操作系统　Operating System

第七章 文件系统

沃天宇
woty@buaa.edu.cn
2021年5月18日

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

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；（data）
  - 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；（meta-data）

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。

一切皆文件

- 文件的本质是一组字节序列，源于用户程序对所要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。

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。

文件系统层次结构

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

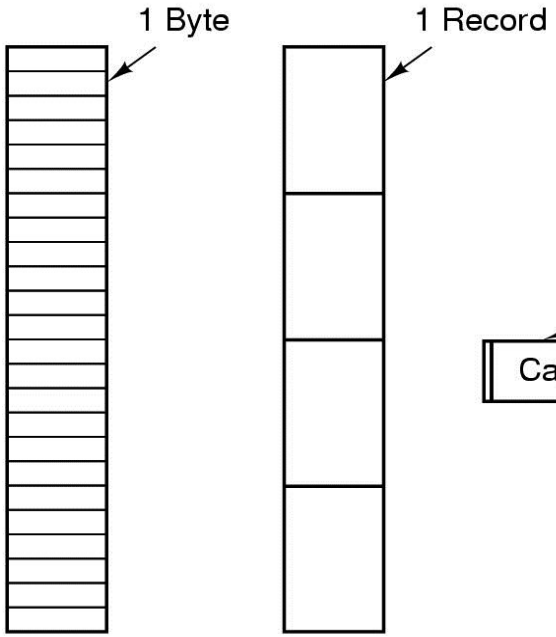
UNIX按性质和用途将文件分为：一般分为普通文件、目录文件、特殊文件（设备文件）、管道文件、套接字。

- 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。

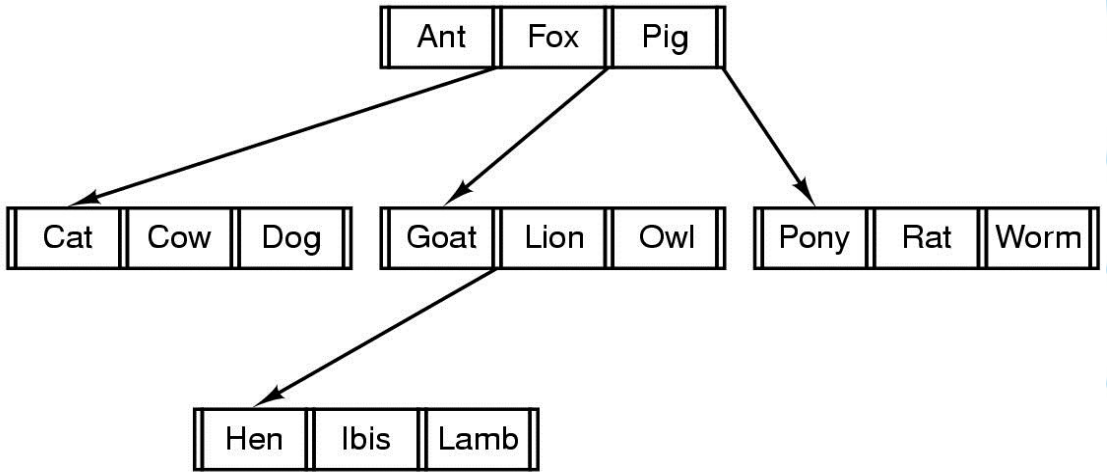
file type	usual extension	function
executable	exe, com, bin or none	read to run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, 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 compressed, 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

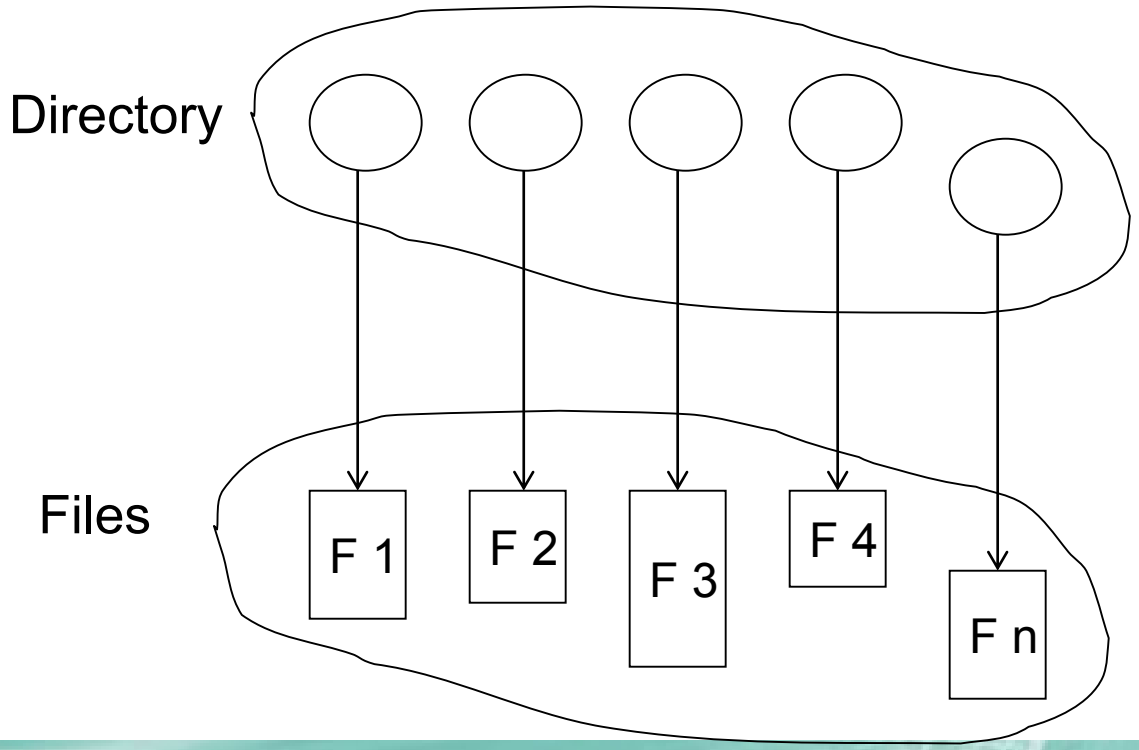
- 文件太多了怎么办？不同的应用程序有不同类型的文件，不同的用户有不同的文件，如何对它们进行组织、分类？
- 如何对文件进行管理？当用户需要访问某个文件时，如何根据这个文件名迅速地定位到相应的文件，从而对文件的属性和内容进行各种操作？

对目录构建的要求

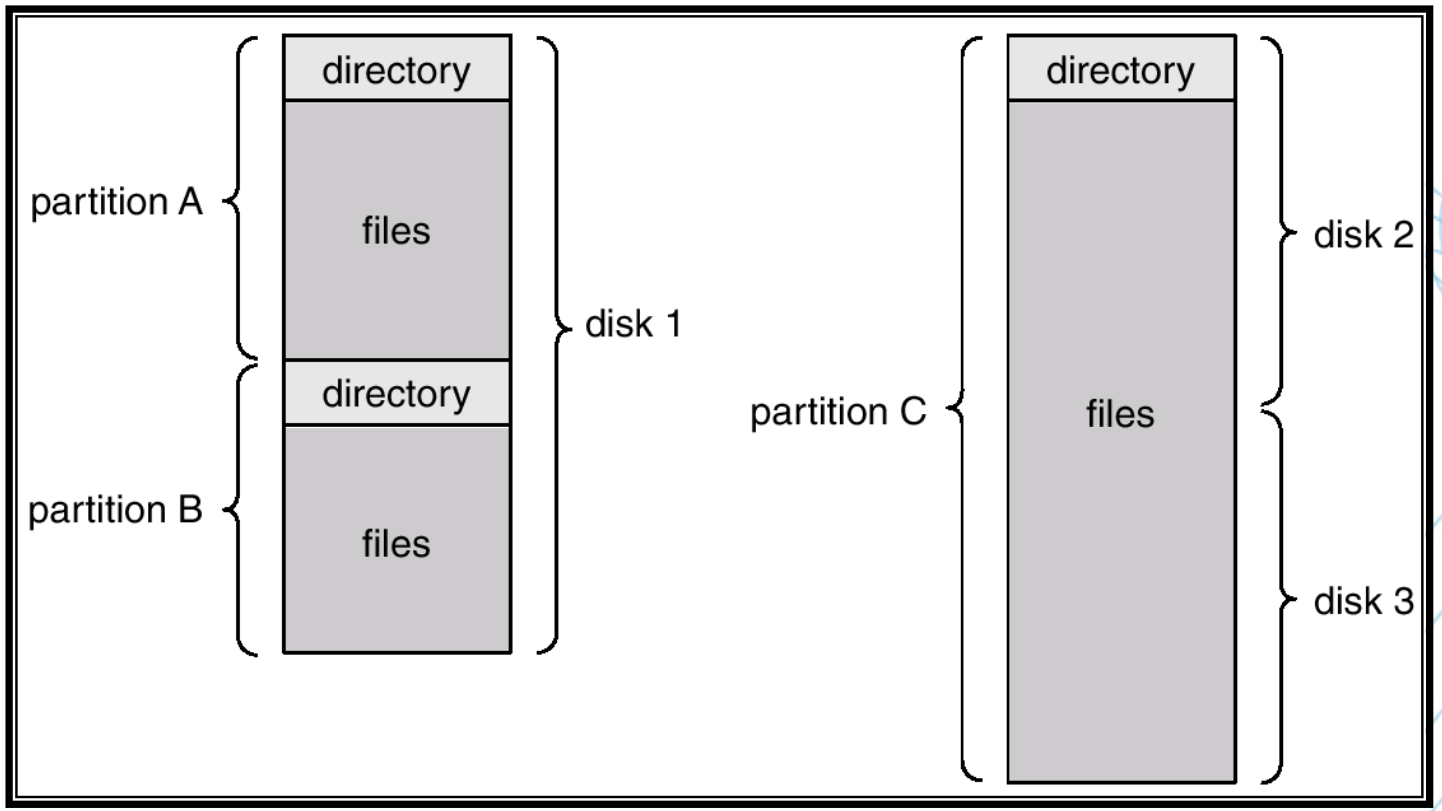
- 效率：要能迅速定位一个文件；
- 命名：要方便用户；
  - 重名：对不同的用户文件，可使用相同名字；
  - 别名：对同一个文件，可有多个不同的名字；
- 分组功能：能够把不同的文件按照某种属性进行分组（如某门课程的所有文件可分为讲义、作业、参考资料等不同的组）。

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

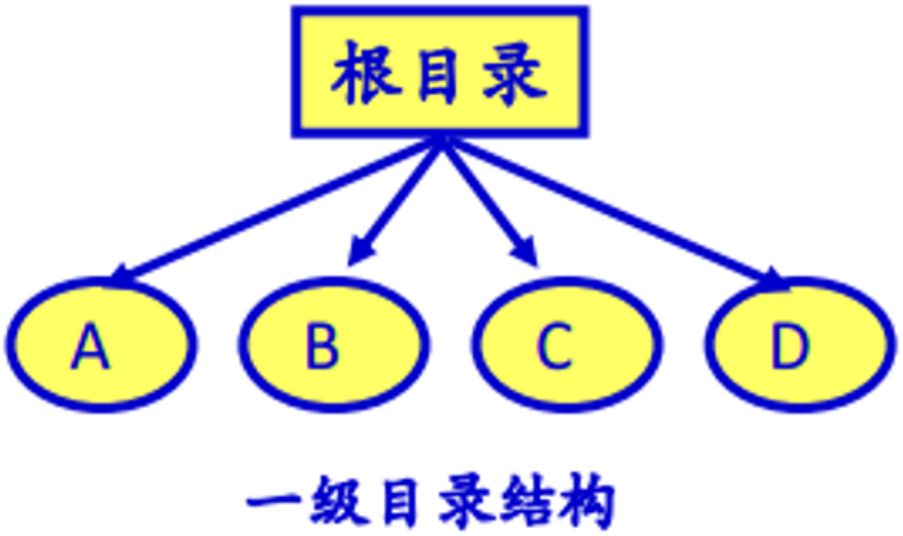
1. Move
2. Copy
3. Rename
4. Link
5. Unlink

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



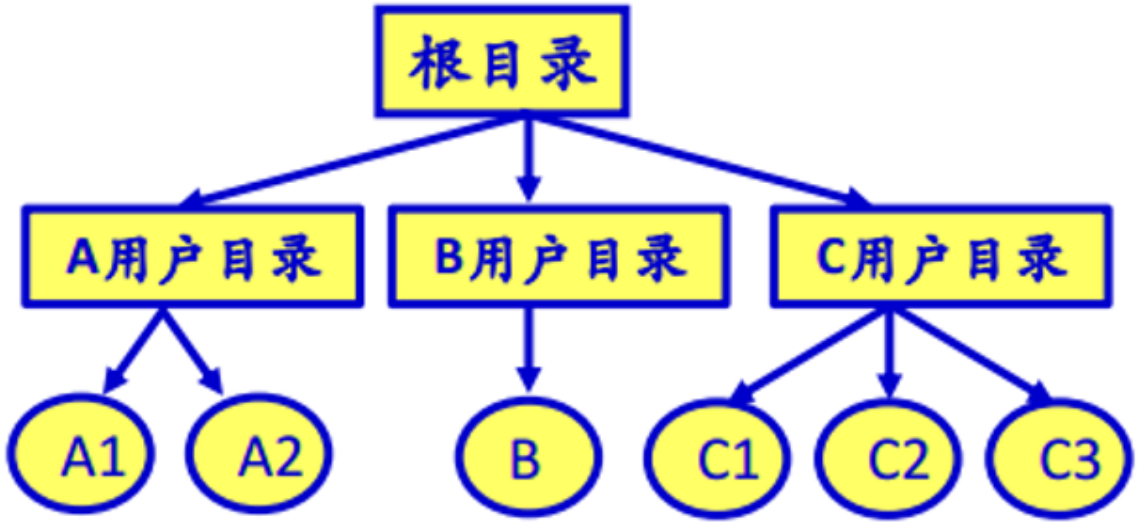
一级目录结构

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

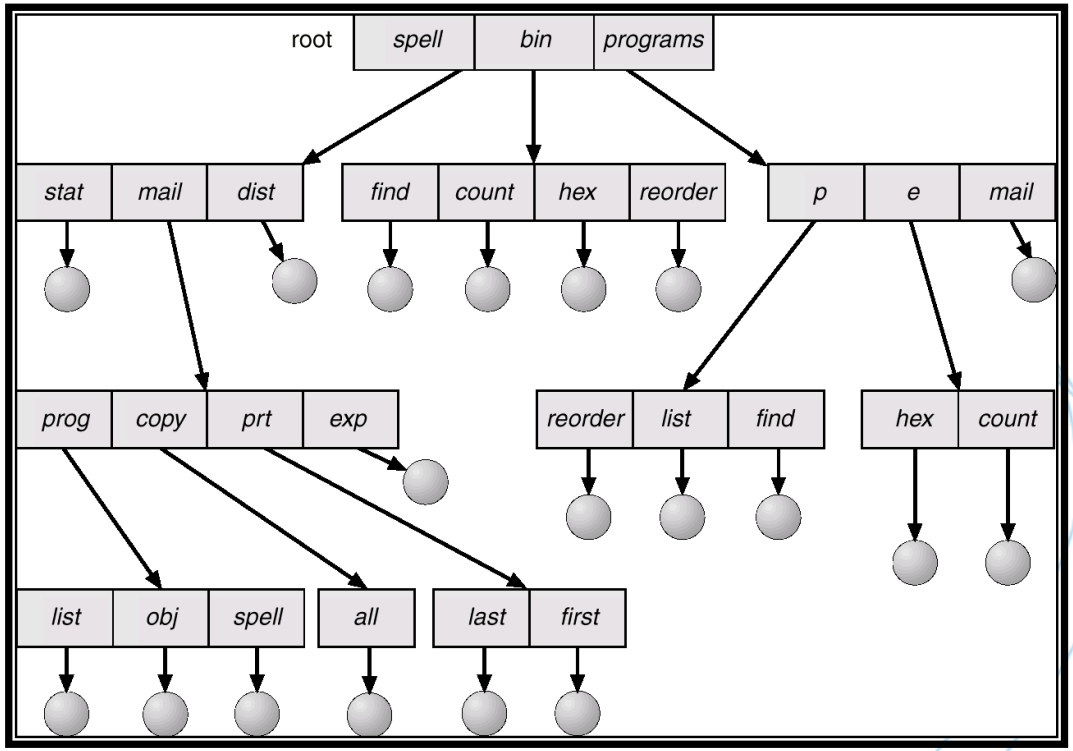
- 在根目录（第一级目录）下，每个用户对应一个目录（第二级目录），在用户目录下是该用户的文件，而不再有下级目录。适用于多用户系统，各用户可有自己的专用目录。



二级目录结构

多级目录（层次目录）

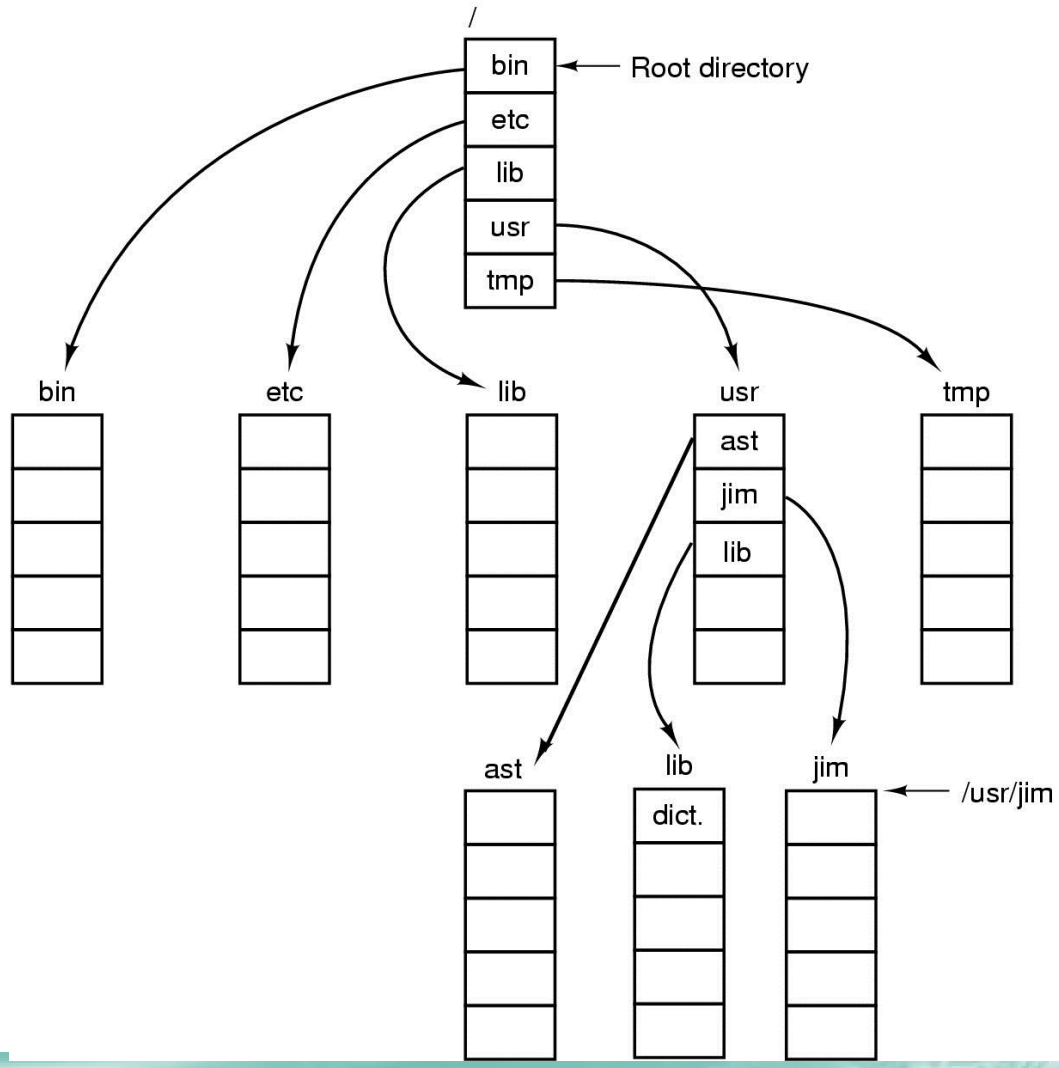
- 在较高的目录级，其目录表目为下一级目录名以及一个指向其目录的指针。
- 在最后一级目录，这个指针指向文件的物理地址。
- 几乎所有现代文件系统都采用这种方案。



