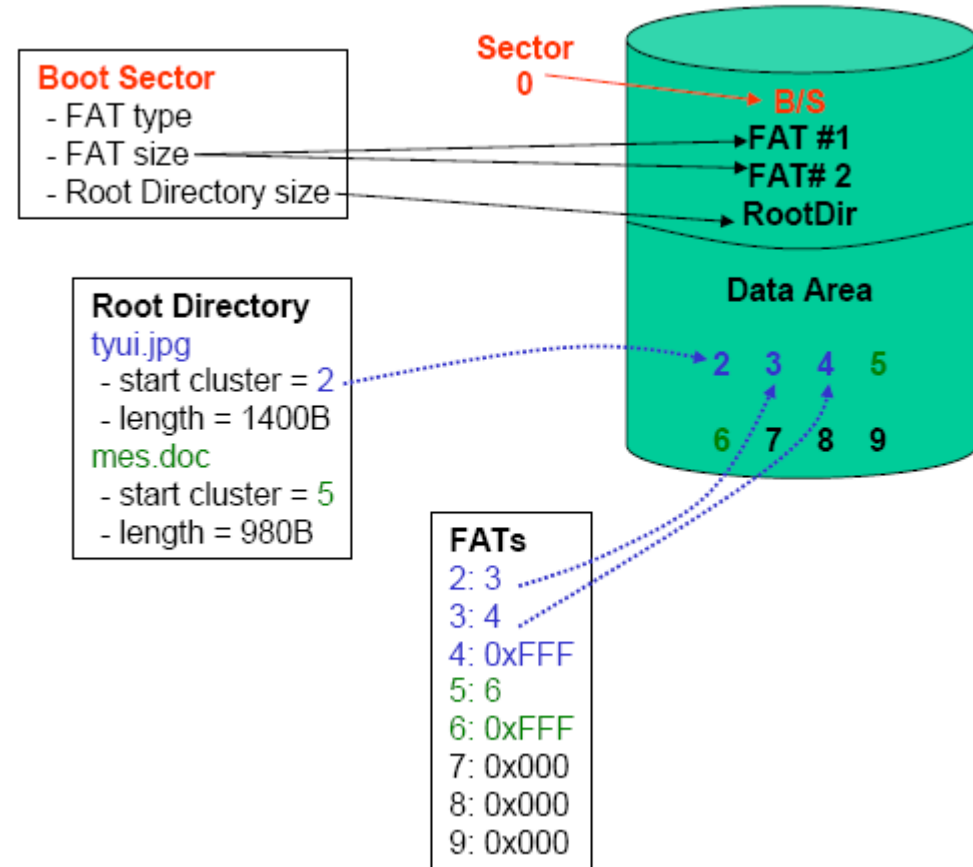
■ File **tyui.jpg**:

- occupies clusters 2, 3, and 4.
- The file size is 1,400 bytes, it occupies 1,536 bytes (3 clusters) on the disk, and cluster 4 includes 136 bytes of slack space.

■ File **mes.doc**:

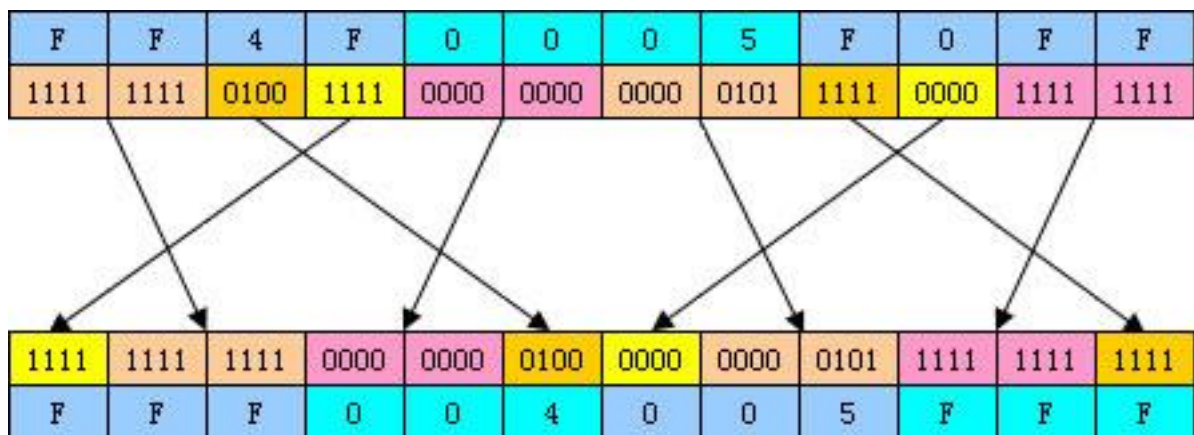
- occupies clusters 5 and 6.
- The file size is 980 bytes, it occupies 1,024 bytes (2 clusters), and has 44 bytes of slack space in cluster 6.

■ Clusters 7, 8, and 9 are unallocated.