多级目录（层次目录）

- 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是透明的机制 ；
- 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、
  - 对文件读何写的管理、
  - 以及对文件的共享何保护的功能。

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

文件系统模型的三个层次

三、对象及其属性

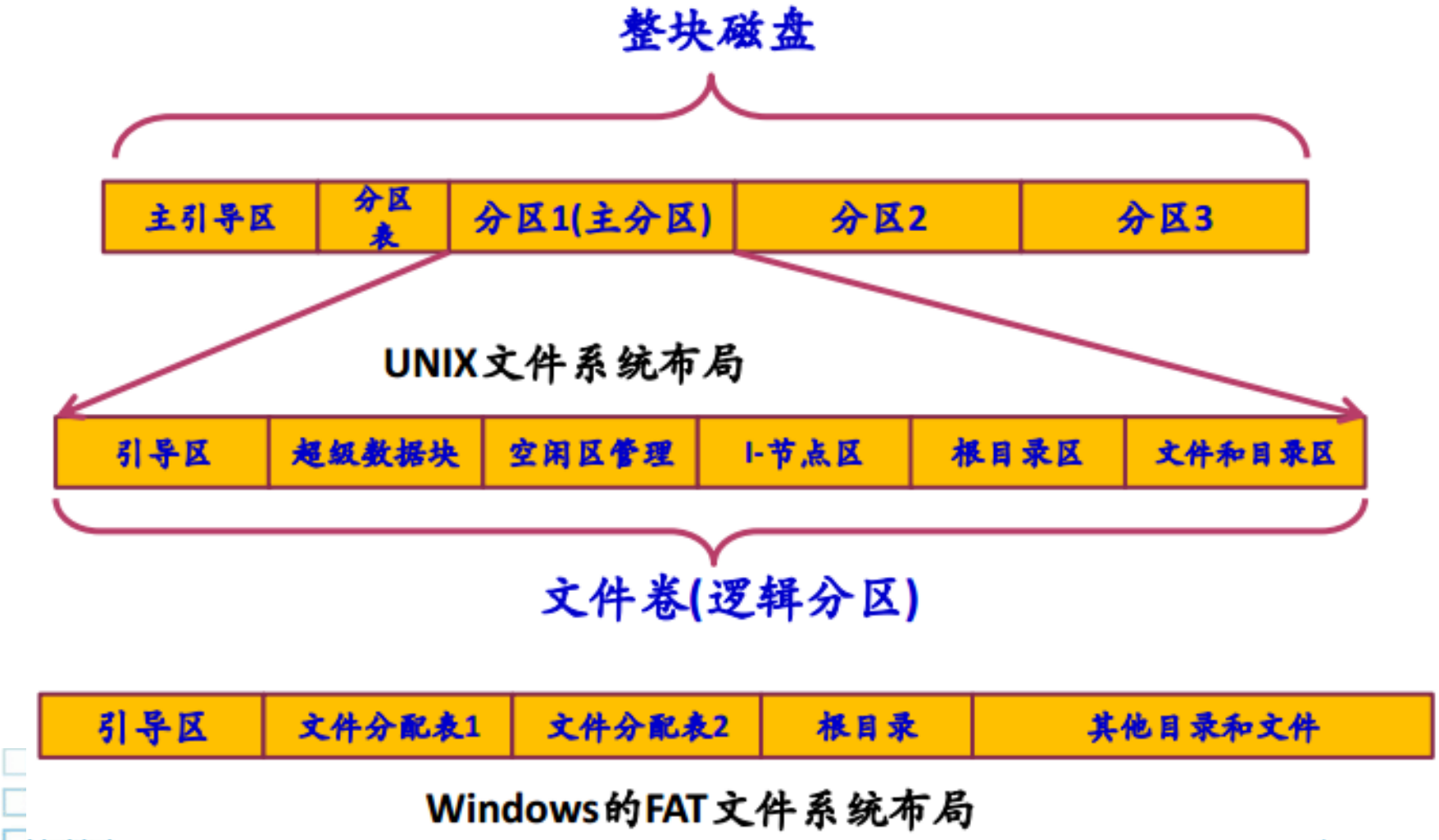
- 文件系统管理的对象有

（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。

磁盘上文件系统的布局



文件的实现需解决以下两个问题

- 【元数据组织】如何描述文件，如何来记录文件的各种管理信息？
- 【数据组织】如何存放文件，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（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。

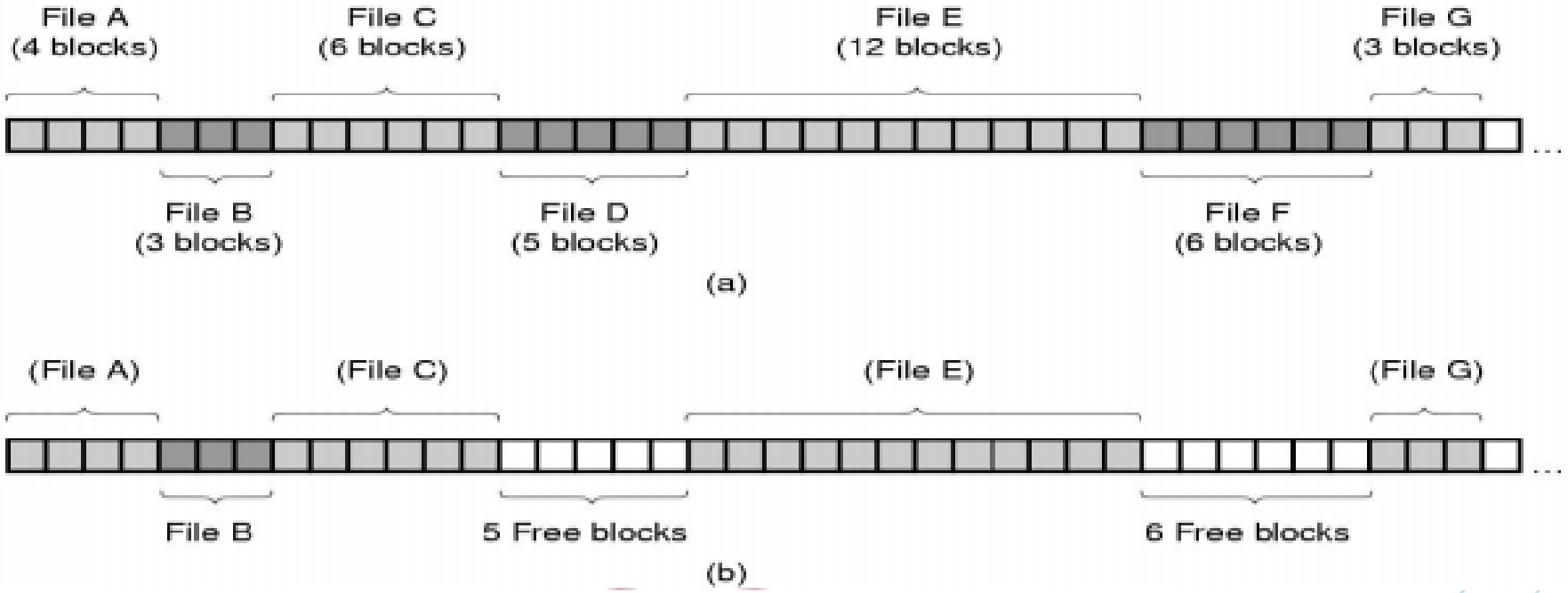
一个典型的文件说明

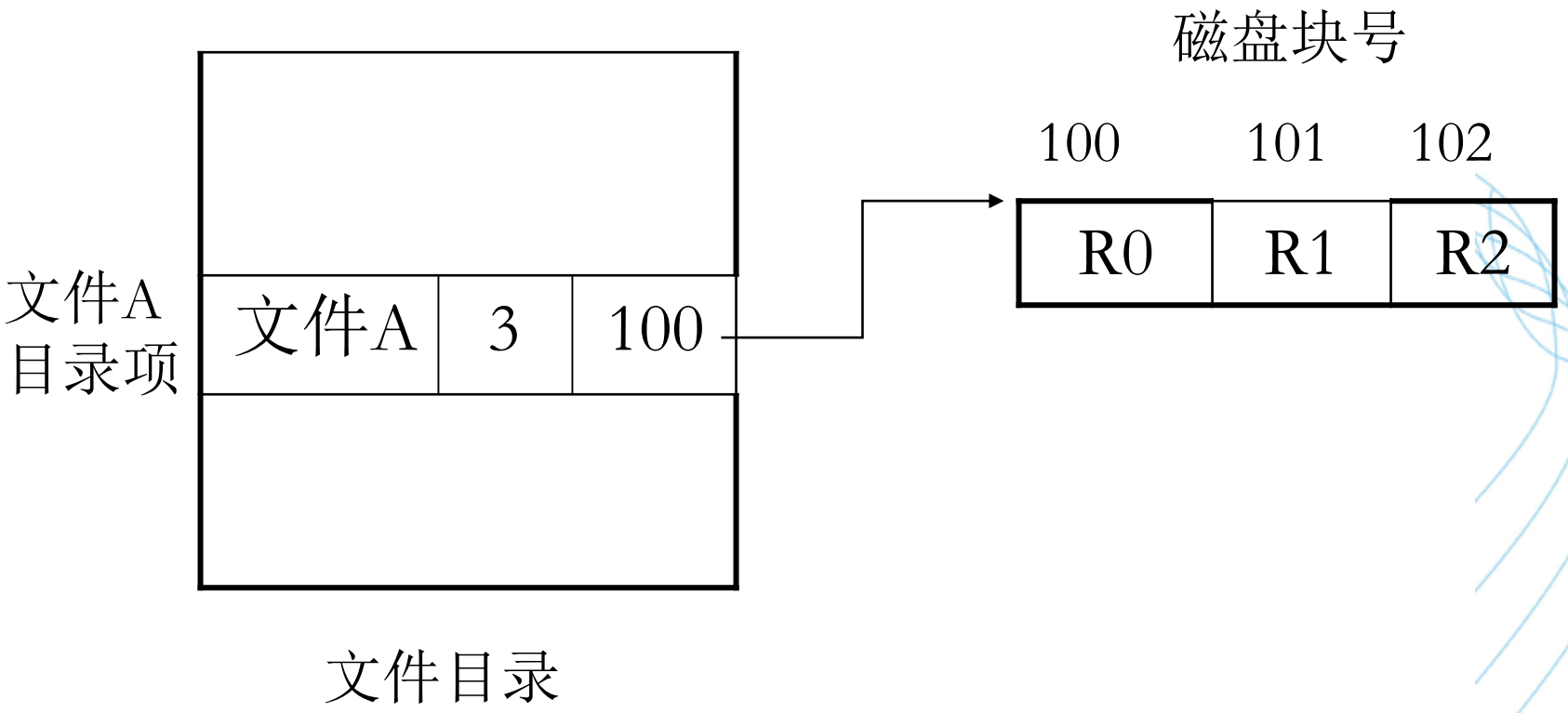
文件名	
文件所在的物理地址	
记录长度	记录个数
文件占有者的存取权限	
其它用户的存取权限	
……	
……	
文件建立时间	
上次存取时间	
暂存文件/永久文件	

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Ri记录时，应如何操作？

连续结构的特点

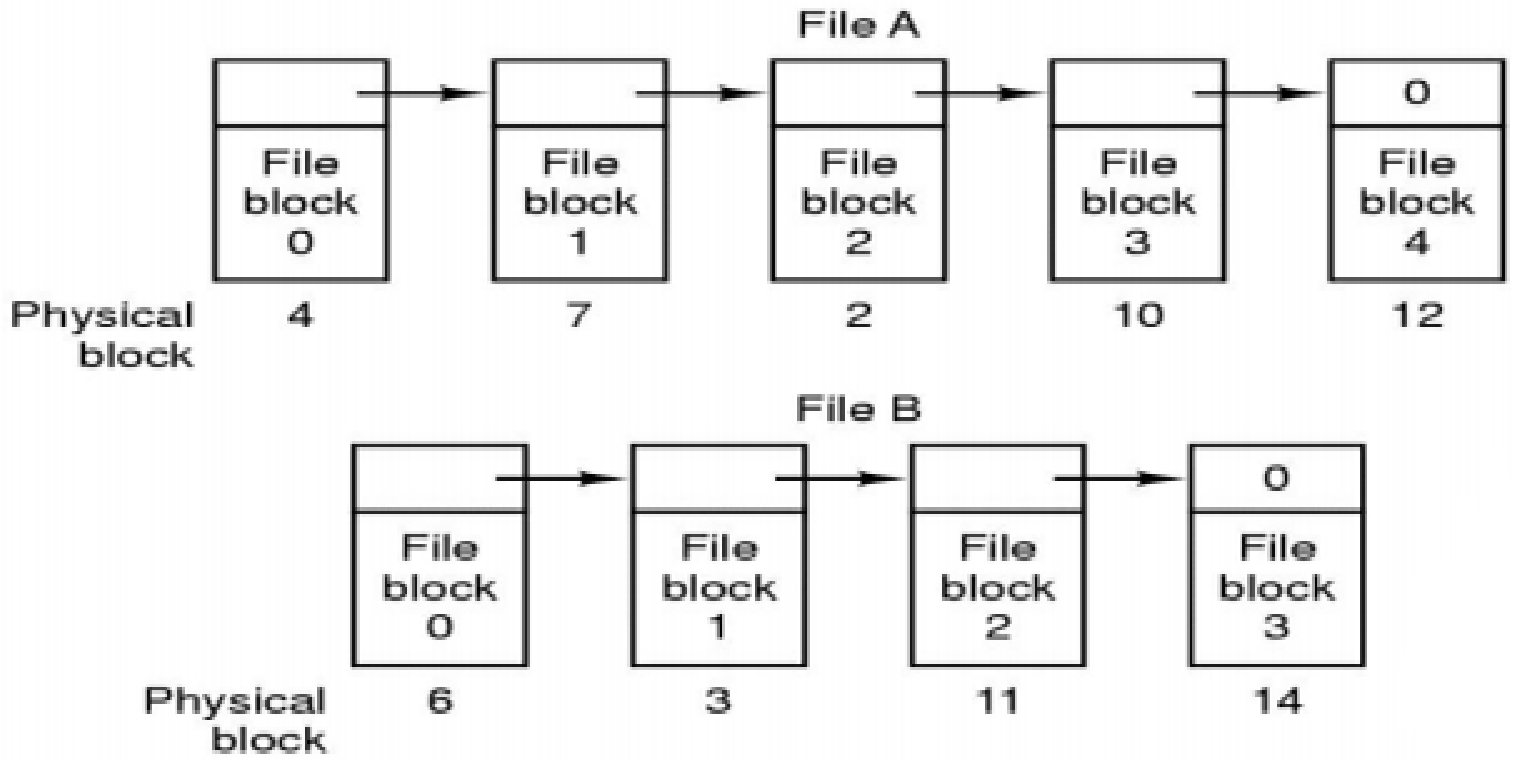
- 优点：
  - 结构简单，实现容易，不需要额外的空间开销
  - 支持顺序存取和随机存取，顺序存取速度快
  - 连续存取时速度较快，
- 缺点：
  - 文件长度一经固定便不易改变；
  - 不利于文件的动态增加和修改；

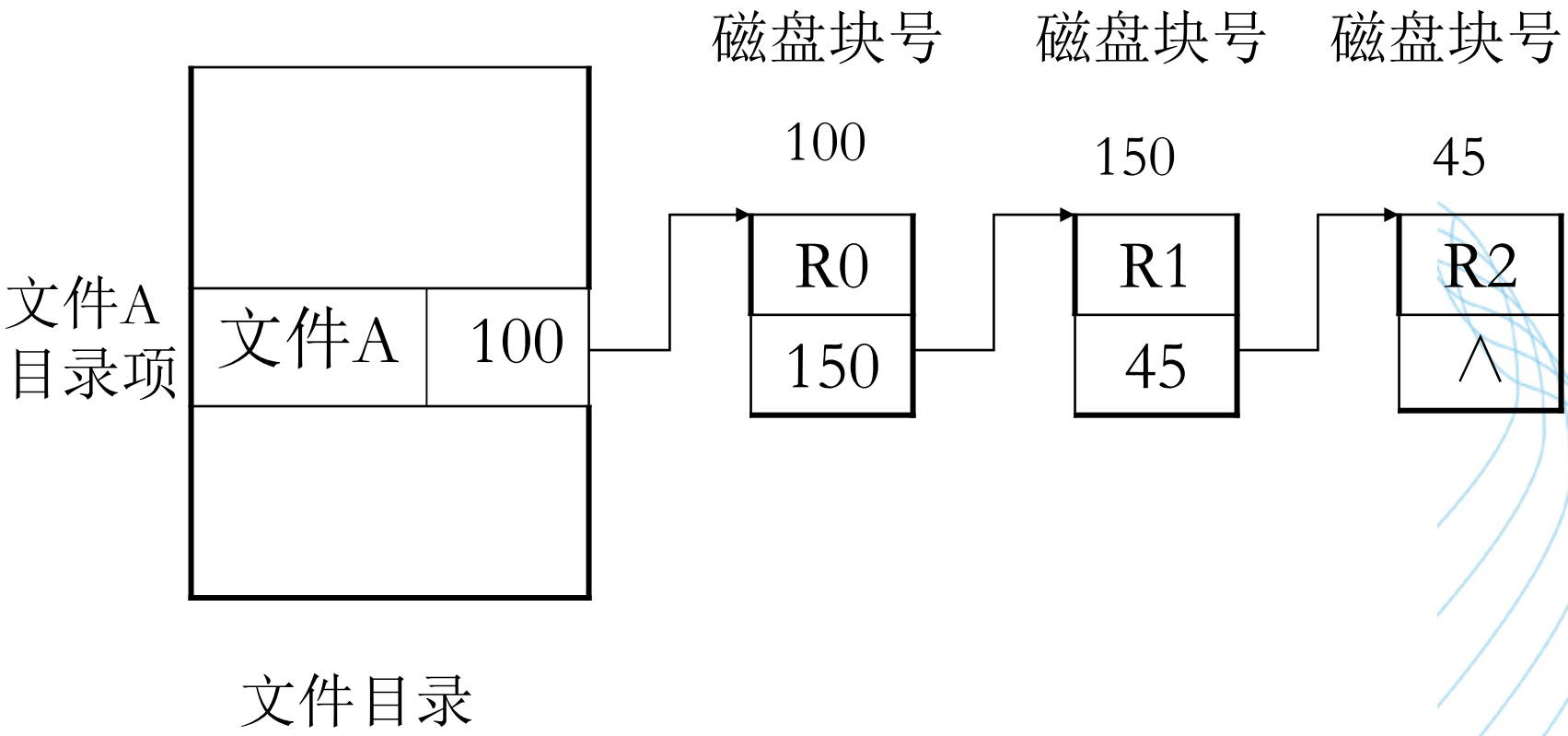
适用于变化不大的顺序访问的文件

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Ri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。

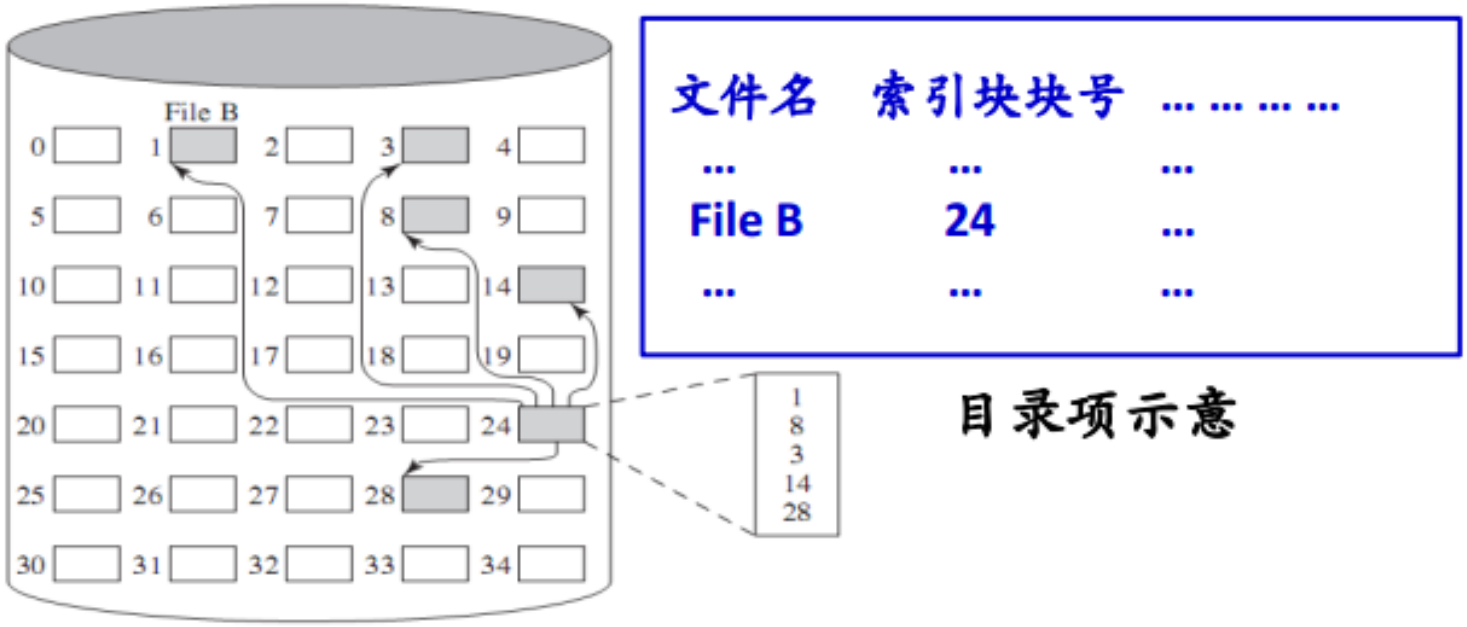
索引结构【重点】

- 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项指向文件的第i块。
- 索引表本身应该存放在何处？
  - 每个文件的索引表长度是不一样的
  - 不能存放在FCB中，FCB中只记录索引表的地址