<http://www.c-jump.com/CIS24/Slides/FAT/lecture.html>

- FF 4F 00 05 F0 FF
- 0fffh 004h 005h 0fffh



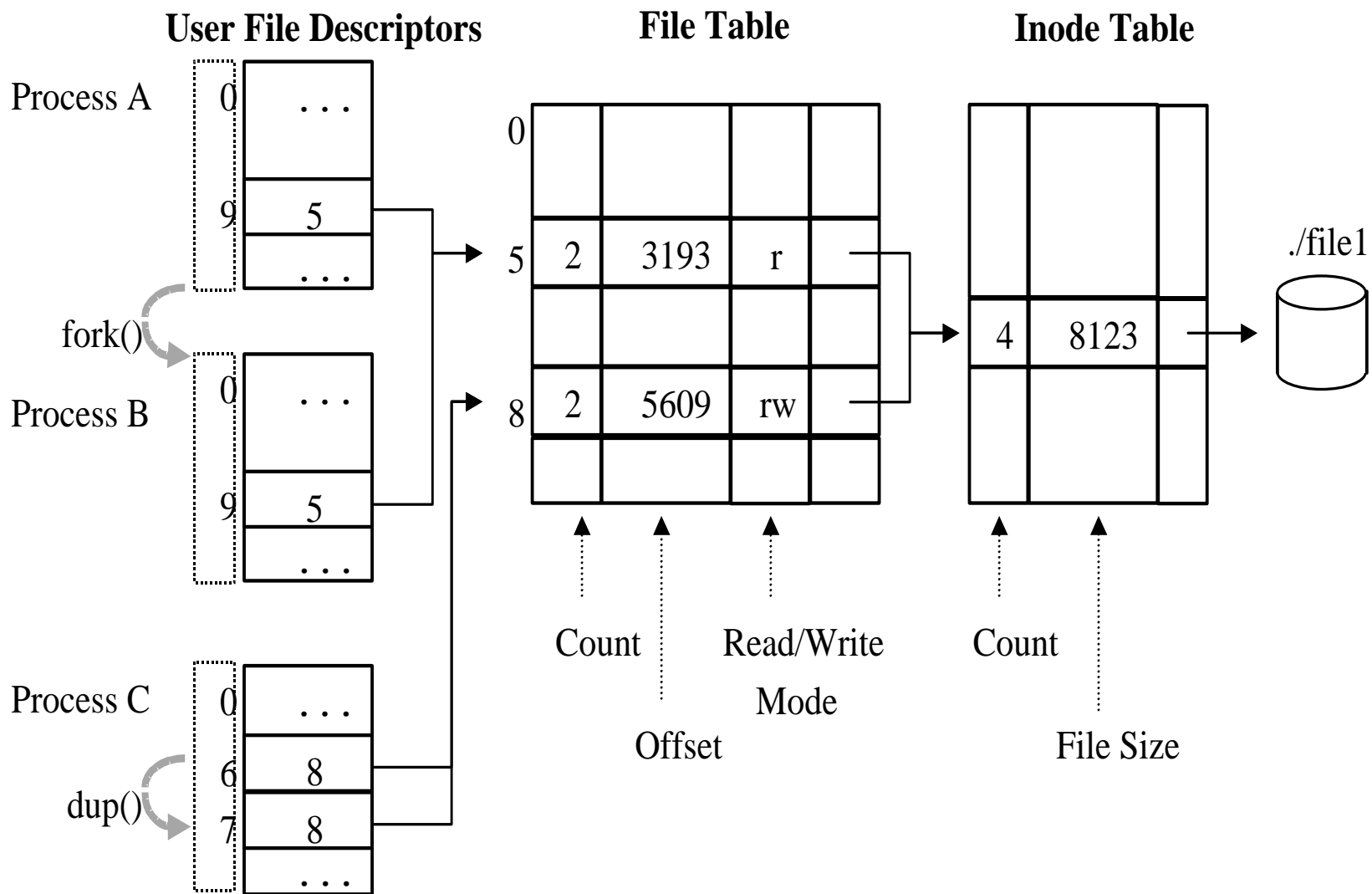
- **xor eax, eax**
- **cmp byte [bOdd], 1 ; 是奇数的簇号还是偶数的簇号**
- **jnz LABEL_EVEN_2**
- **shr ax, 4**
- **LABEL_EVEN_2:**
- **and ax, 0FFFh**

UNIX的文件系统

- 多级文件目录，可以建立文件别名（索引结点方式和符号链接方式），有用户访问权限控制（文件的读R、写W和执行X，相应于目录的检索文件、增删文件和进入目录）
 - 注意：如果对文件具有写权限，而对文件所在目录没有写权限，仍然可以改变该文件的长度（如添加数据），因为除文件名外的其他文件目录内容都存放在索引结点而不是在目录文件。
- 文件类型：常规文件(ordinary file)，目录文件(directory)，特殊文件(special file)如外设，先进先出文件(FIFO)如命名管道

Sector #	0	1		
	Boot Record	Super Block	inode Table	Data (File & Directory)

Volume Structure in UNIX



- 进程打开文件表，系统打开文件表
- 内存i结点表：空闲的内存i结点—>组织成链表；占用的内存i结点—>哈希表（对冲突采用链表方式解决）
 - 内存i结点的分配
 - 内存i结点的释放
 - 内存i结点的锁定

- 进程所进入的目录：包括各个打开文件的目录路径，以及进程的当前目录
 - 进入一个目录，如果该目录文件的内存i结点已经存在，则只需把内存i结点的引用计数加1，否则读入该目录文件的磁盘i结点并建立内存i结点。
 - 系统起动之后，进入根目录并建立相应内存i结点，直到系统关闭
 - 退出目录：如关闭文件或修改当前目录

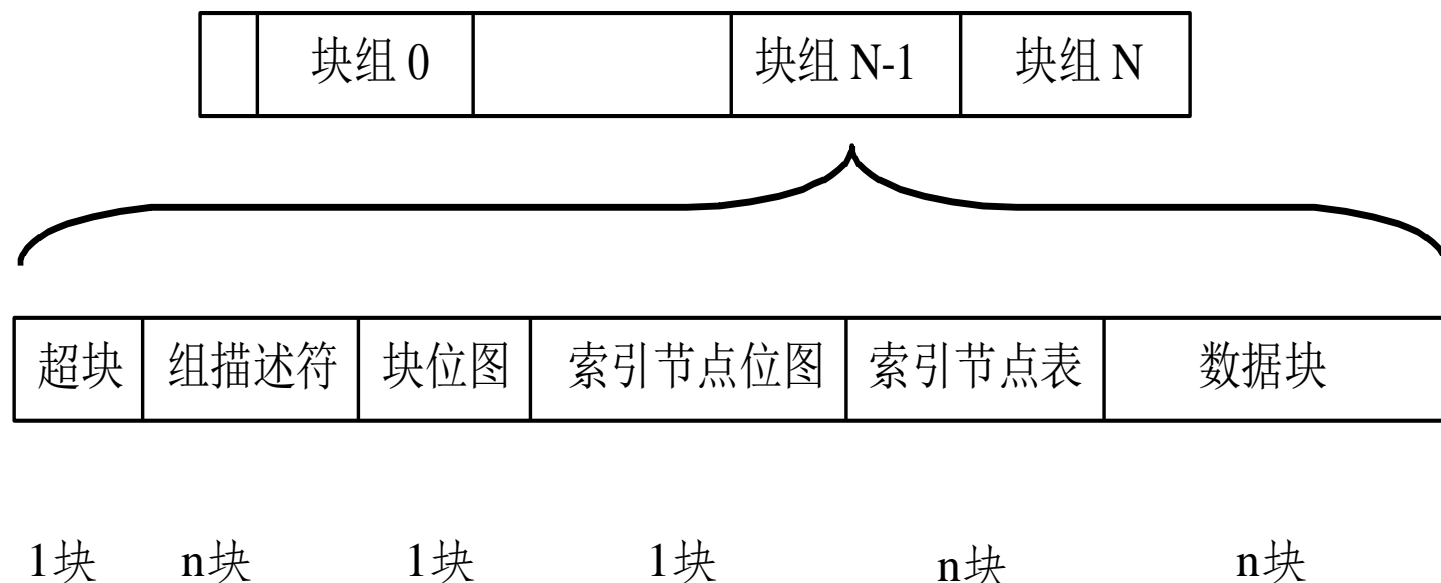
- 目录路径查找：把目录路径分解为各级目录名，然后逐级进入目录并查找。如：目录路径为 `/usr/include/sys`，则各级目录名为 `usr`, `include`, `sys`。查找过程举例：
 - 已经进入上级目录 `usr`，根据其内存 `i` 结点查找其目录文件，找到 `include` 目录项
 - 根据 `include` 目录项，建立其内存 `i` 结点，则进入 `include` 目录
 - 如果查找成功完成，则会进入目录路径中所包含的各级目录，如：打开文件为 `/usr/include/sys/types.h`，则进入的各级目录依次为 `/usr`, `/usr/include`, `/usr/include/sys`