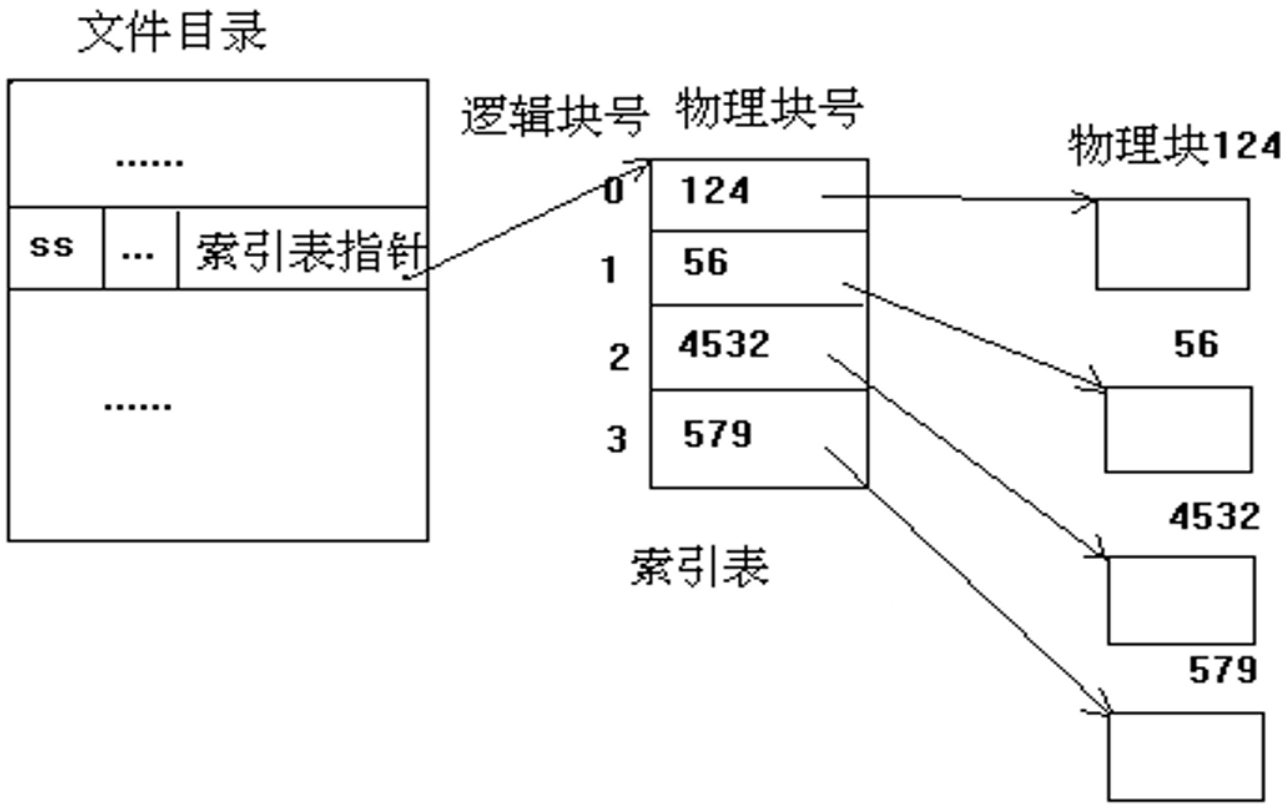
索引文件

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

文件大小

	•••
	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

File1 i 结点 1235

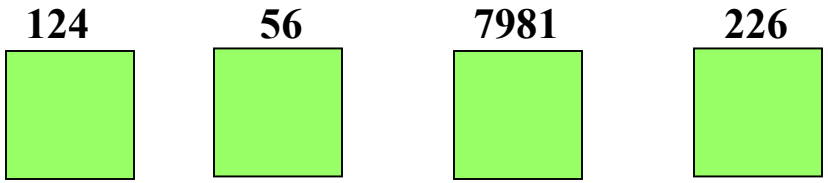
文件目录

Flile 1	1235
•••	

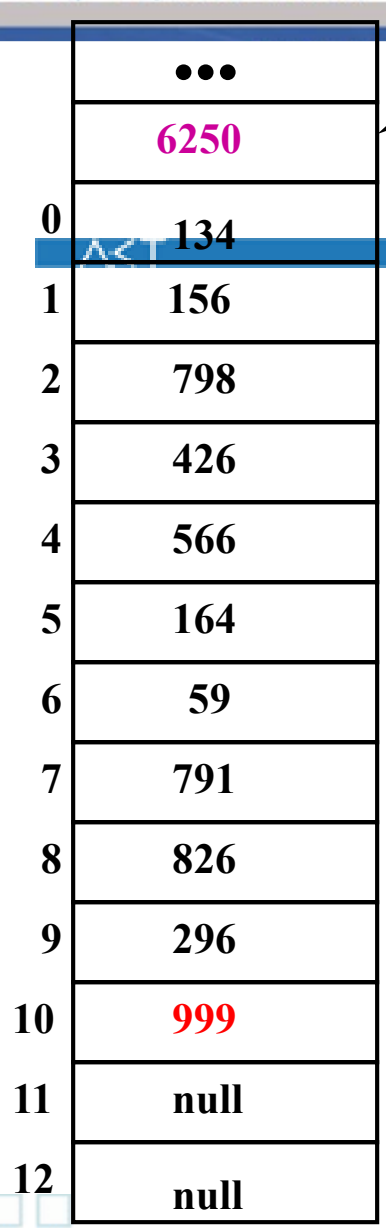
假设：

磁盘块大小：512字节

磁盘块号：4字节

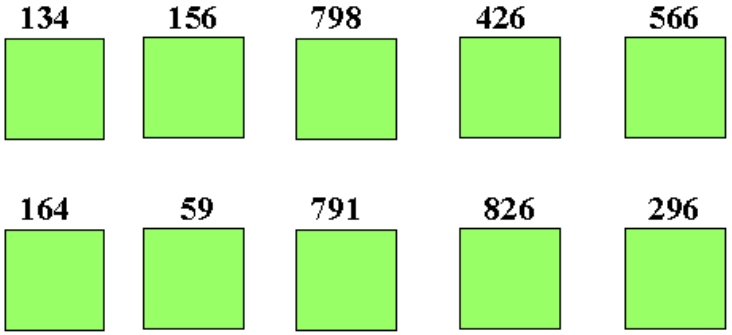


直接索引

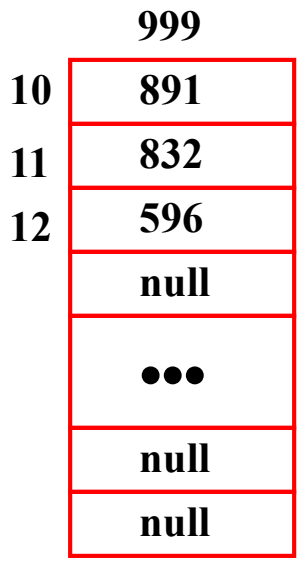


文件目录

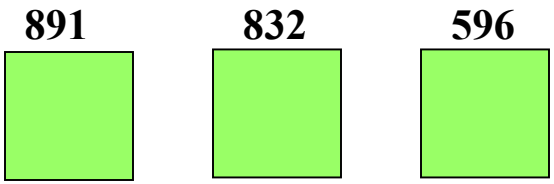
Flile 1	1235
Flile 2	896
•••	



直接索引

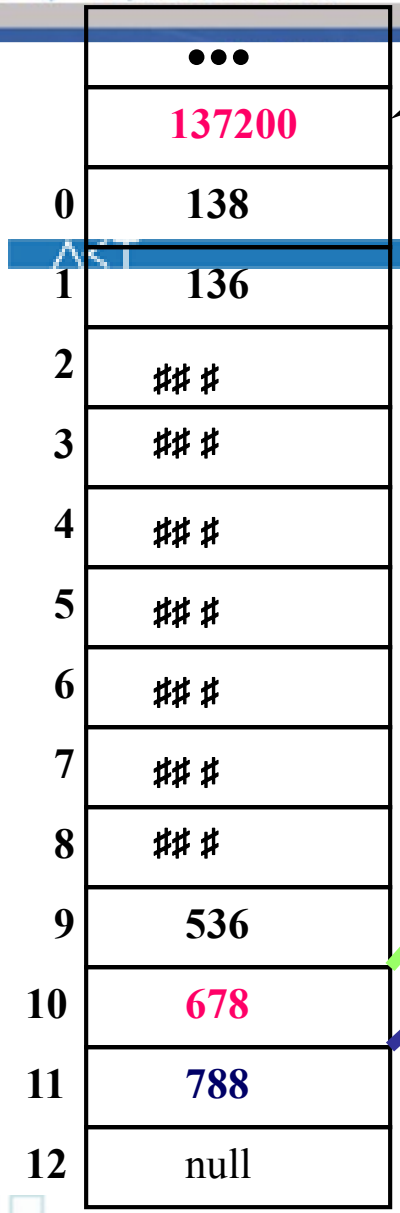


一次索引块



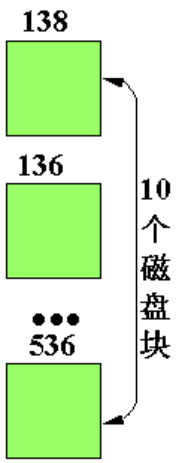
一次间接索引

File2 i 结点 896



文件大小

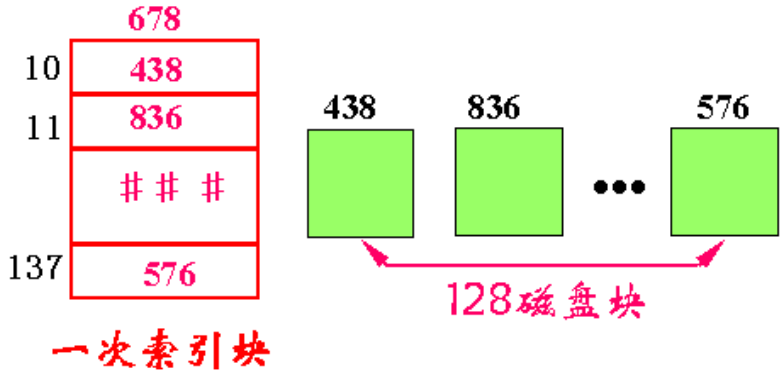
直接索引



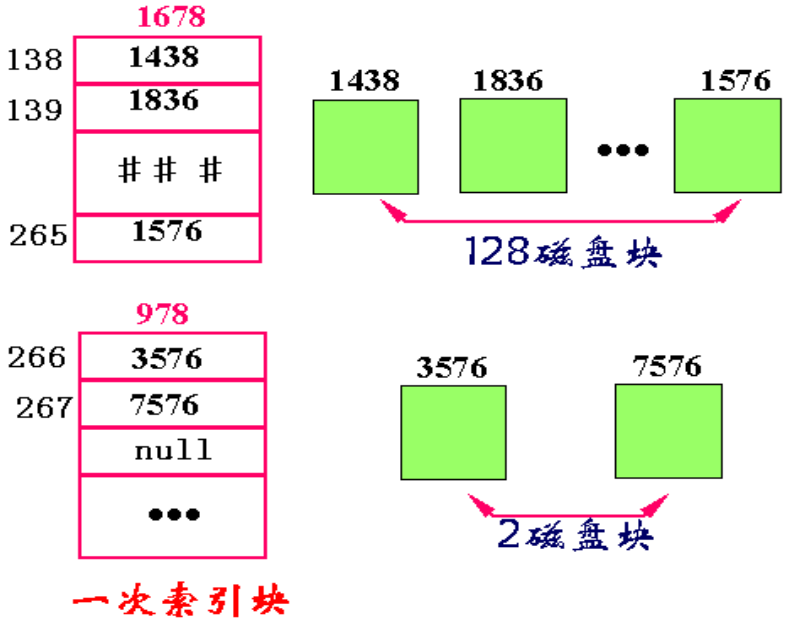
一次间接索引

文件目录

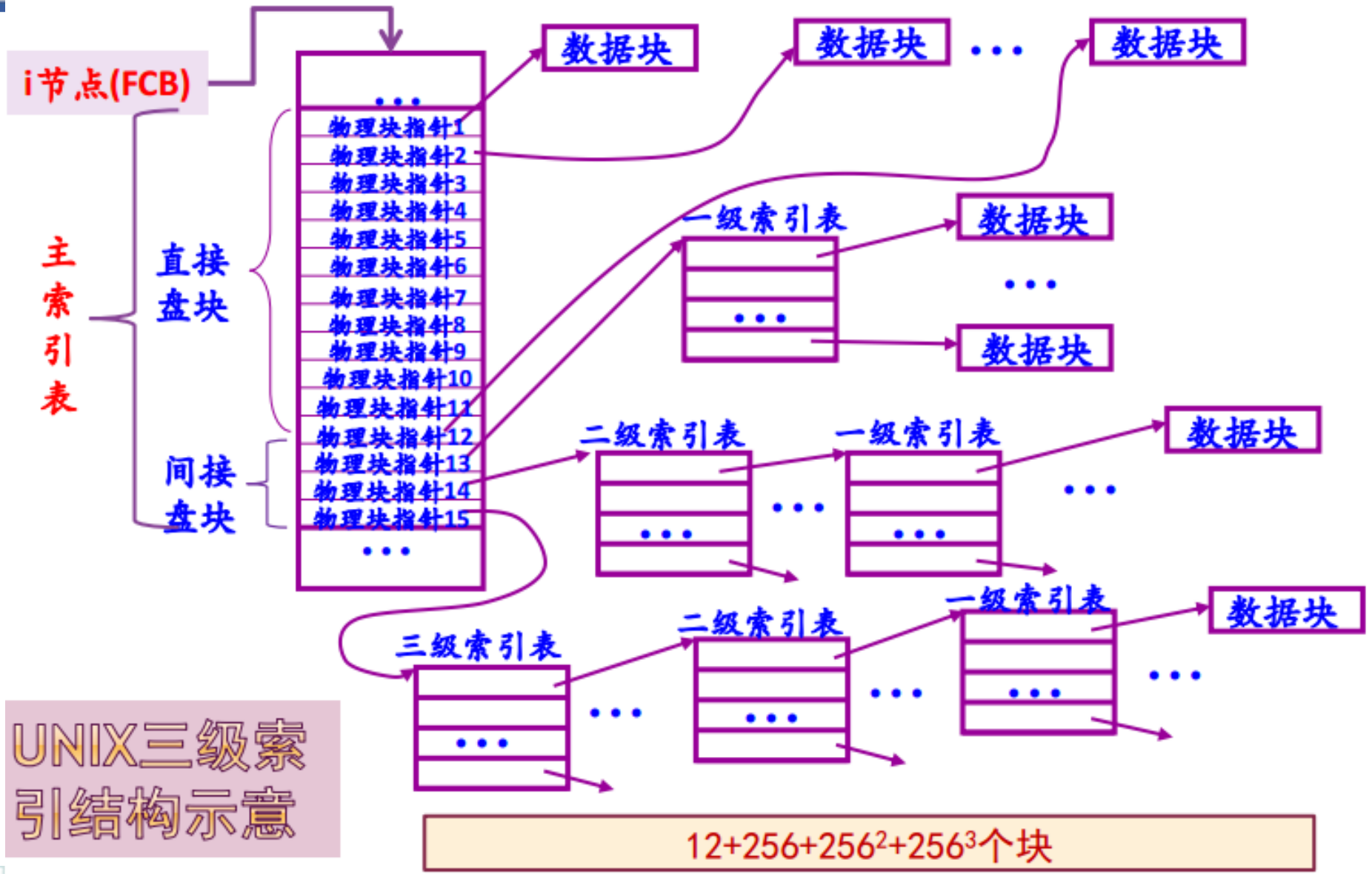
Flile 1	1235
Flile 3	1896



二次间接索引



File3 i 结点 1896



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。

目录的主要功能和实现

目录的主要功能：

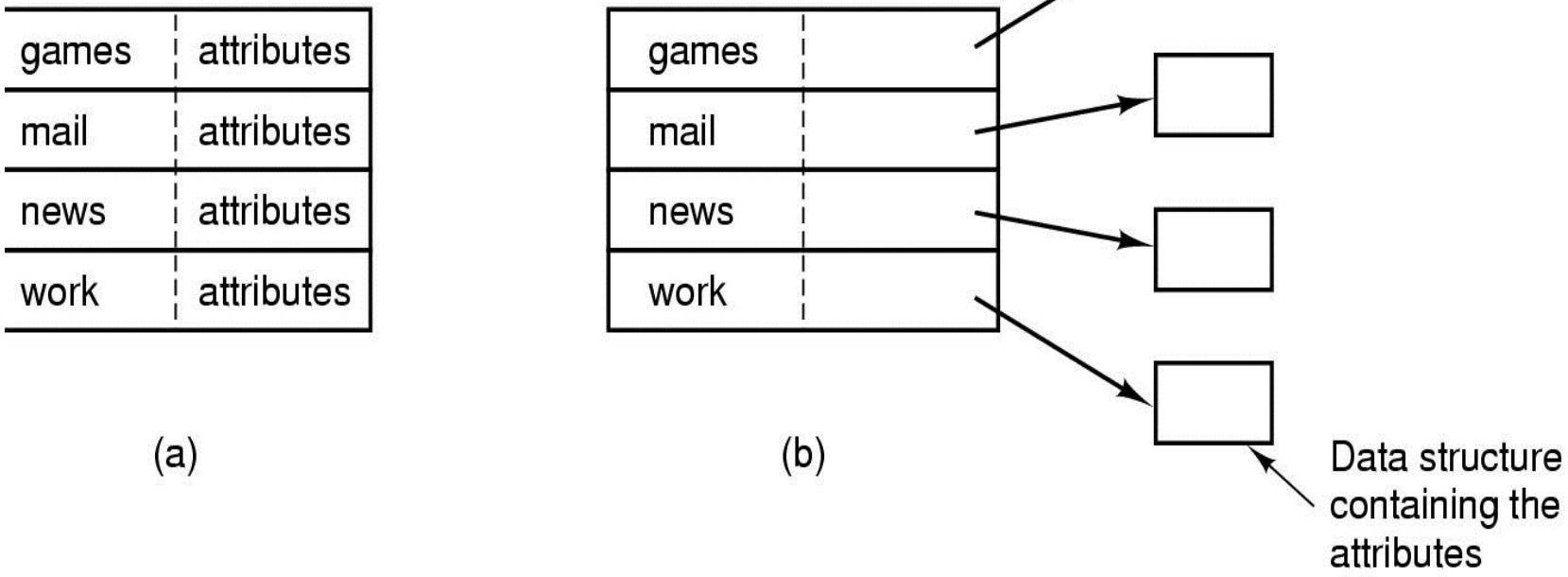
- 根据用户给出的ASCII形式的文件名（路径名），迅速地定位到相应的文件控制块。

  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

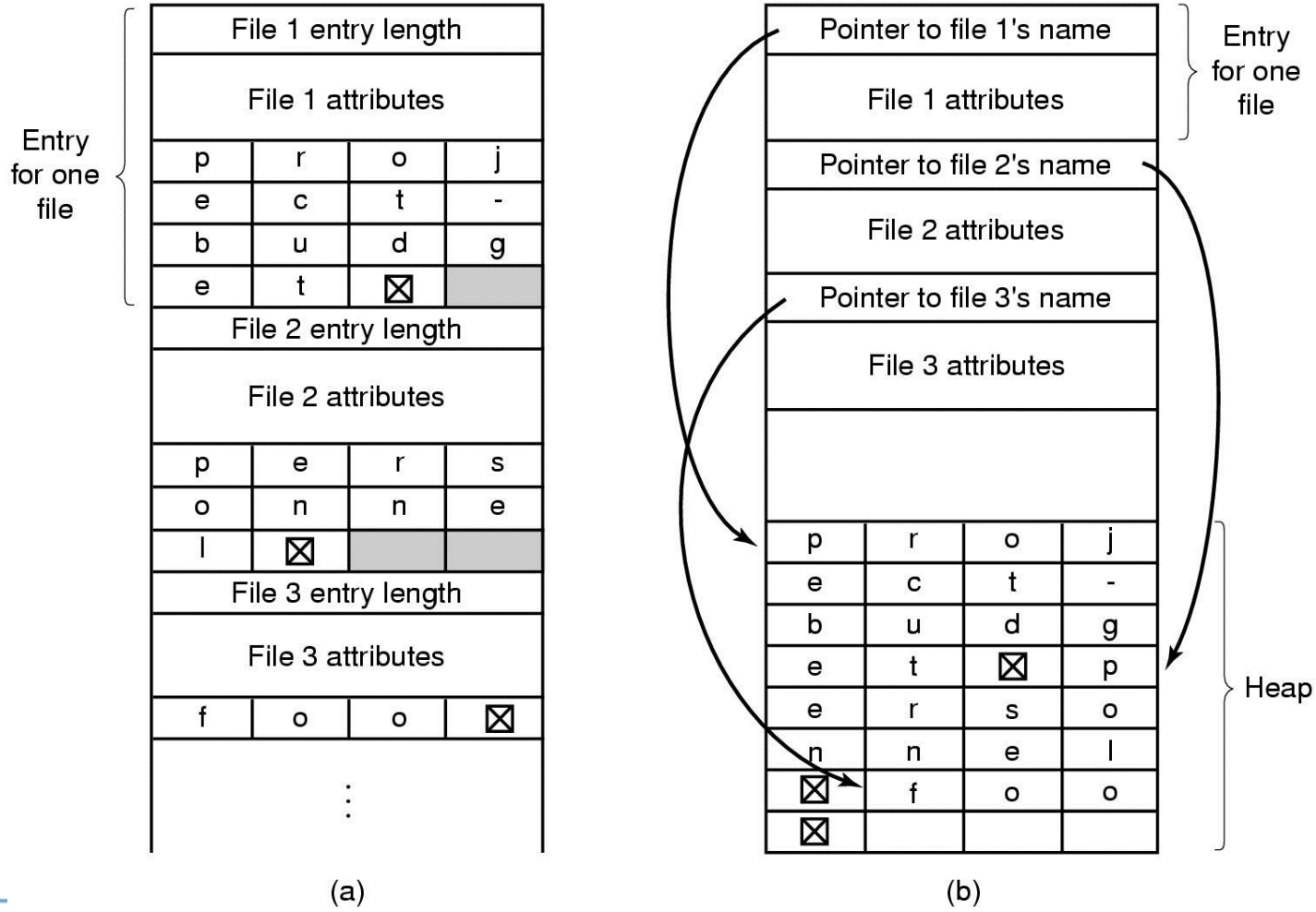


直接法　　间接法

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



方法 2

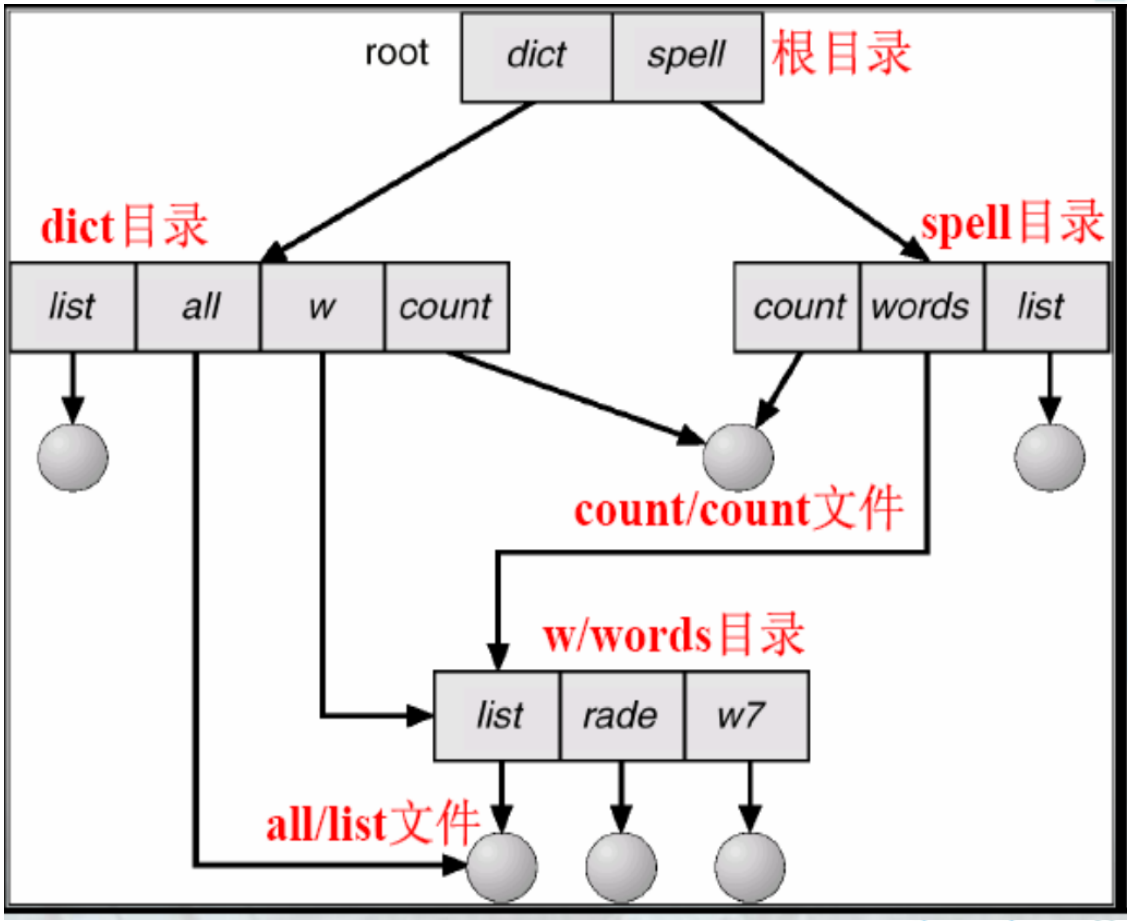
方法 3

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

磁盘空间的管理

- 空闲表法
- 空闲链表法
- 位示图
- 成组链接法

空闲表

序号	第一空闲盘块号	空闲盘块数
1	2	4
2	9	3
3	15	5
4	—	—

空闲链表法

- 将所有的空闲盘区连成一条空闲链，可有两种方式：空闲盘块链、空闲盘区链。
- 空闲盘块链，将所有空闲存储空间，以盘块为基本元素成链。优点：分配和回收盘块的过程非常简单；缺点是空闲盘块链可能很长。
- 空闲盘区链，将磁盘上的所有空闲盘区（每个盘区可包含若干个盘块）拉成一条链。在每个盘区上除了含有用于指示下一个空闲盘区的指针外，还应标有指明本盘区大小（盘块数）的信息。这方法分配和回收过程较复杂，但空闲盘区链较短。

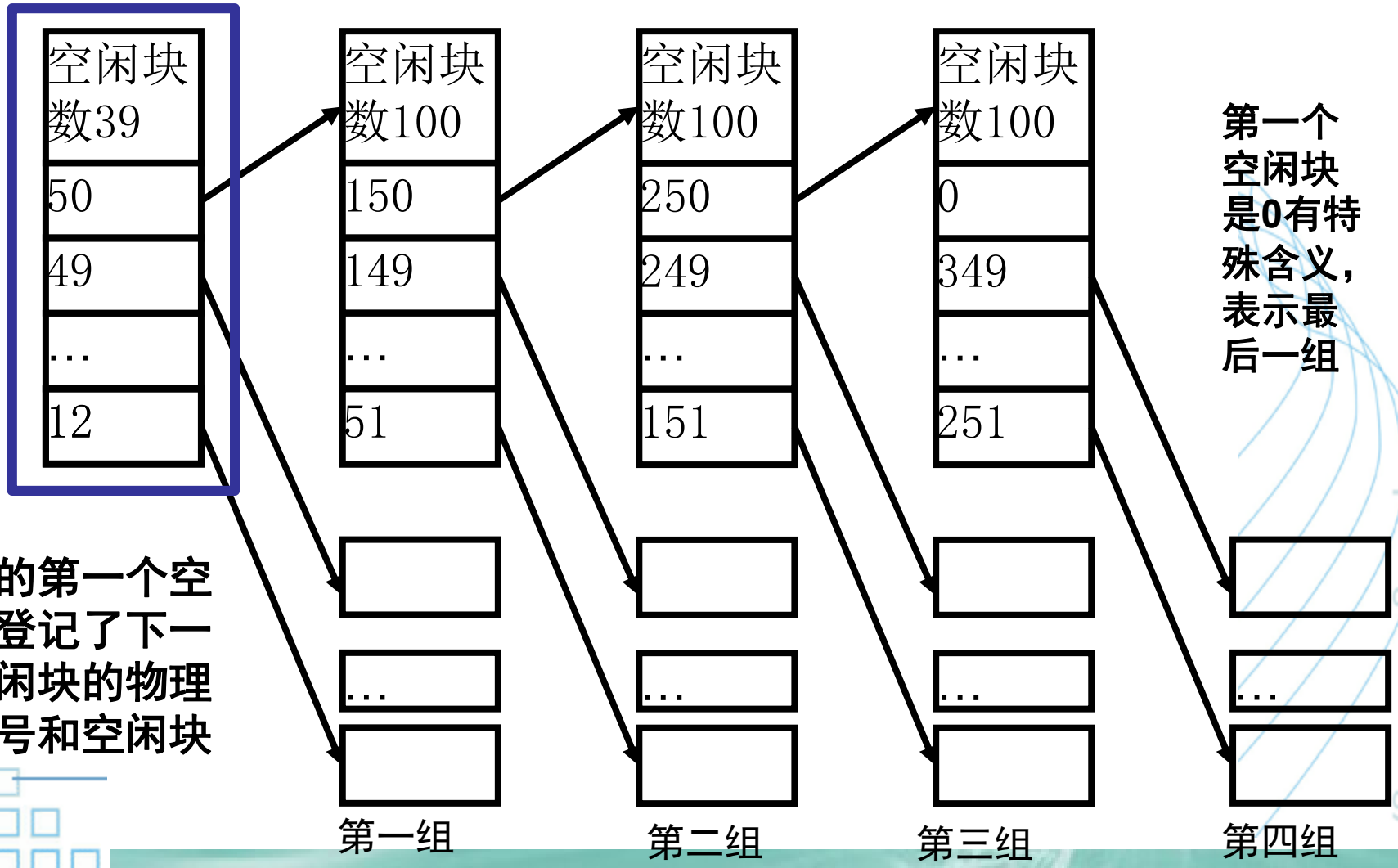
位示图

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	1	1	0	0	0	1	1	1	0	0	1	0	0	1	1	0
2	0	0	0	1	1	1	1	1	1	0	0	0	0	1	1	1
3	1	1	1	0	0	0	1	1	1	1	1	1	0	0	0	0
4																
⋮																
16																

用一串二进制位反映磁盘空间中的分配使用情况，每个物理块对应一位，分配物理块为1，否则为0。

成组链接法

第一个空闲块中的内容



每组的第一个空闲块登记了下一组空闲块的物理盘块号和空闲块总数

第一个空闲块是0有特殊含义，表示最后一组

空白物理块链

- **成组链接法：**

  把空白物理块分成组，在通过指针把组与组之间链接起来，这种管理空白块的方法称为成组链接法。

- **成组链接法的优点：**
  - 空白块号登记不占用额外空间；
  - 节省时间；
  - 采用后进先出的栈结构思想 。

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。

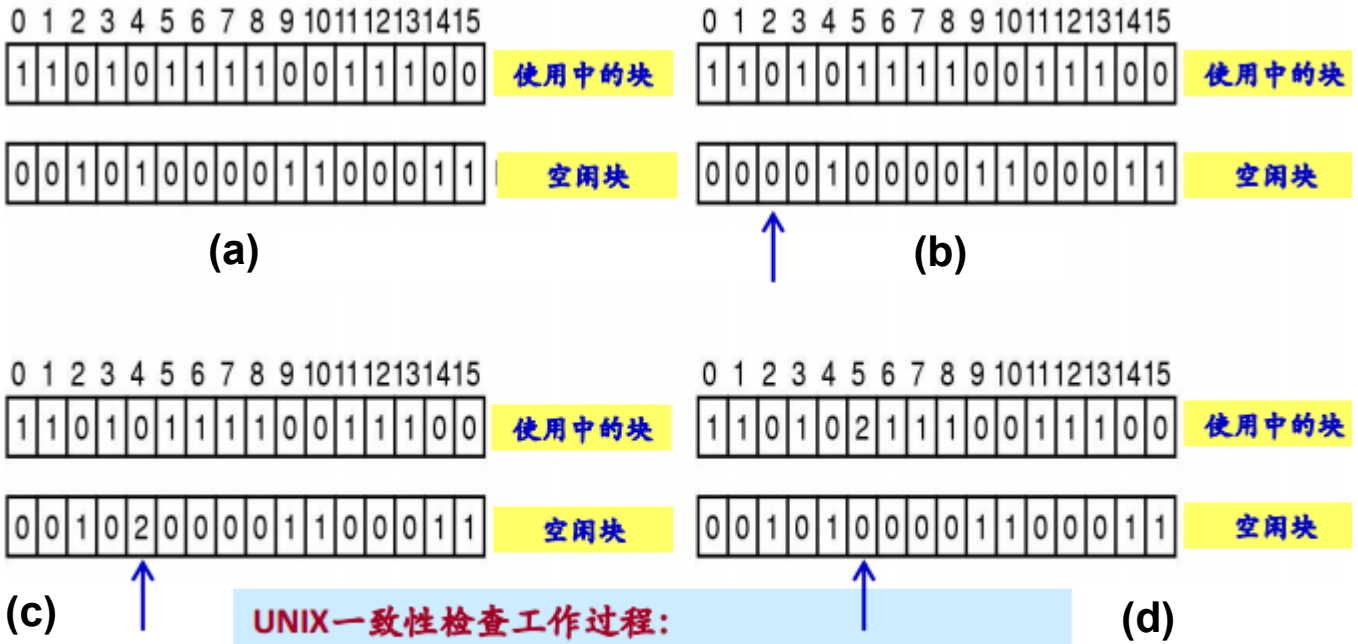
磁盘块不一致的例子

(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



UNIX一致性检查工作过程:

两张表，每块对应一个表中的计数器，初值为0

表一：记录了每块在文件中出现的次数

表二：记录了每块在空闲块表中出现的次数

文件的存取控制

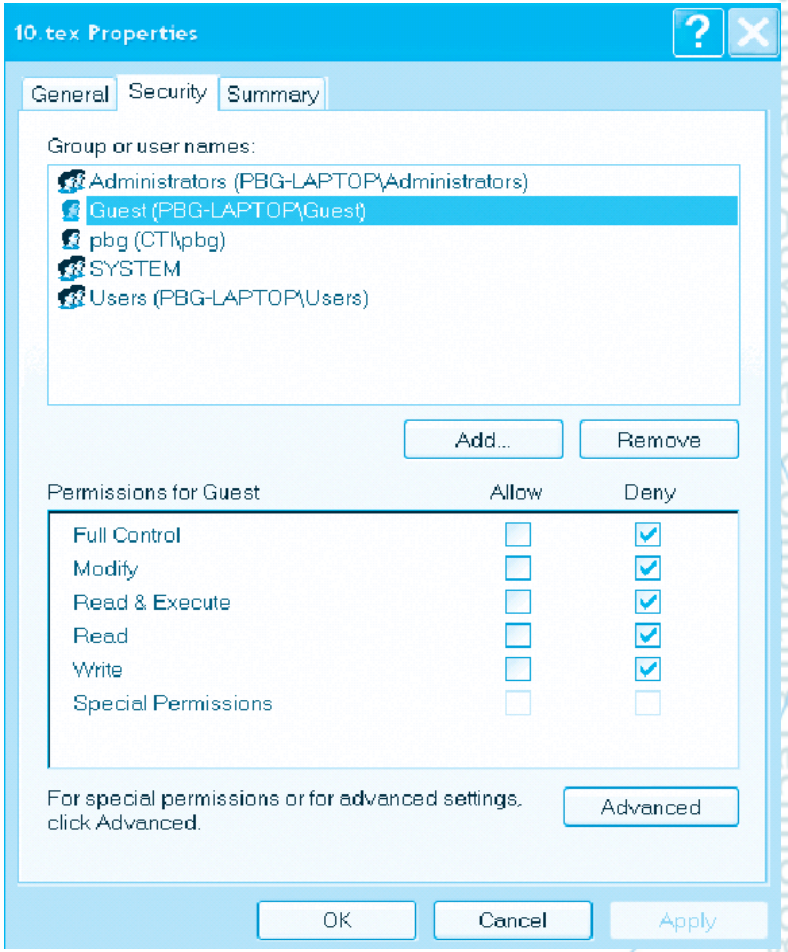
- 文件保护机制
  - 防止未被核准的用户存取文件；
  - 防止一个用户冒充另一个用户来存取；
  - 防止核准用户（包括文件主）误用文件；
- 存取权限验证步骤
  - 审定用户的权限；
  - 比较用户的权限与本次存取要求是否一致；
  - 将存取要求和被访问文件的保密性比较，看是否有冲突。

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

RWX
a) owner access 7⇒1 1 1
RWX
b) **group access** 6 ⇒1 1 0
RWX
c) **public access** 1 ⇒0 0 1



小练习

```
woty@ubuntu:~/oslab$ ls -l
总用量 44
drwxrwxr-x 2 woty woty 4096  4? 29 09:33 boot
drwxrwxr-x 3 woty woty 4096  3? 17 08:38 drivers
drwxrwxr-x 2 woty woty 4096  5? 17 01:12 gxemul
drwxrwxr-x 4 woty woty 4096  5?  9 11:05 include
-rwxrwxr-x 1 woty woty  278  4? 28 23:04 include.mk
drwxrwxr-x 2 woty woty 4096  5?  9 11:11 init
drwxrwxr-x 2 woty woty 4096  4? 28 23:37 lib
-rwxrwxr-x 1 woty woty  931  4? 28 23:04 Makefile
drwxrwxr-x 2 woty woty 4096  5?  9 11:55 mm
-rw-rw-r-- 1 woty woty   18  4? 28 11:19 readme
drwxrwxr-x 2 woty woty 4096  5? 17 01:12 tools
```

哪些用户可以读、写readme文件？
执行chmod 777 readme命令以后呢？
执行什么命令可以恢复上面的状态？

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

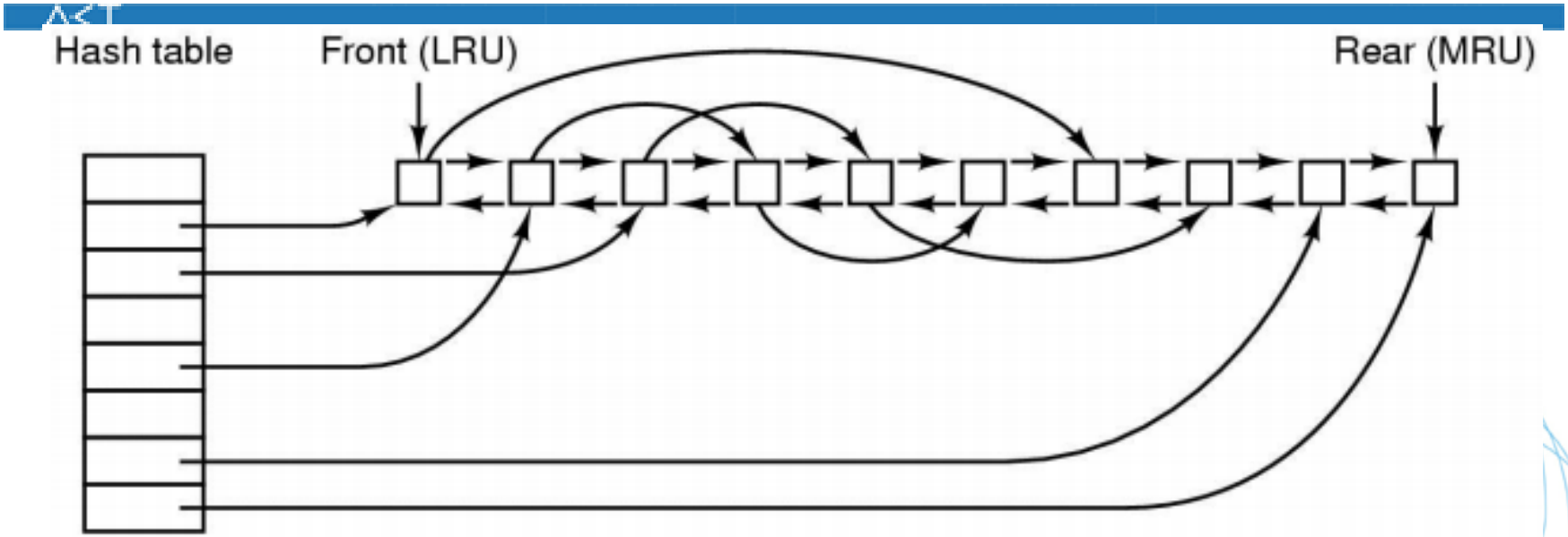
文件系统的性能问题

- 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冲中
  - 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，很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



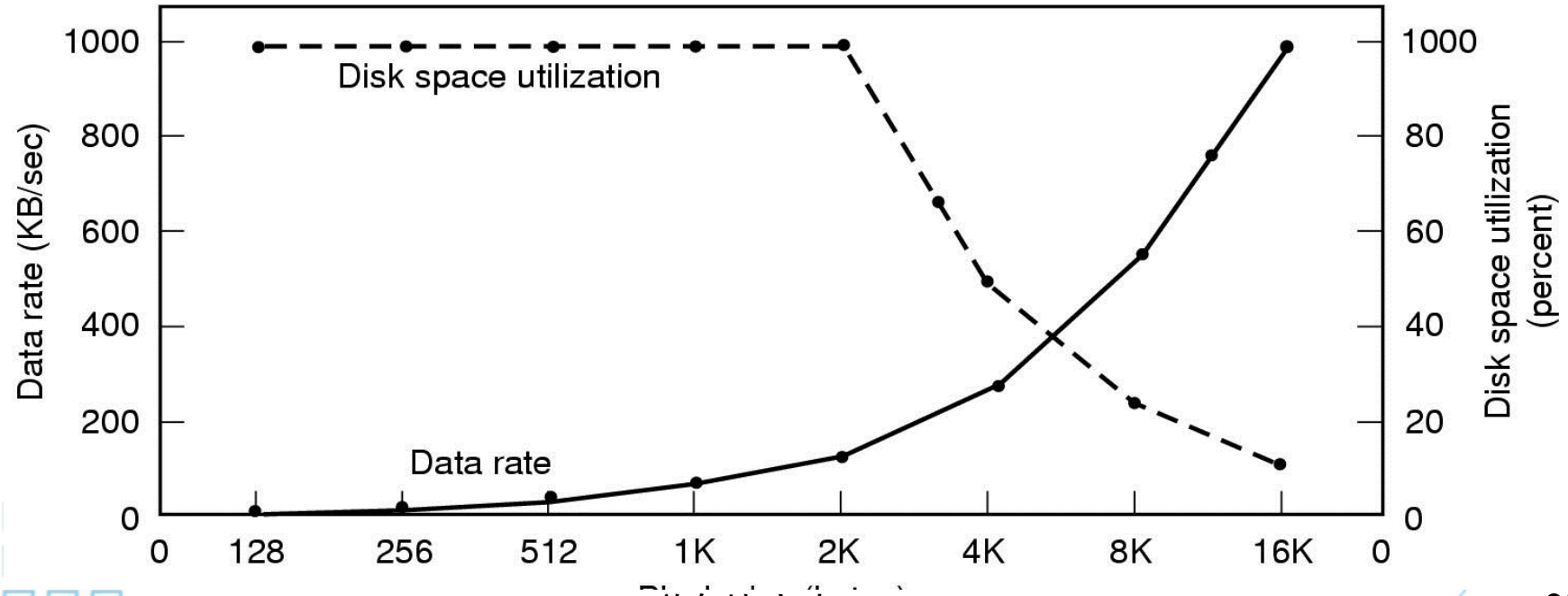
- 在块高速缓存中有若干个数据块，首先将这些块使用一个双向链表组织起来，当要访问这个链的时候就将其从此链中拿出来，然后挂接到链尾，而我们对于某个文件使用的块要检查其是否在高速缓存中，所以这里又使用块号进行散列以提高检查速度。

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）。

磁盘块的大小

- 块过大：读取磁盘次数少，但浪费空间
- 块过小：磁盘空间利用率高，但因多次读取磁盘而浪费时间



文件系统总结

- 文件系统基本概念
- 抽象
  - 文件系统：存储块（字节）——→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

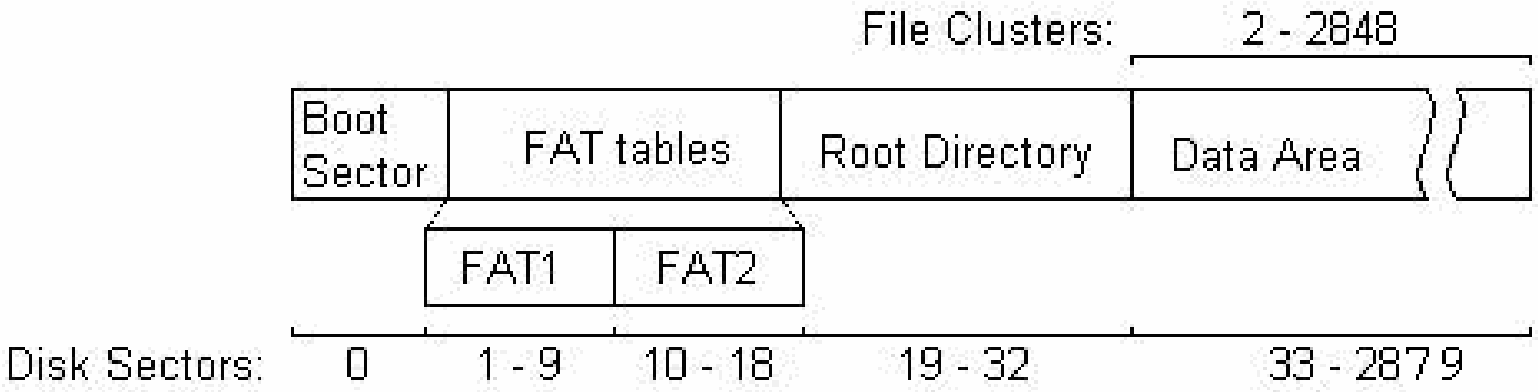
内容提要

- 文件系统基本概念
- 文件系统实现方法
- 文件系统实例分析
  - FAT
  - EXT2
  - Linux VFS
  - LFS

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）

FAT文件系统：分区引导扇区DBR

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）

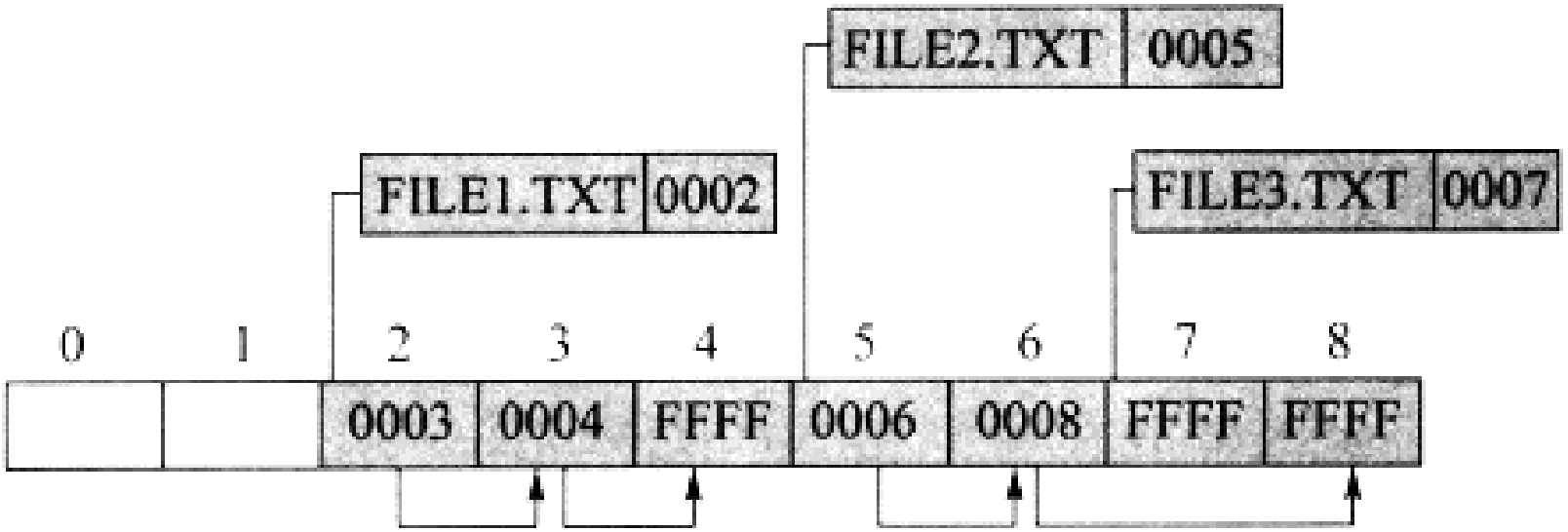
字节偏移量(16进制)	域长	含义
24	4字节	每个文件分配表的扇区数(FAT32用)
28	2字节	标记：bit7为1，表示只有一个FAT；否则，两个FAT互为镜像
2A	2字节	版本号
2C	2字节	根目录起始簇号，通常为2
30	2字节	FSINFO所在扇区，通常1扇区
32	2字节	引导扇区备份，通常是6号扇区
34	12字节	未用
40	1字节	BIOS Int 13H设备号
41	1字节	未用
42	1字节	扩展引导标识，如果后面三个值有效，设为0x29
43	4字节	卷序列号：当格式化卷时创建的一个唯一的数字
47	11字节	卷标(ASCII码)，建立文件系统时由用户指定
52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

- 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：物理坏簇
  - 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：
  簇编号位数/（8*512*每个簇的扇区数）

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。



目录

- 目录项的顺序文件，不对目录项排序。若目录中包含的文件数目较多，则搜索效率低。每个目录项大小为32字节，其内容包括：文件名（8+3个字符），属性（包括文件、子目录和文件卷标识），最后一次修改时间和日期，文件长度，第一个簇的编号。
- 在目录项中，若第一个字节为 E5h，则表示空目录项；若为 05h，则表示文件名的第一个字符为 E5h。
- 文件名不区分大小写