Linux文件系统

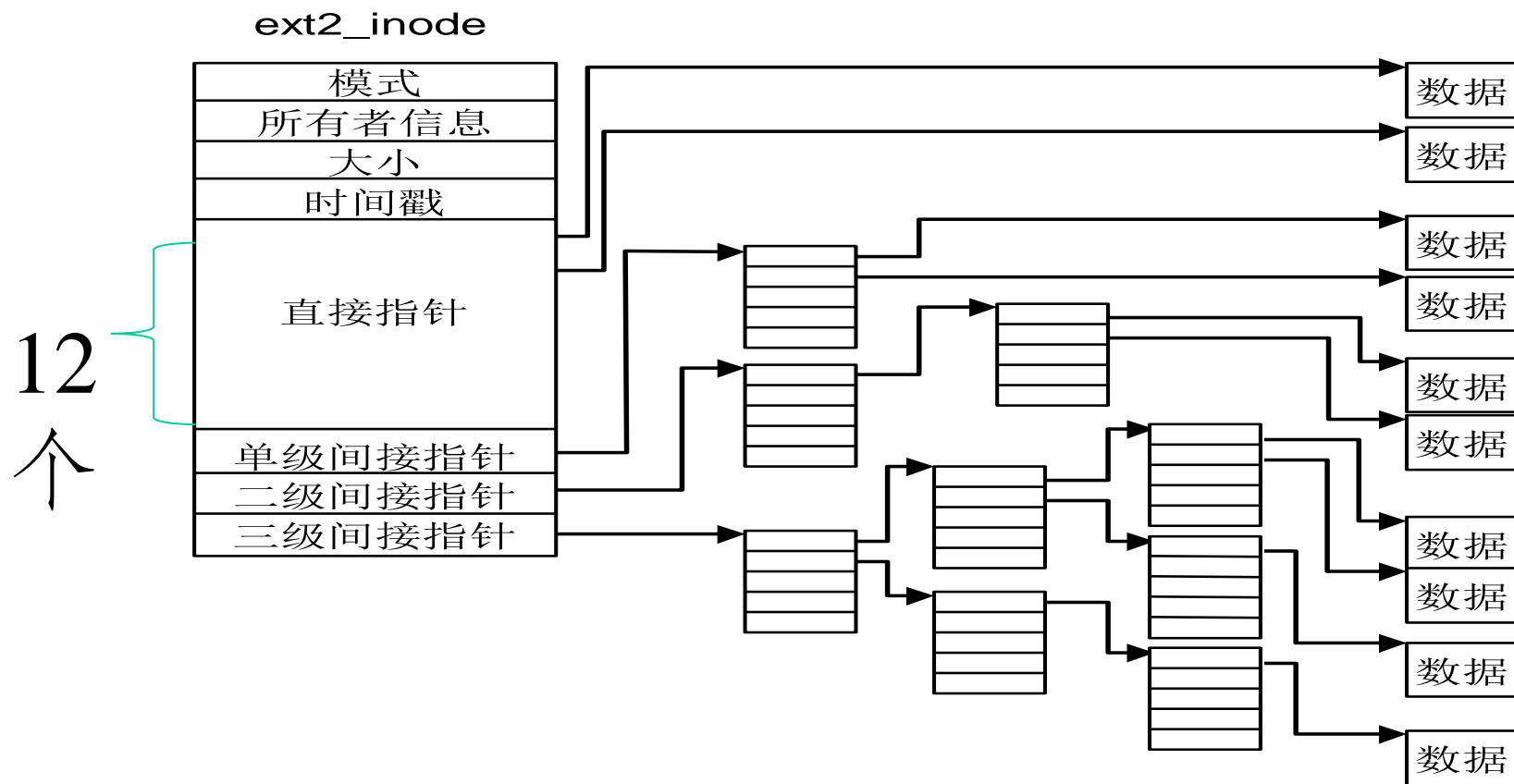
- Ext2 文件系统利用索引节点（inode）来描述文件系统的拓扑结构。在单个文件系统中，每个文件对应一个索引节点，而每个索引节点有一个唯一的整数标识符。文件系统中所有文件的索引节点保存在索引节点表中。
- Ext2 文件系统中的目录实际是一种特殊文件，它们也有对应的索引节点，索引节点指向的数据块中包含该目录中所有的目录项（文件、目录、符号链接等），每个目录项对应自己的索引节点。

EXT2硬盘组织结构

- 磁盘划分成大小相同的块（例如1KB），然后组成若干块组，每个块组有相应的描述符。



inode



- 模式：该数据域包含索引节点所描述的对象类型，以及用户的许可信息。在 Ext2 中，每个节点可描述一个文件、目录、符号链接、块设备、字符设备或一个 FIFO。
- 所有者信息：文件或目录所有者的用户和组标识符
- 大小：文件以字节为单位的大小。
- 时间戳：该索引节点的创建时间以及索引节点的最后改动时间。
- 数据块：包含该索引节点所描述的数据的数据块指针。对于前 12 个数据块指针来说，它们指向的数据块是包含实际文件数据的数据块，而后面的三个指针则包含间接数据块指针。