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

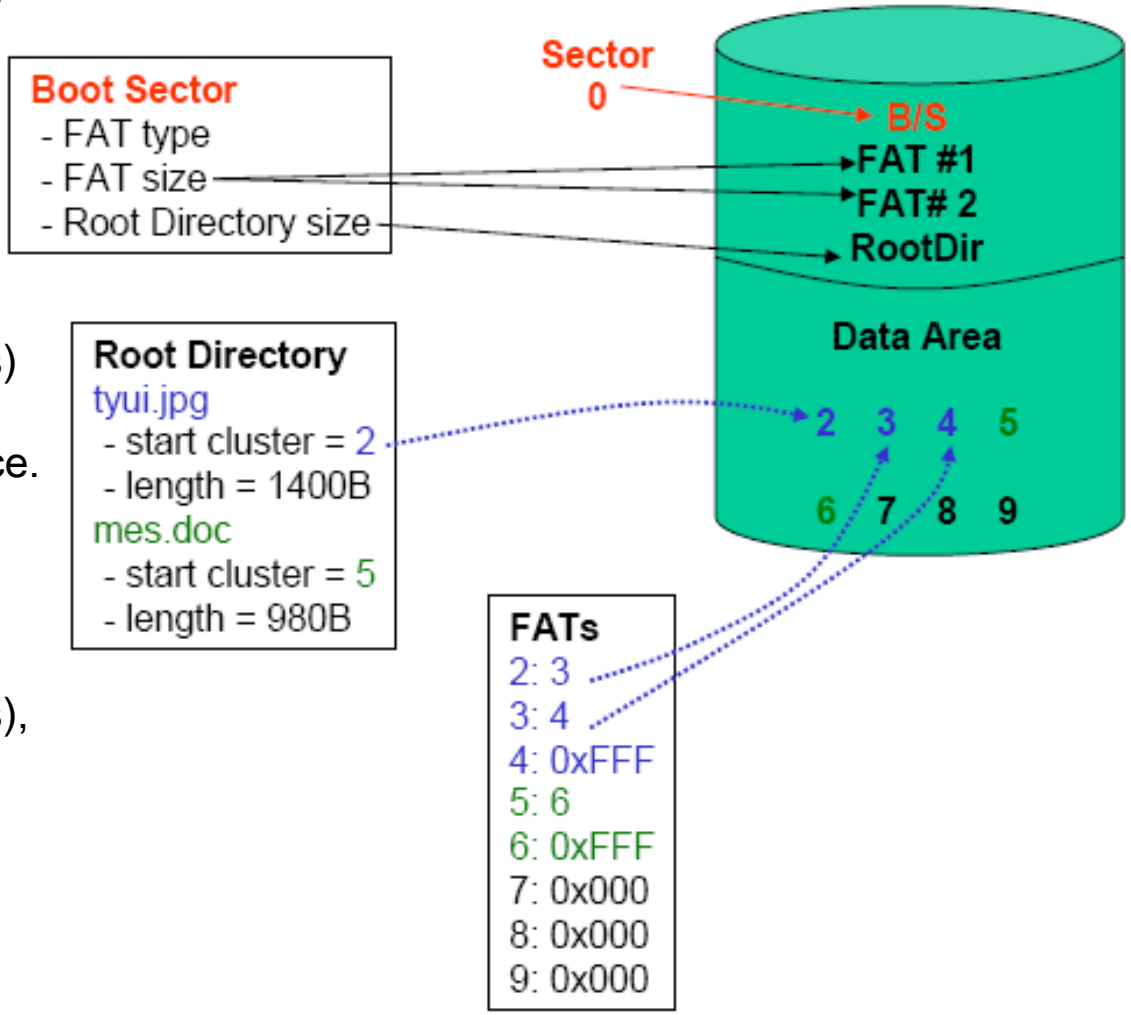
FAT32目录项

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.



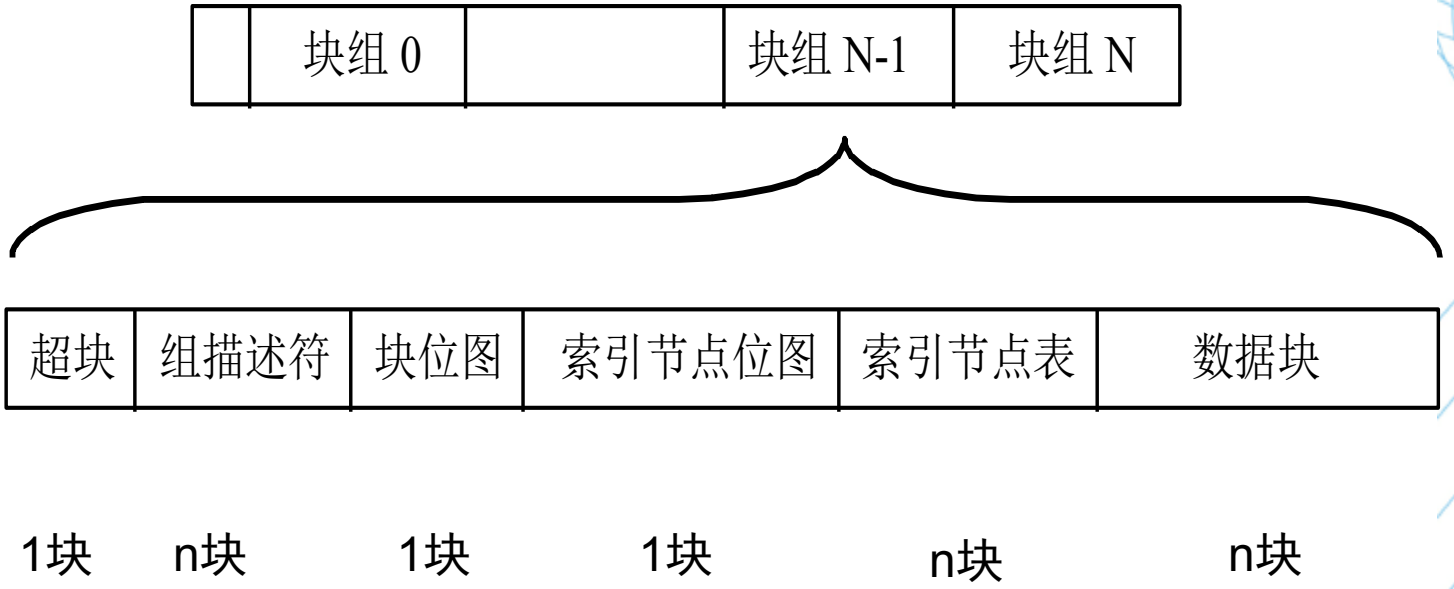
http://www.c-jump.com/CIS24/Slides/FAT/lecture.html

ext2文件系统

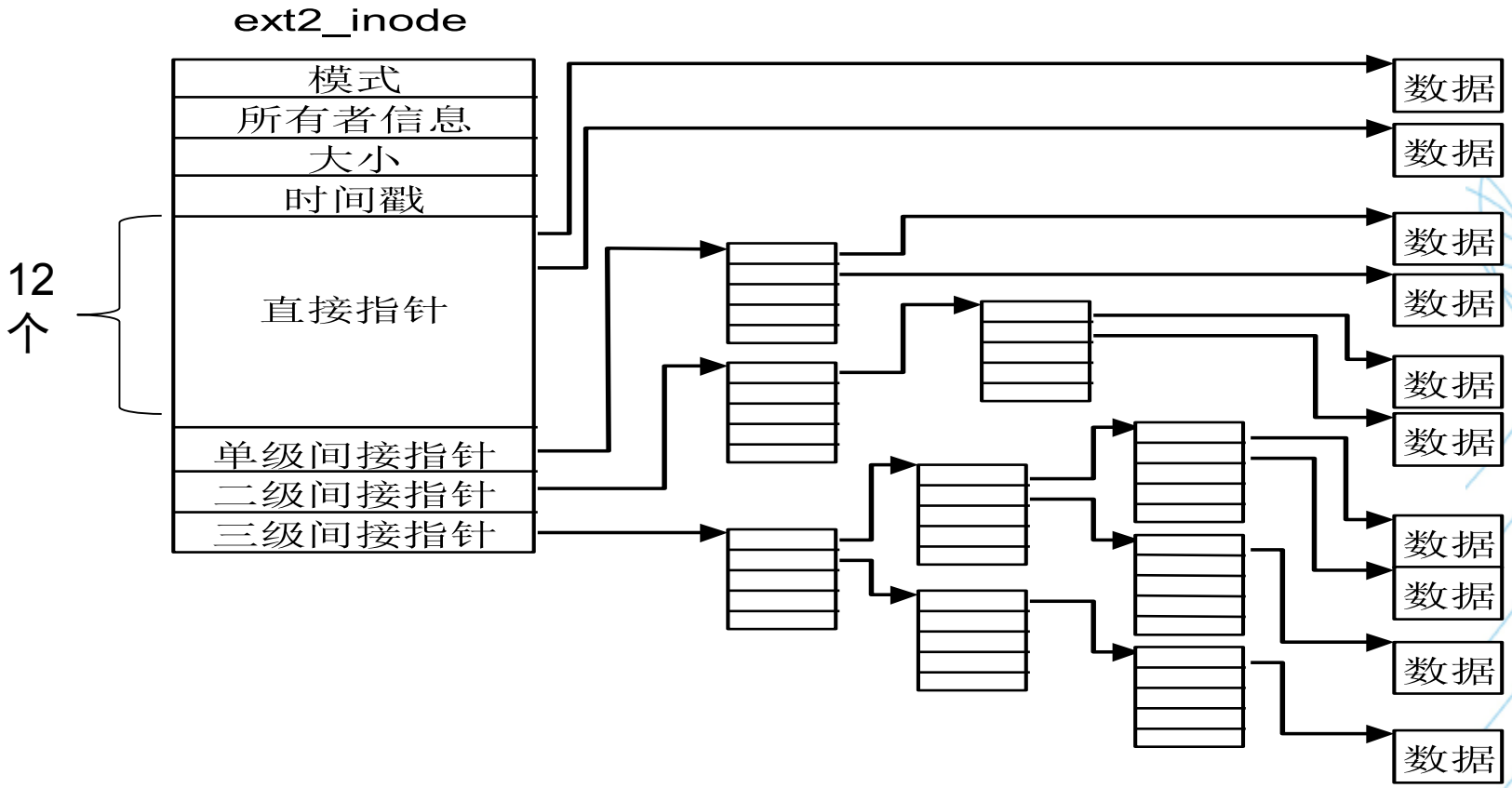
- Ext2 文件系统利用索引节点（inode）来描述文件系统的拓扑结构。在单个文件系统中，每个文件对应一个索引节点，而每个索引节点有一个唯一的整数标识符。文件系统中所有文件的索引节点保存在索引节点表中。
- Ext2 文件系统中的目录实际是一种特殊文件，它们也有对应的索引节点，索引节点指向的数据块中包含该目录中所有的目录项（文件、目录、符号链接等），每个目录项对应自己的索引节点。

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

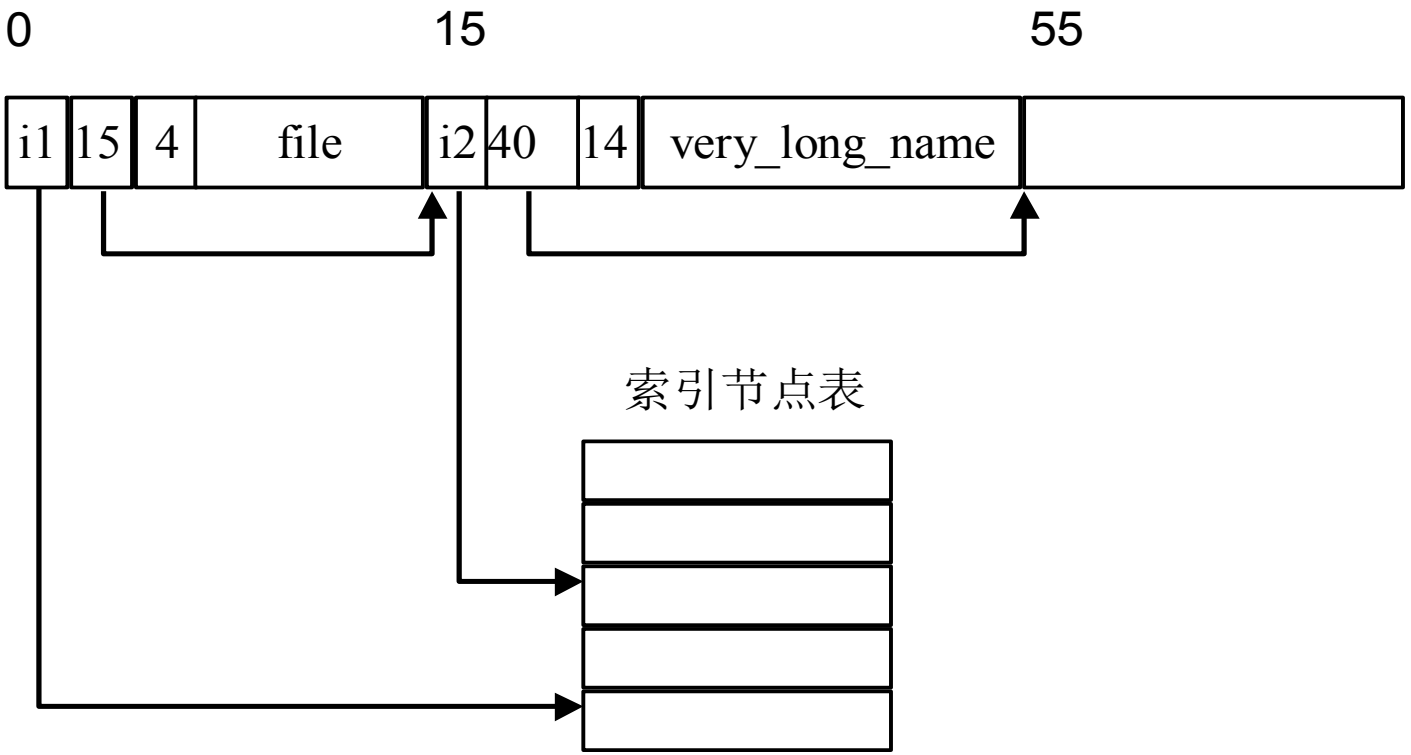
- 文件系统的超块（Superblock）包含了文件系统的基本大小和形式。其中包含的数据由文件系统管理程序用来进行文件系统的维护。每个块组中包含有相同的超块信息，但通常只需读取块组 0 的超块。

- 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。

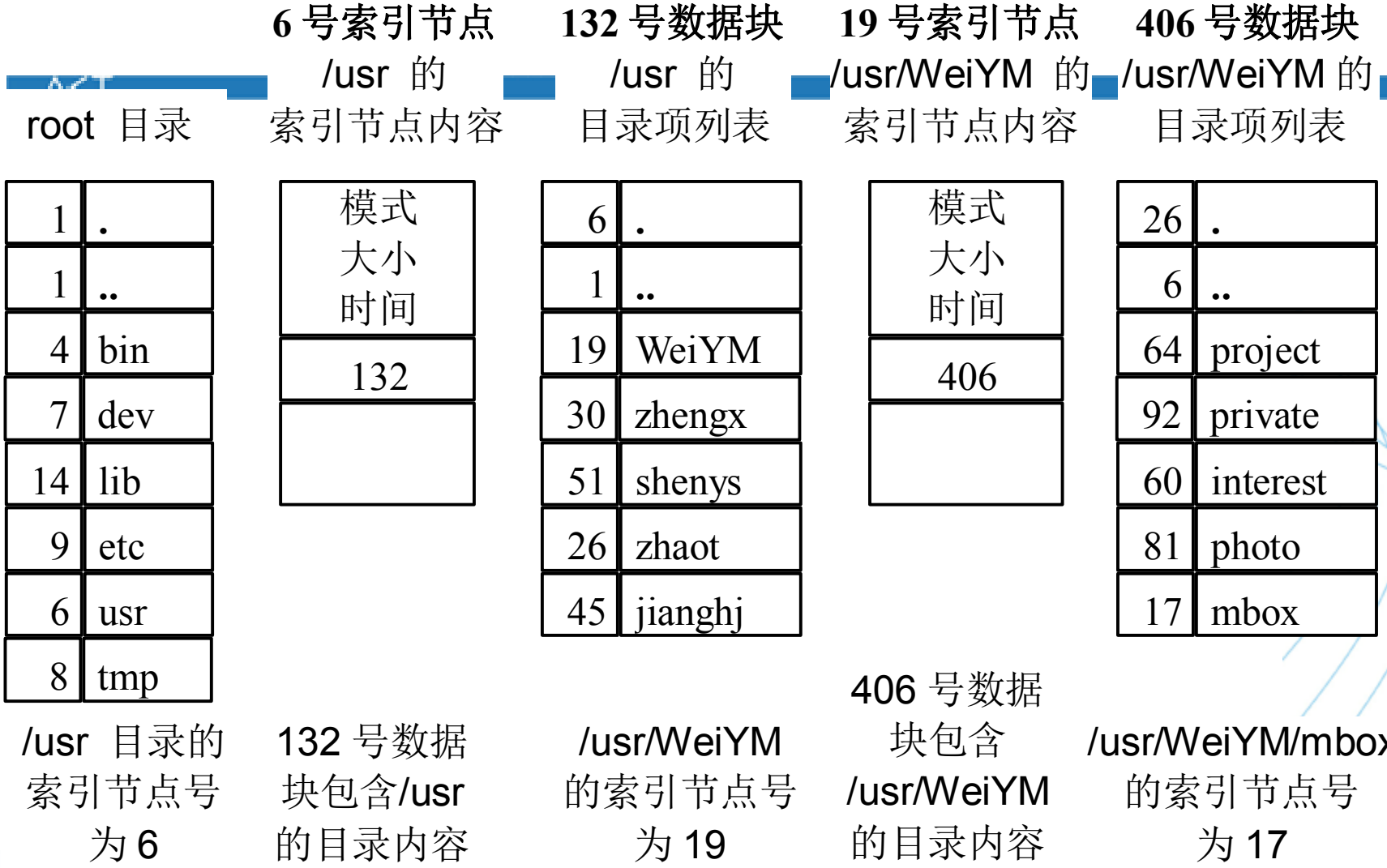
Ext2 块组描述符

- 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

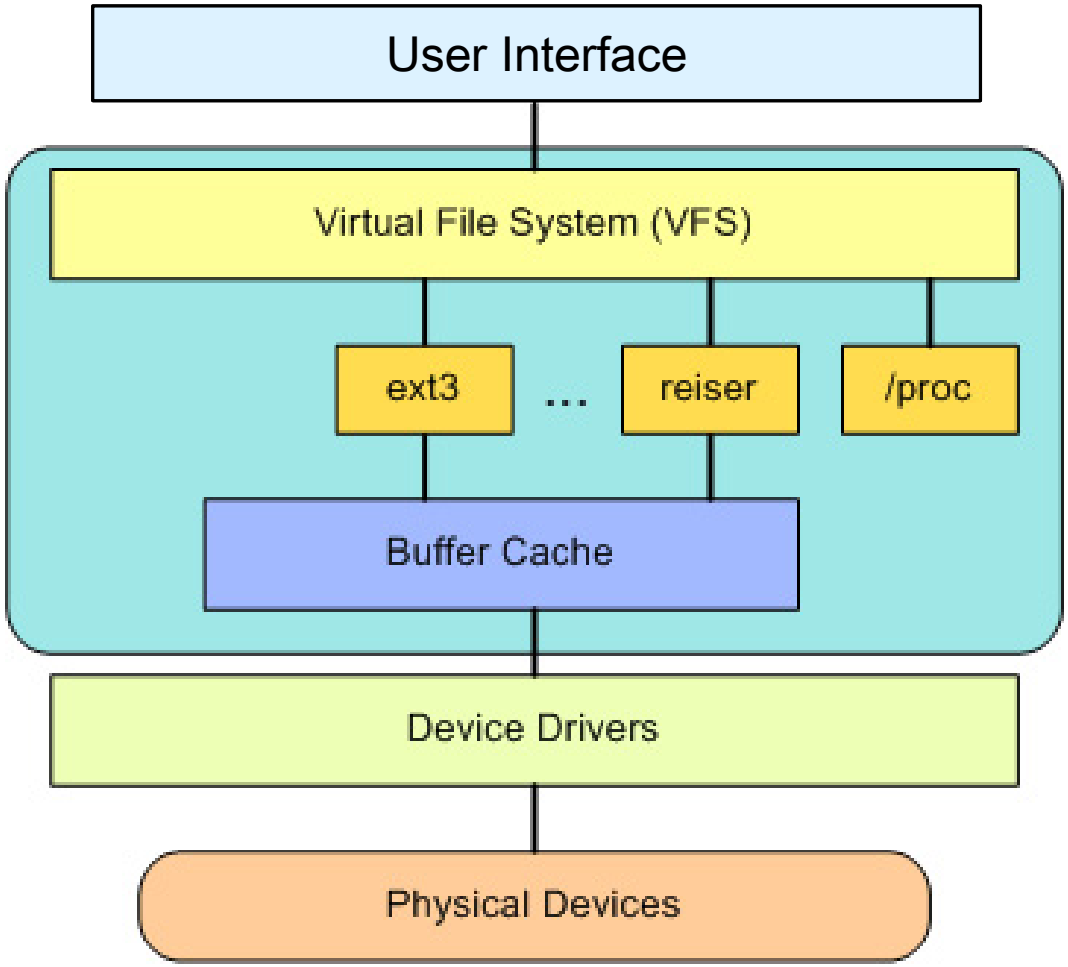


Root目录总是位于第2号索引节点。

Linux虚拟文件系统

- Linux 将独立的文件系统组合成了一个层次化的树形结构，并且由一个独立实体代表这一文件系统。
- Linux 将新的文件系统通过一个称为“挂装”或“挂上”的操作将其挂装到某个目录上，从而让不同的文件系统结合成为一个整体。
- Linux 的一个重要特点是它支持许多不同类型的文件系统，并且将它们组织成一个统一的虚拟文件系统。
  - Linux 中最普遍使用的文件系统是 Ext2，它也是 Linux 土生土长的文件系统。
  - 也支持 FAT、VFAT、FAT32、MINIX 等不同类型的文件系统，从而可以方便地和其它操作系统交换数据。

Linux虚拟文件系统结构



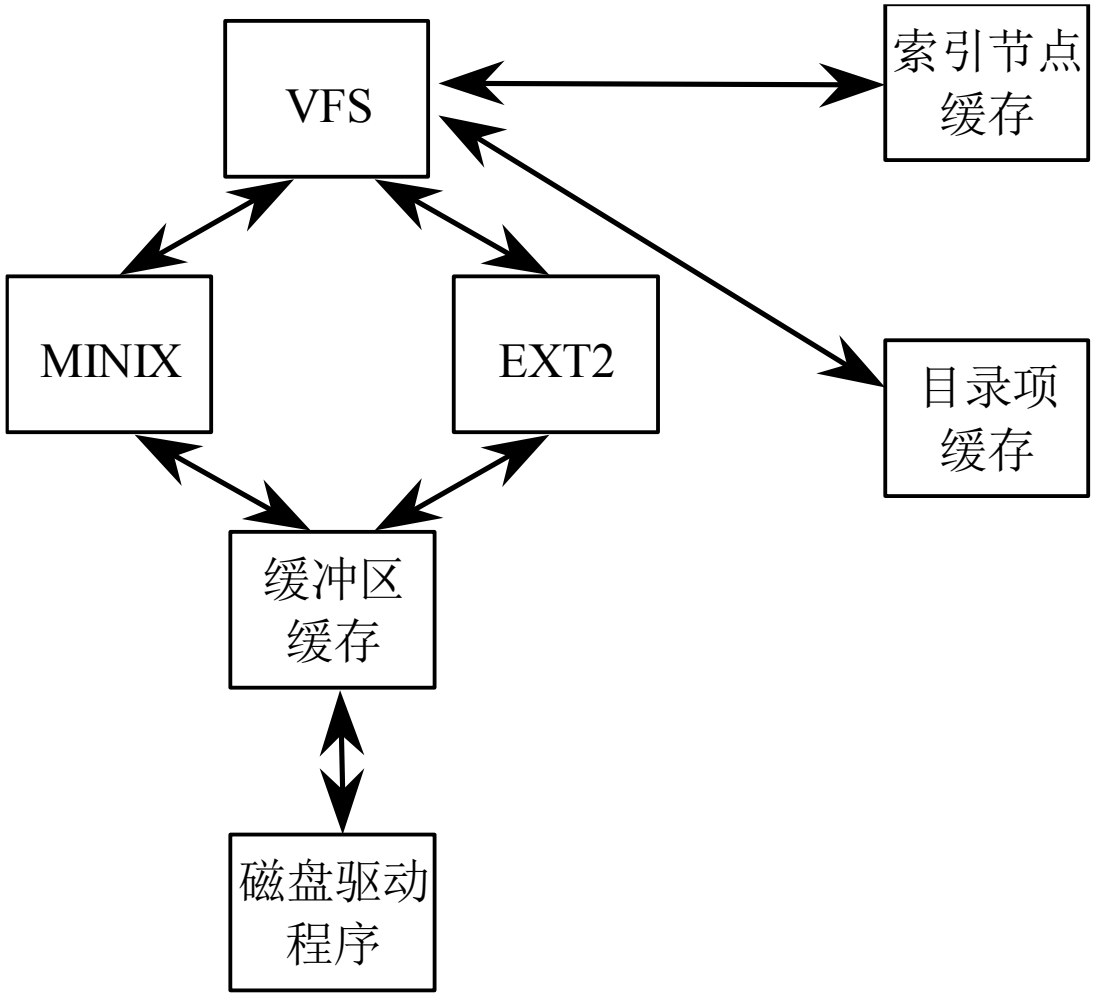
虚拟文件系统（Virtual File System,VFS）

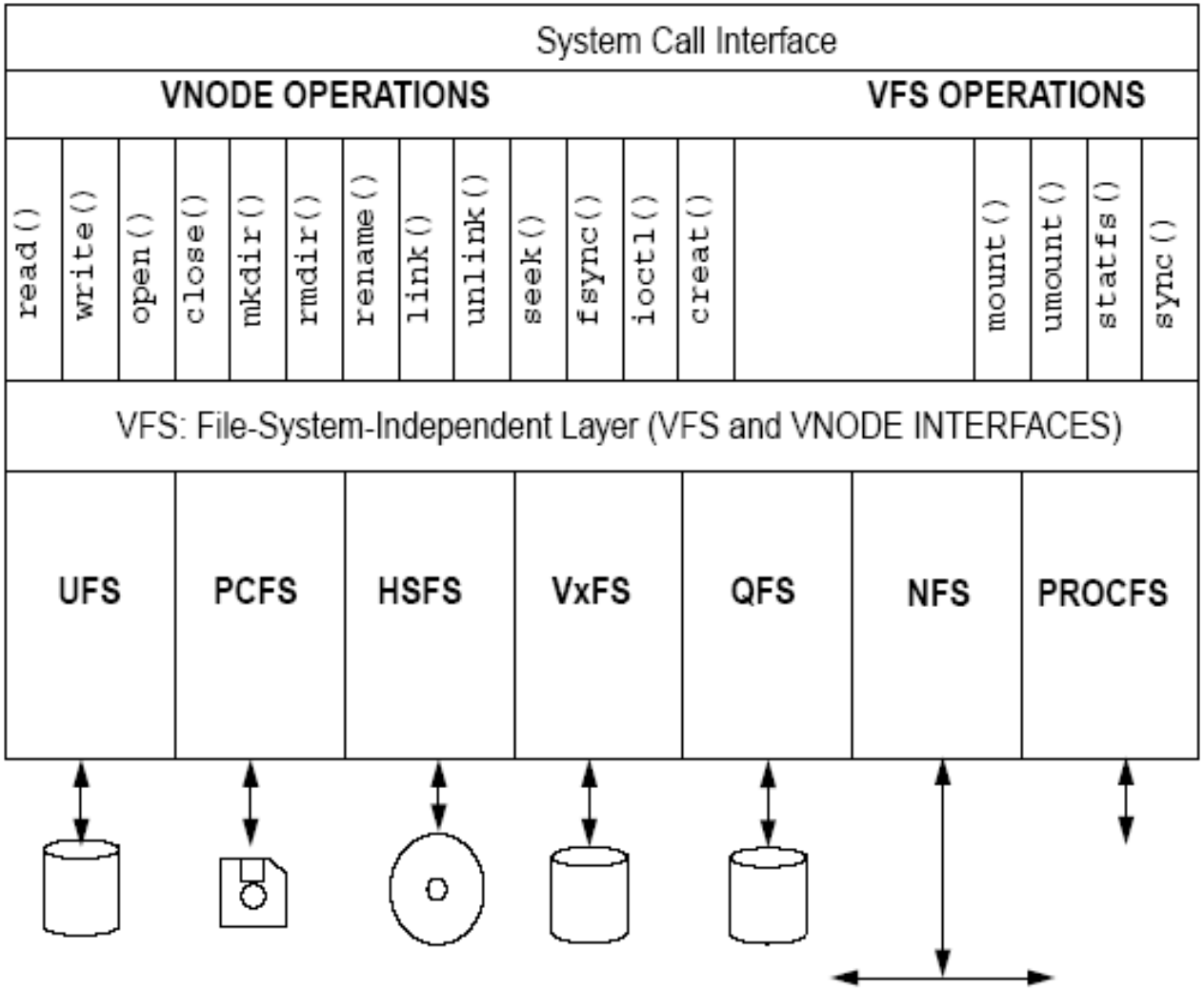
- 隐藏了各种硬件的具体细节，把文件系统操作和不同文件系统的具体实现细节分离了开来，为所有的设备提供了统一的接口，VFS提供了多达数十种不同的文件系统。
- 虚拟文件系统可以分为逻辑文件系统和设备驱动程序。逻辑文件系统指Linux所支持的文件系统，如ext2,fat等，设备驱动程序指为每一种硬件控制器所编写的设备驱动程序模块。
- 虚拟文件系统（VFS）是 Linux 内核中非常有用的一个方面，因为它为文件系统提供了一个通用的接口抽象。即VFS 在用户和文件系统之间提供了一个交换层。

虚拟文件系统（Virtual File System,VFS）

- 在 VFS 上面，是对诸如 open、close、read 和 write 之类的函数的一个通用 API 抽象。接口下面是文件系统抽象，它定义了上层函数的实现方式。VFS可挂接超过 50 个特定文件系统。文件系统的源代码可以在 ./linux/fs 中找到。
- 文件系统层之下是缓冲区缓存，这个缓存层通过将数据保留一段时间（如预先读取数据，以便在需要时使用）优化了对物理设备的访问。缓冲区缓存之下是设备驱动程序。
- 用户和进程不需要知道文件所在的文件系统类型，而只需要象使用 Ext2 文件系统中的文件一样使用。

虚拟文件系统




System Call Interface
VNODE OPERATIONS
VFS OPERATIONS
read()
write()
open()
close()
mkdir()
rmdir()
rename()
link()
unlink()
seek()
fsync()
ioctl()
creat()
mount()
umount()
statfs()
sync()
VFS: File-System-Independent Layer (VFS and VNODE INTERFACES)
UFS
PCFS
HSFS
VxFS
QFS
NFS
PROCFS

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。

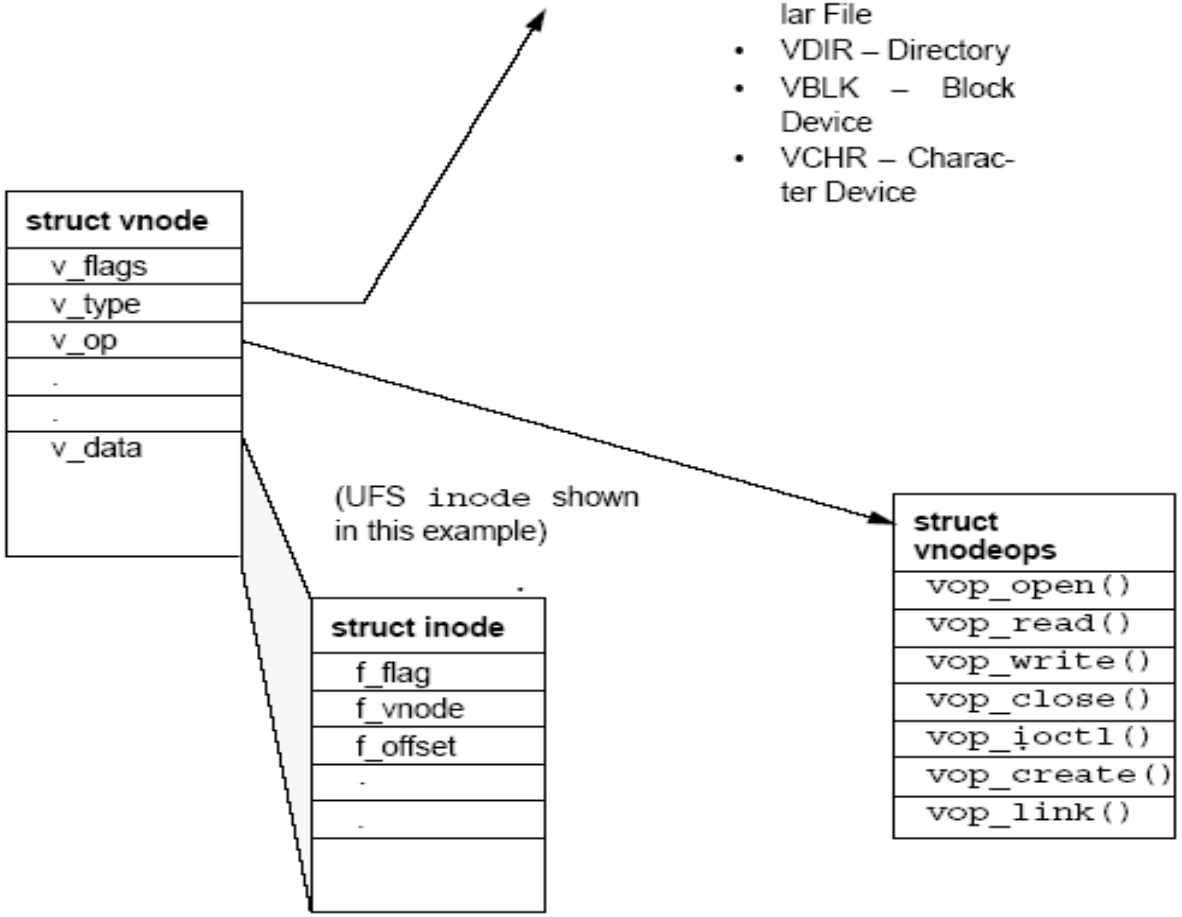
VFS 索引节点

- 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。

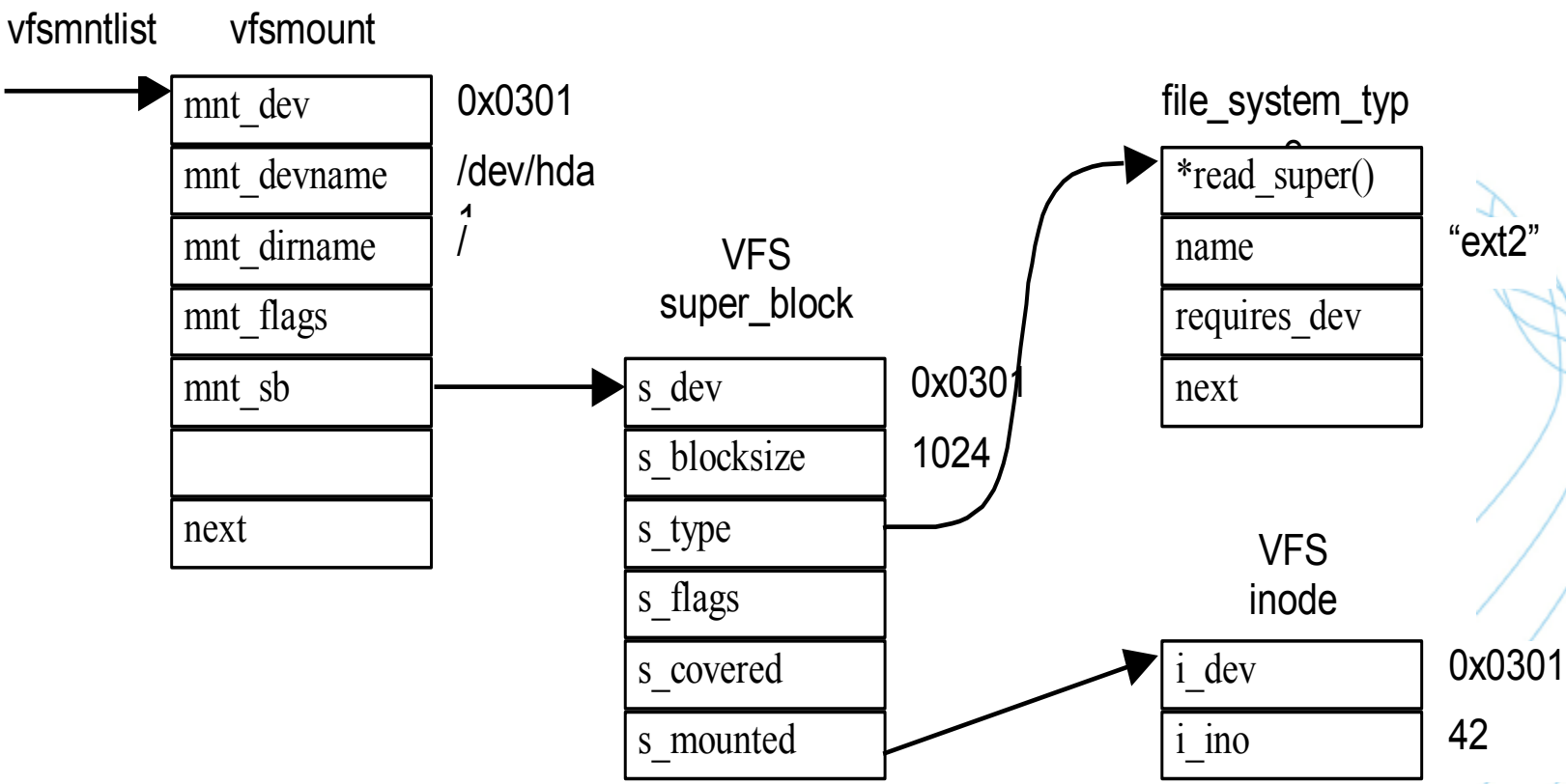
- VREG – Regular File
- VDIR – Directory
- VBLK – Block Device
- VCHR – Character Device