Ext2 文件系统的超块

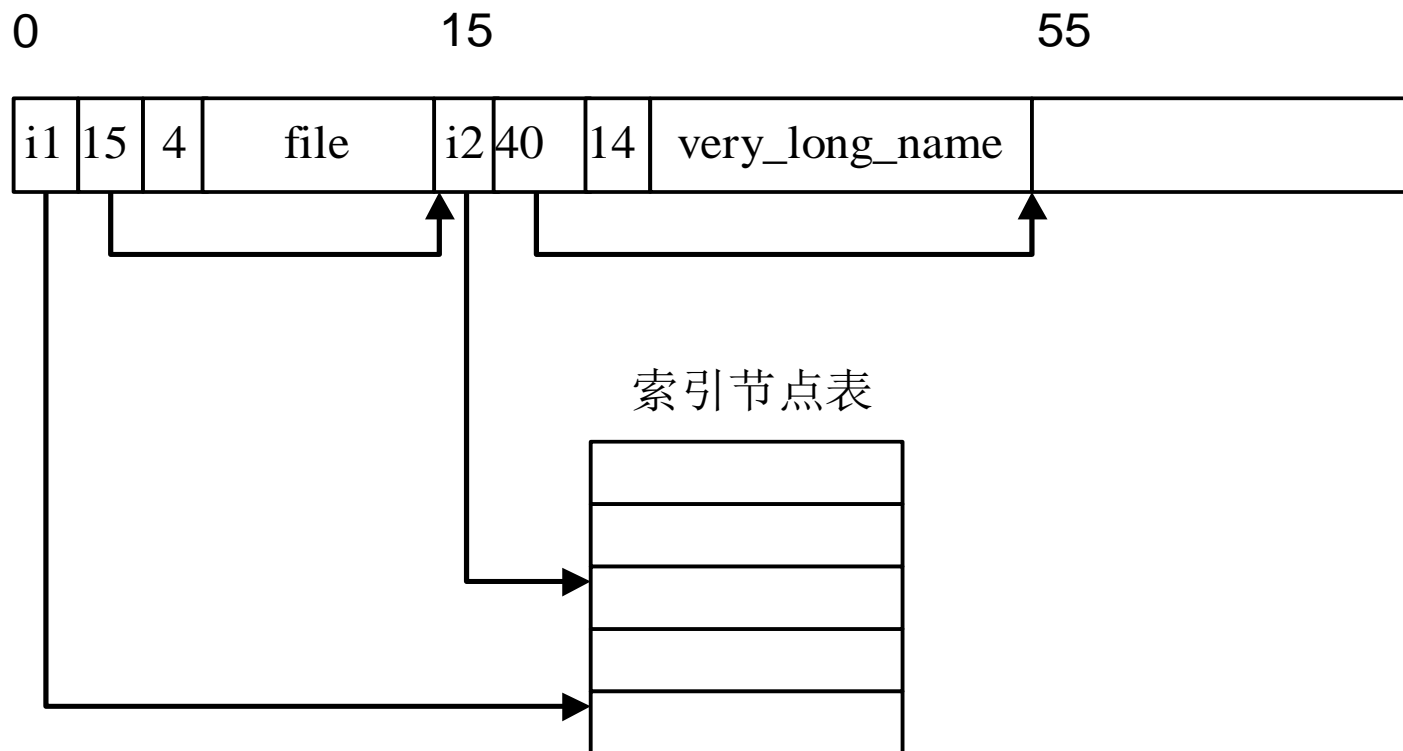
- 文件系统的超块（Superblock）包含了文件系统的基本大小和形式。其中包含的数据由文件系统管理程序用来进行文件系统的维护。每个块组中包含有**相同的**超块信息，但通常只需读取块组 0 的超块。

- 幻数：幻数用来标识超块的类型。文件系统的挂装软件可据此判断是否为有效的 Ext2 文件系统超块。当前的 Ext2 文件系统幻数为 *0xEF53*。
- 修订级别：包含两个数据，分别为主修订级别和次修订级别。文件系统的挂装软件可据此判断文件系统是否支持某种特定的功能。实际上，该信息作为兼容性标志而存在。
- 组块编号：保存该超块的组块编号。
- 数据块大小：文件系统的数据块大小，以字节为单位。
- 每组的数据块个数：每组的数据块个数，该值在建立文件系统时设置并保持不变。

Ext2 块组描述符

- 每个块组中包含有一个数据结构描述该块组。和超块类似，所有块组的块组描述符重复出现在所有的块组中。
 - 数据块位图位置：包含块组数据块分配位图的数据块编号。 每个块组包含多少块？
 - 索引节点位图位置：包含块组索引节点分配位图的数据块编号。 每个块组包含多少inode？
 - 索引节点表位置：包含块组索引节点表的起始数据块编号。
 - 空闲块计数、空闲索引节点计数、已用目录计数

目录



访问/usr/WeiYM/mbox文件过程

6号索引节点

/usr 的

索引节点内容

132号数据块

/usr 的

目录项列表

19号索引节点

/usr/WeiYM 的

索引节点内容

406号数据块

/usr/WeiYM 的

目录项列表

1	.
1	..
4	bin
7	dev
14	lib
9	etc
6	usr
8	tmp

/usr 目录的
索引节点号
为 6

模式
大小
时间
132

132号数据
块包含/usr
的目录内容

6	.
1	..
19	WeiYM
30	zhengx
51	shenys
26	zhaot
45	jianghj

/usr/WeiYM
的索引节点号
为 19

模式
大小
时间
406

406号数据
块包含
/usr/WeiYM
的目录内容

26	.
6	..
64	project
92	private
60	interest
81	photo
17	mbox

/usr/WeiYM/mbox
的索引节点号
为 17

Root目录总是位于第2号索引节点。

虚拟文件系统VFS

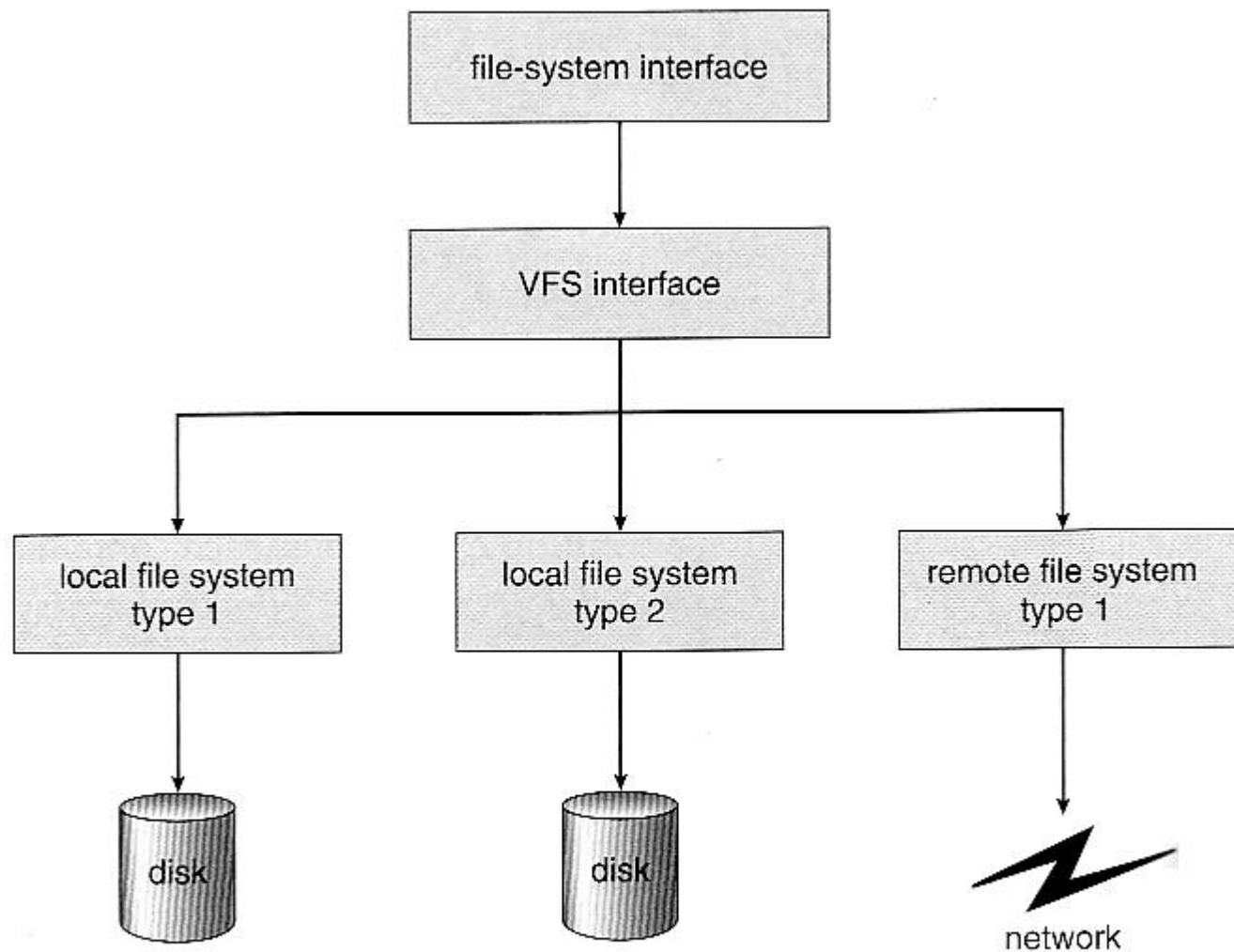
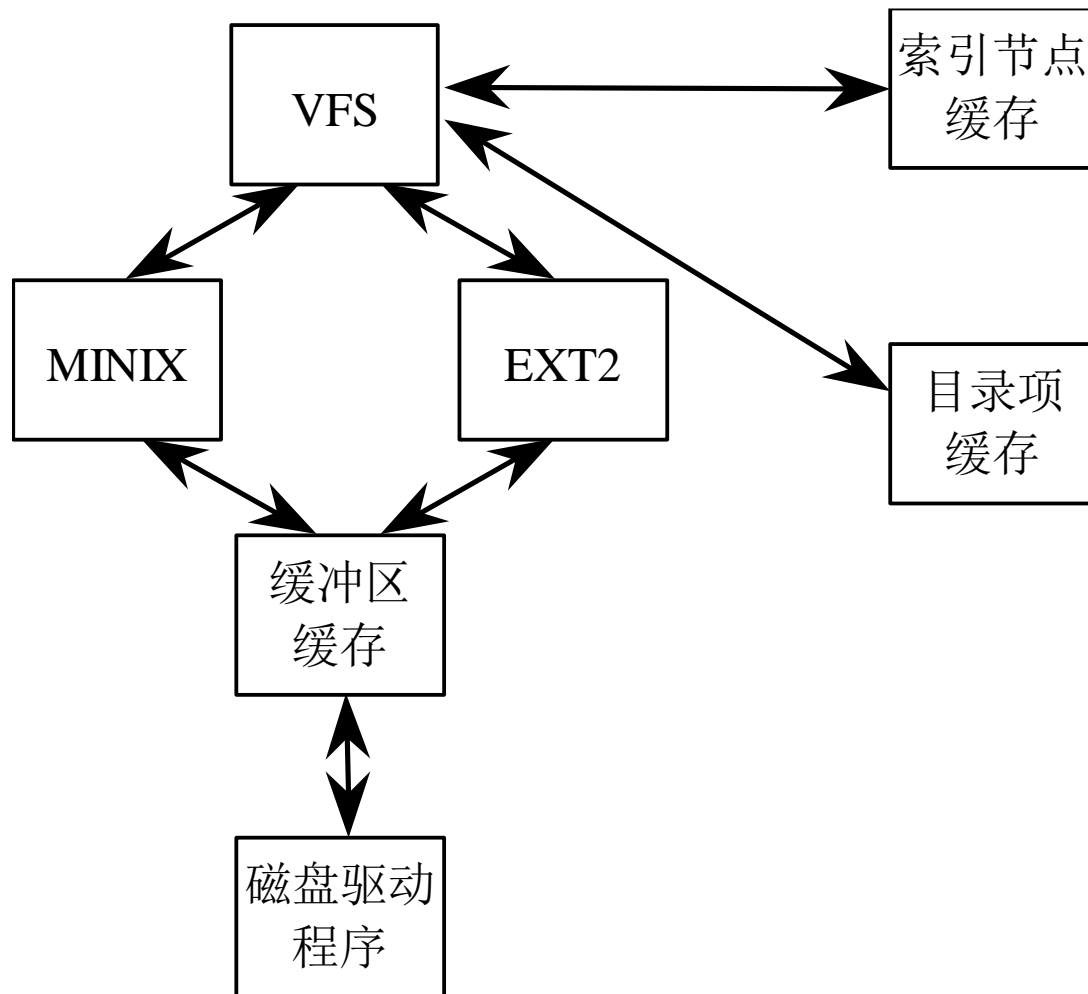
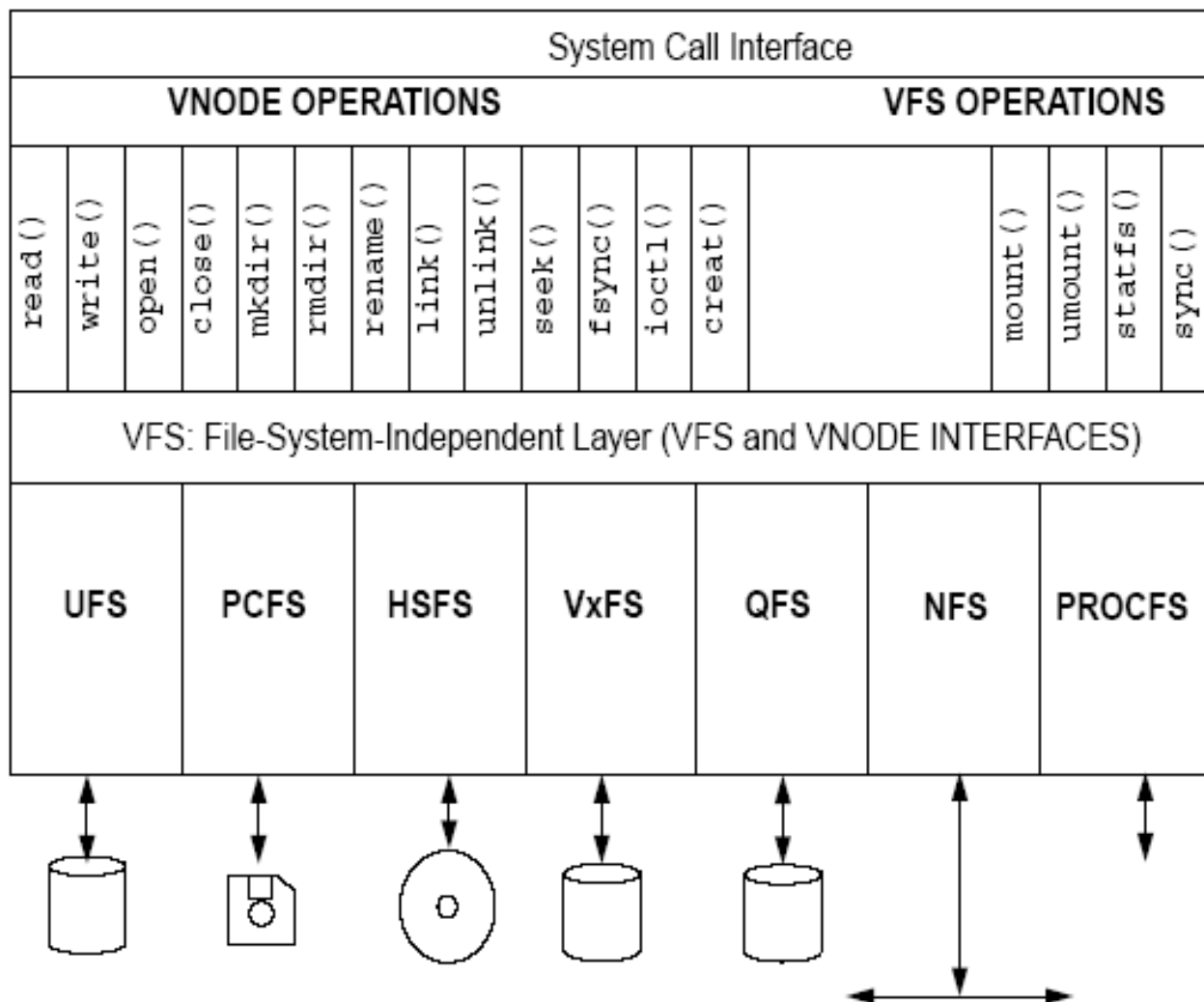


Figure 11.4 Schematic view of a virtual file system.

虚拟文件系统





VFS 超块

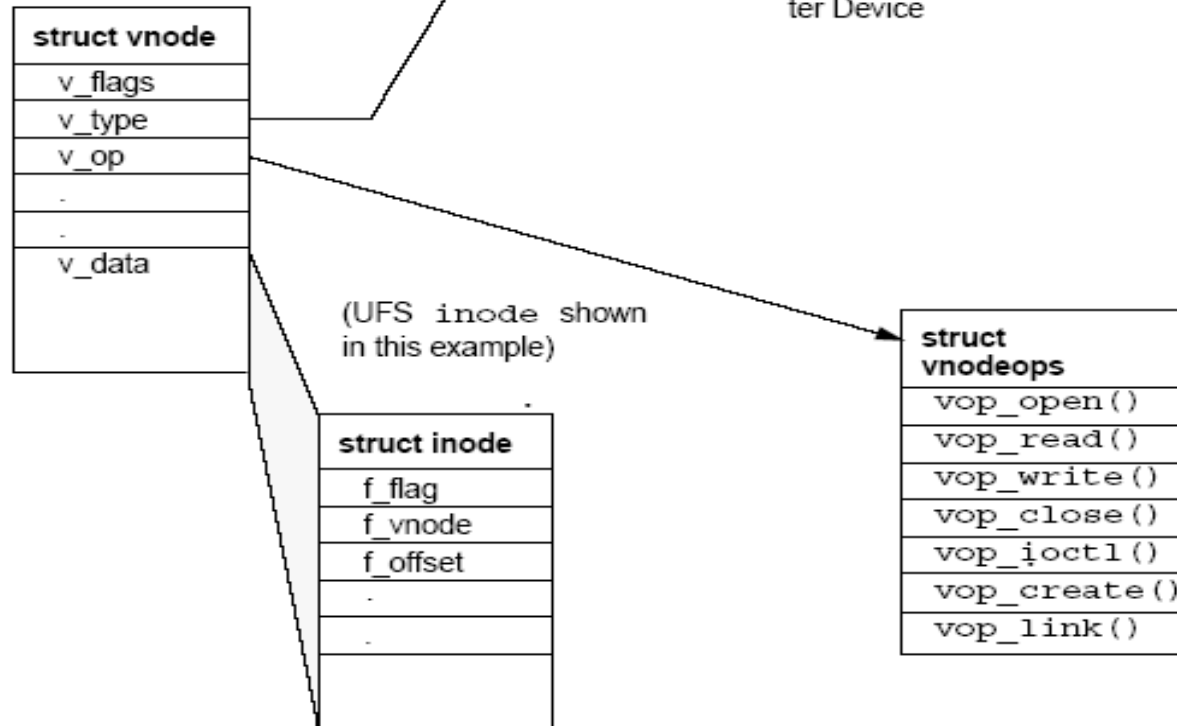
- 设备：包含文件系统的块设备标识符。对于 `/dev/hda1`，其设备标识符为 `0x301`。
- 索引节点指针：这里包含两个索引节点指针。
`mounted` 索引节点指针指向文件系统的第一个节点；
`covered` 指针指向代表文件系统挂装目录的节点。
`root` 文件系统的 VFS 超块没有 `covered` 指针。
- 数据块大小：文件系统中数据块的大小，以字节为单位。
- 超块操作集：指向一组超块操作例程集的指针。这些例程由 VFS 用来读取和写入索引节点以及超块。
- 文件系统类型：指向文件系统的 `file_system_type` 数据结构的指针。
- 文件系统的特殊信息：该文件系统的特殊信息。

VFS 索引节点

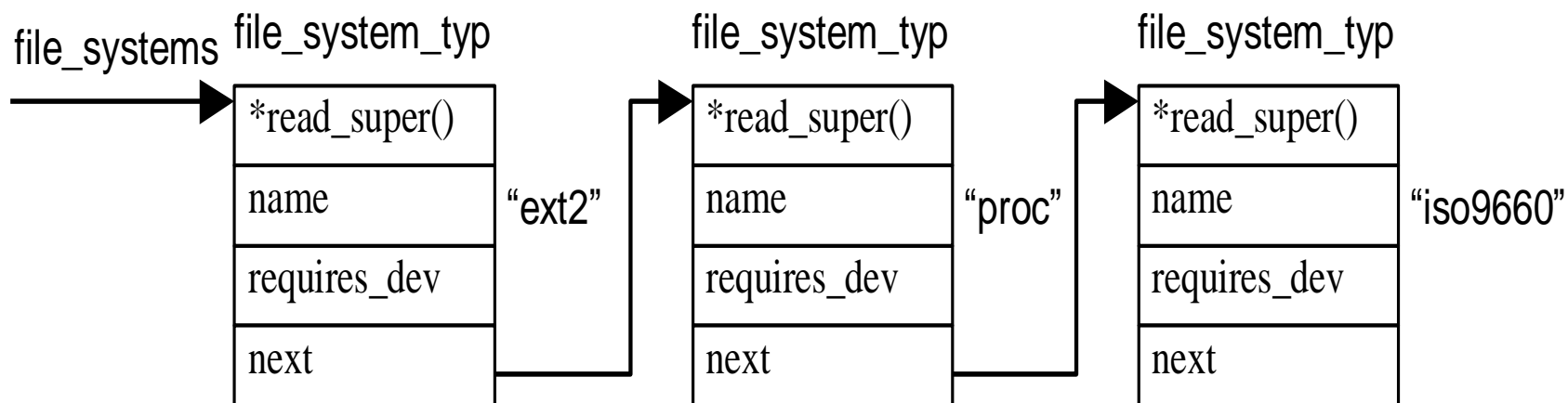
- 设备：包含该文件或其他任何 VFS 索引节点所代表的对象的设备标识符。
- 索引节点编号 索引节点编号，在文件系统中唯一。设备和索引节点的组合在 VFS 中唯一。
- 模式：VFS 索引节点代表的对象类型，以及相应的访问权限。
- 用户标识符：所有者标识符。
- 时间：建立、修改和写入的时间。
- 块大小：数据块的大小，以字节为单位。

- 索引节点操作集：指向索引节点操作例程集的指针。这些例程专用于该文件系统，用来执行针对该节点的操作。
- 计数：系统组件使用该 VFS 节点的次数。计数为零，表明该节点可丢弃或重新使用。
- 锁定：该数据域用来锁定 VFS 节点。
- 脏：表明 VFS 节点是否被修改过，若如此，该节点应当写入底层文件系统。
- 文件系统的特殊信息：和文件系统相关的特殊信息。

- VREG – Regular File
- VDIR – Directory
- VBLK – Block Device
- VCHR – Character Device



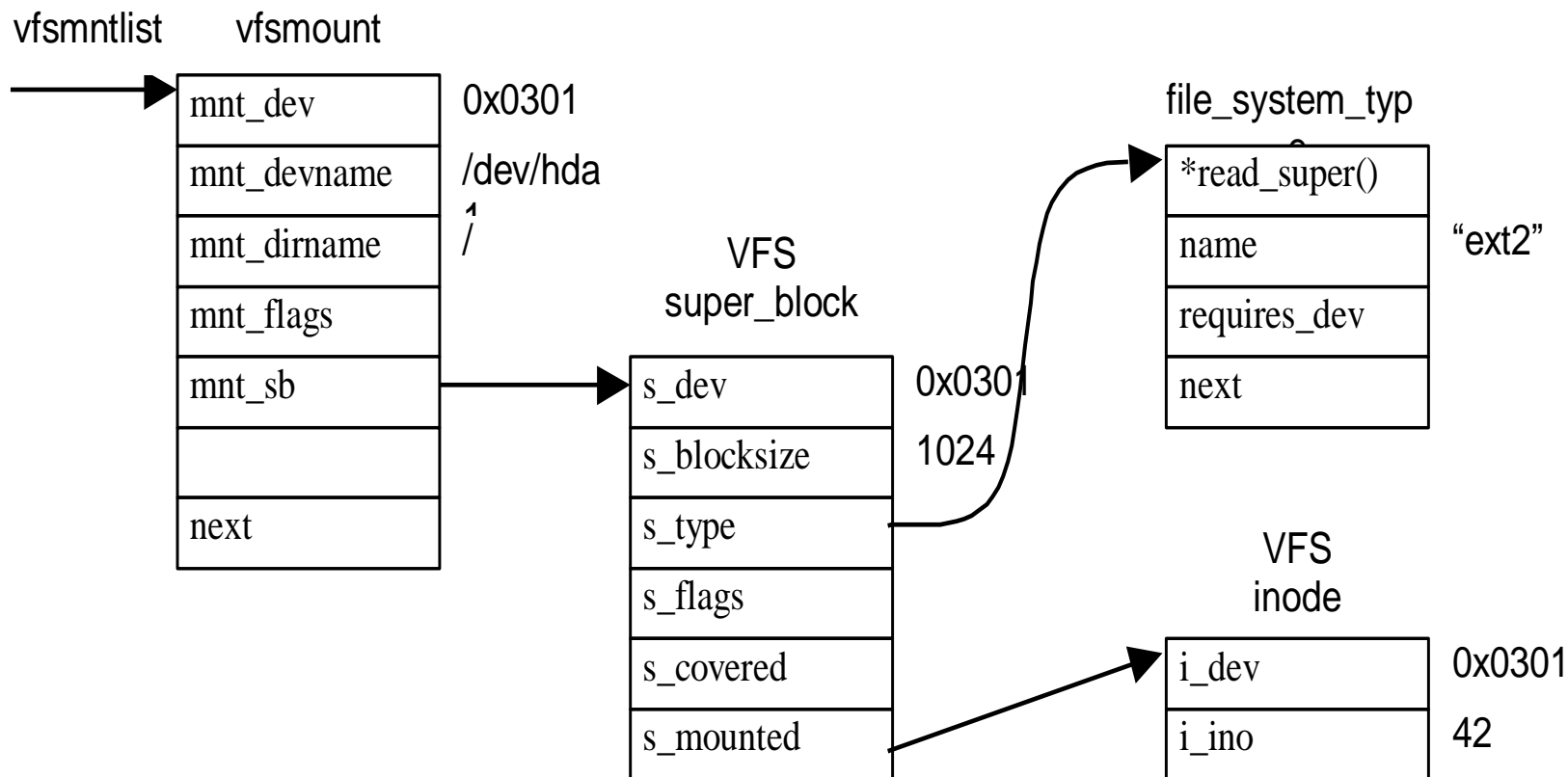
文件系统的注册



文件系统的挂装和卸装

1. 寻找对应的文件系统信息。VFS 通过 `file_systems` 在 `file_system_type` 组成的链表中根据指定的文件系统名称搜索文件系统类型信息。
2. 如果在上述链表中找到匹配的文件系统，则说明内核具有对该文件系统的内建支持。否则，说明该文件系统可能由可装载模块支持，VFS 会请求内核装入相应的文件系统模块，此时，该文件系统在 VFS 中注册并初始化。
3. 不管是哪种情况，如果 VFS 无法找到指定的文件系统，则返回错误。
4. VFS 检验给定的物理块设备是否已经挂装。如果指定的块设备已被挂装，则返回错误。

- VFS 查找作为新文件系统挂装点的目录的 VFS 索引节点。该 VFS 索引节点可能在索引节点高速缓存中，也有可能需要从挂装点所在的块设备中读取。
- 如果该挂装点已经挂装有其他文件系统，则返回错误。因为同一目录只能同时挂装一个文件系统。
- VFS 挂装代码为新的文件系统分配超块，并将挂装信息传递给该文件系统的超块读取例程。系统中所有的 VFS 超块保存在由 **super_blocks** 指向的 **super_block** 数据结构指针数组中。
- 文件系统的超块读取例程将对应文件系统的信息映射到 VFS 超块中。如果在此过程中发生错误，例如所读取的超块幻数和指定的文件系统不一致，则返回错误。



LINUX已支持的文件系统

- MINIX: MINIX 操作系统使用的文件系统, Linux 最初使用该文件系统开发。是最老的、也是最可靠的文件系统。但该文件系统的能力受到一定限制, 例如, 省略了某些时间戳, 文件名长度不能超过 30 个字符等。
- Xia: MINIX 文件系统的更新版, 突破了部分限制, 但没有新的功能特色。
- Ext2: 功能强大的 Linux 土生土长的文件系统。
- Ext: Ext2 文件系统的老版本。该文件系统可转换到 Ext2 文件系统。
- Ext3、Ext4、xfs、btrfs、ReiserFS、...

- Msdos: 和 MS-DOS 兼容的 FAT 文件系统。
- Umsdos: 对 Linux 中 msdos 文件系统驱动程序扩展，以便能够支持长文件名、所有者、许可、链接和设备文件。它使得通常的 msdos 文件系统可以当作 Linux 固有的文件系统一样使用。
- Vfat: Microsoft 对原 fat 文件系统的扩展，可以支持长文件名。
- Iso9660: 该文件系统是标准的 CD-ROM 文件系统。对该标准更加流行的 Rock Ridge 扩展允许长文件名的自动支持。
- F2FS: A New File System for Flash Storage
 - <https://www.usenix.org/conference/fast15/technical-sessions/presentation/lee>

- nfs: 允许在多台计算机之间共享文件系统的网络文件系统。
- Hpfs: 高性能文件系统, 是 OS/2 的文件系统
- Ntfs: Windows NT 的文件系统。
- Sysv: System V/386、Coherent 和 Xenix 的文件系统。

小结

■ 文件系统基本概念

抽象

- 文件系统：存储块-----→文件

■ 文件系统实现方法：用户可理解名称→系统资源

- 文件、目录
- 已分配空间管理
- 空闲空间管理

问题？