文件系统的挂装和卸装

- 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。
- 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。
- 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。
- 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。


vfsmntlist
vfsmount
mnt_dev
mnt_devname
mnt_dirname
mnt_flags
mnt_sb
next
0x0301
/dev/hda
1
/
VFS
super_block
s_dev
s_blocksize
s_type
s_flags
s_covered
s_mounted
0x0301
1024
file_system_typ
*read_super()
name
requires_dev
next
"ext2"
VFS
inode
i_dev
i_ino
0x0301
42

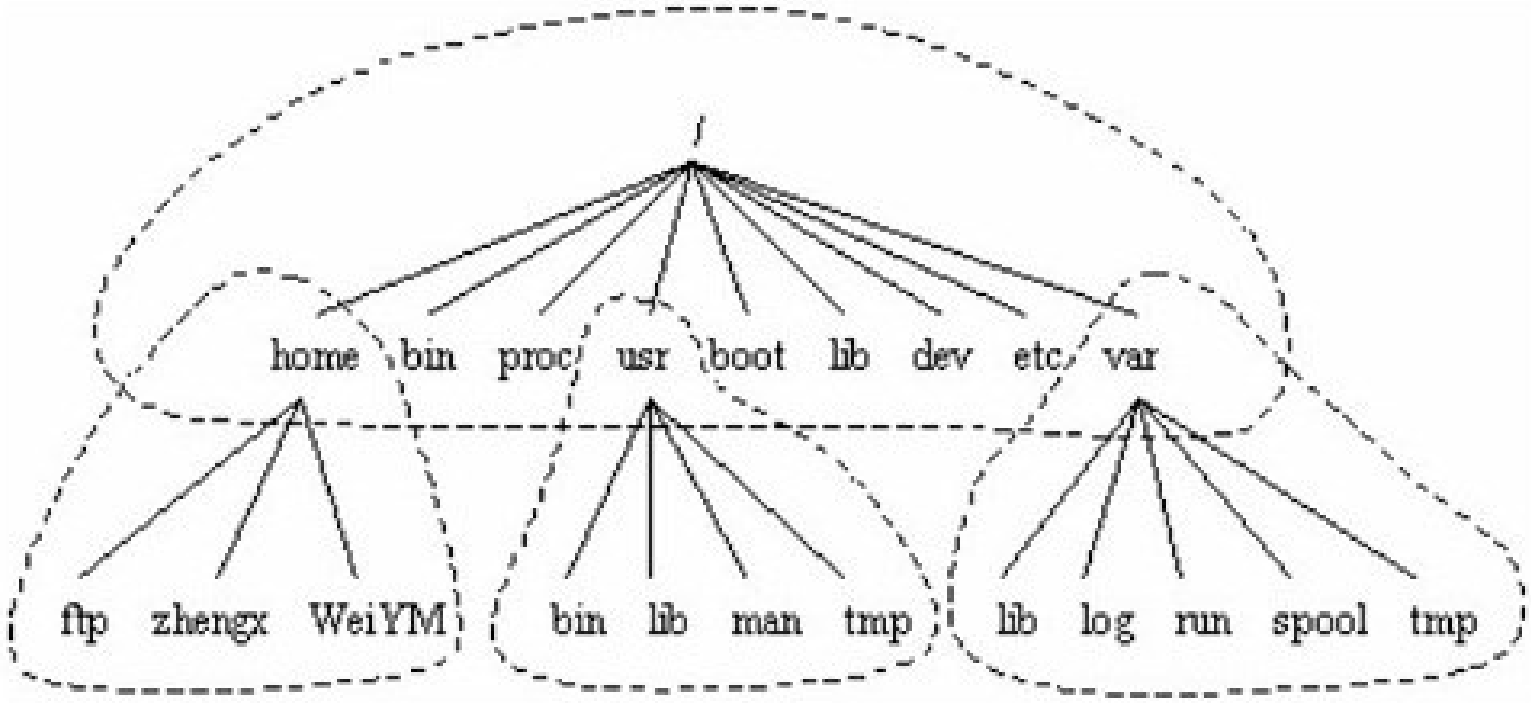
Linux文件类型

- Linux下面的文件类型主要有：
  - 普通文件：C语言元代码、SHELL脚本、二进制的可执行文件等。分为纯文本和二进制。
  - 目录文件：目录，存储文件的唯一地方。
  - 链接文件：指向同一个文件或目录的的文件。
  - 设备文件：与系统外设相关的，通常在/dev下面。分为块设备和字符设备。
  - 管道(FIFO)文件：提供进程建通信的一种方式。
  - 套接字(socket) 文件：该文件类型与网络通信有关

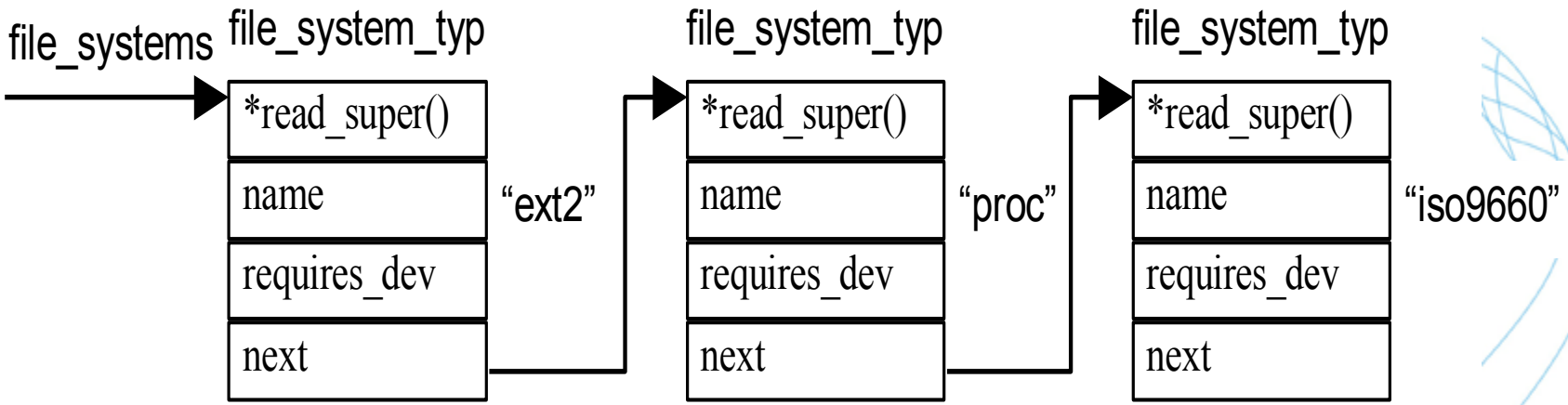
Linux目录

- 文件结构是文件存放在磁盘等存储设备上的组织方法。主要体现在对文件和目录的组织上。
- Linux使用标准的目录结构，在安装的时候，安装程序就已经为用户创建了文件系统和完整而固定的目录组成形式，并指定了每个目录的作用和其中的文件类型。
- Linux采用的是树型结构。最上层是根目录，其他的所有目录都是从根目录出发而生成的。完整的目录树可划分为小的部分，这些小部分又可以单独存放在自己的磁盘或分区上。这样，相对稳定的部分和经常变化的部分可单独存放在不同的分区中，从而方便备份或系统管理。

Linux目录



文件系统的注册



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，文件名程度不能超过 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 文件系统驱动程序的扩展，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。

思考与阅读

- 如果一个文件正在写入时系统crash或断电，会怎么样？
  - 谁是坏人？
  - 我不是

- 日志文件系统
  - http://www.linfo.org/journaling_filesystem.html

基于日志结构的文件系统 log-structured file system

什么是日志？

- 生活中，日志必须把一天中发生的事情全部翔实地记录下来，要求真实客观不加杂任何个人感情成份。
- 计算机中，记录程序、服务或操作系统产生的消息的文件。

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.

提出背景及基本结构

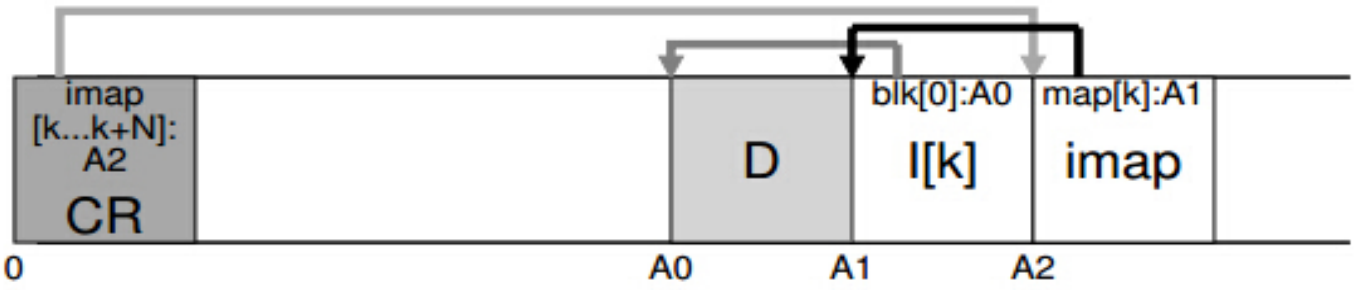
- LFS提出的动机是提高大量的“小量写”的效率。
- LFS将整个磁盘看作一个日志(log)，并周期性地追加(Append)新日志。
- 写操作并非被直接反映到磁盘上，而是被暂存到内存的缓冲区中，其中包括新写的数据，也包括更新数据。当其积累到一定规模时，作为一个segment追加到日志的末尾。
- 一个segment包括inode、目录、以及数据块。每个segment的开始处包含一个总结信息(summary)，说明该segment的信息组织情况。

LFS的特点

- 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: （imap）一个存放此段inode节点在磁盘上的位置的表；
  - inode map块作为段的一部分，在磁盘的固定区域存放的检查点区(CR: checkpoint region) 中保存有指向这些inode map块的指针。

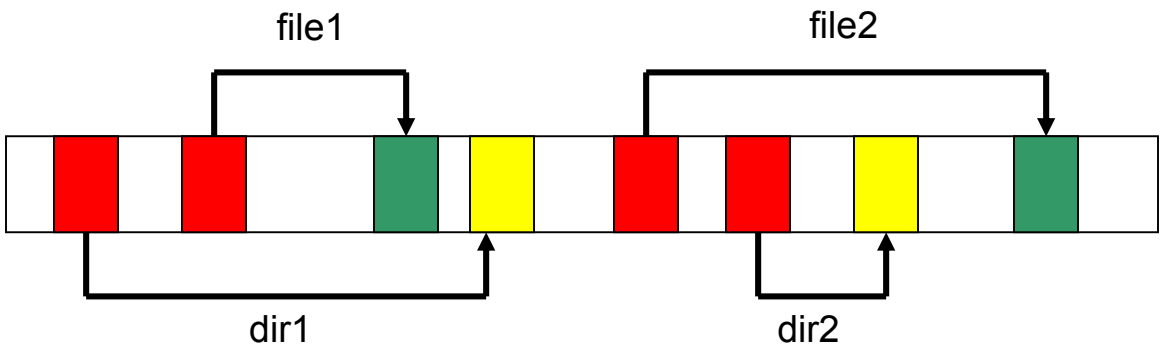


LFS的数据结构

- 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。

  - **由于inode的存储位置不再能由其编号确定，系统维护一个inode map，以实现i-number到磁盘inode位置的映射，inode map保存在磁盘上，但被缓冲到内存以提高访问速度。打开文件时由i-number查inode map找到inode，进而找到文件数据块。**

LFS vs. UFS



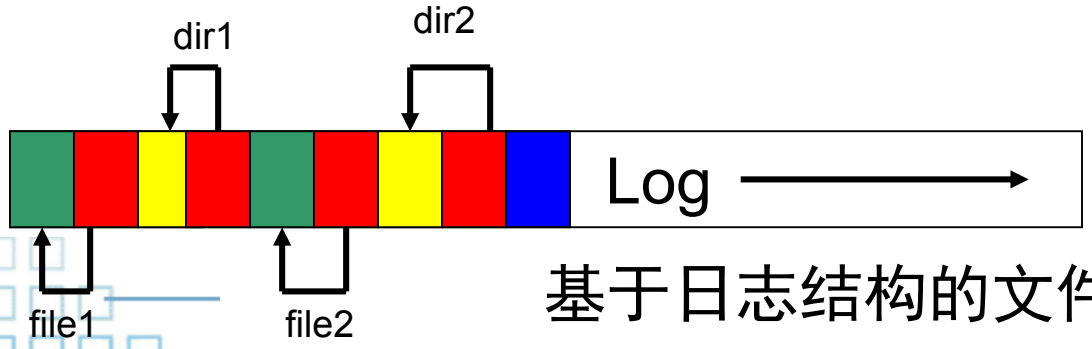
Unix 文件系统

inode

目录

数据

inode map



基于日志结构的文件系统

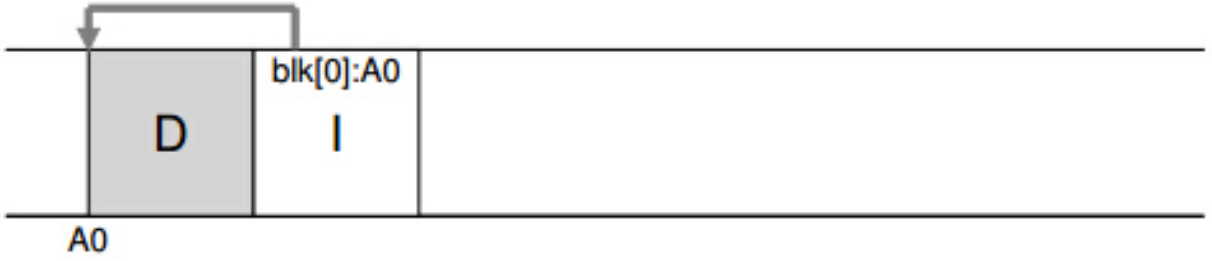
在Unix 文件系统和LFS下创建两个包含一个数据块文件：dir1/file1和 dir2/file2

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；
- 特别要注意的是：LFS并不会覆写已有的数据，而是把segment中的数据写入到磁盘中新的位置。
- 这样在磁盘中就会有old data和new data。

LFS：用顺序化提高性能

- 怎么把对文件系统状态的修改更新转换为一个序列操作呢？在常规文件系统中，当我们在新建文件或文件夹，以及向文件中添加或删除数据时后，不仅仅是文件数据在磁盘的写入，与文件相关的i-node也要写入磁盘，i-node和文件所在磁盘位置相差可能很远。
- 而LFS就通过把数据更新缓存到内存中，然后一起写入到磁盘中，将文件数据和i-node放在磁盘相邻的位置，减少磁盘寻道时间。

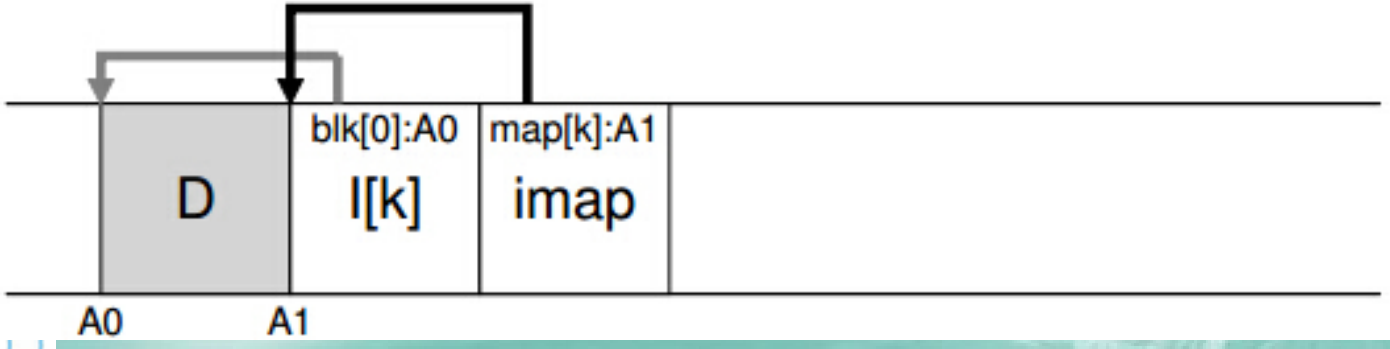


LFS：如何找到i-node

- 在传统的文件系统比如UNIX系统中，i-node存在一个数据结构中，保存在磁盘中固定的位置。因此只要给定inode的序号以及i-node所在磁盘的地址，就可以通过i-node序号乘以i-node结构的长度，然后再加上磁盘地址就可以得到所需i-node的精确地址。
- 在LFS系统中，i-node是分布在磁盘中的，并且由于在LFS系统中并不覆写数据，所以一个文件的最新版本的i-node在磁盘中的位置是不断变化的。
- LFS通过引入imap来解决i-node查找问题。imap是这样的一个结构，输入一个inode number，生成最新版本的inode所在磁盘地址。基于此，imap可以实现为一个数组，每一项是4字节作为磁盘指针，当一个inode写入磁盘后，imap就更新相应的inode地址。

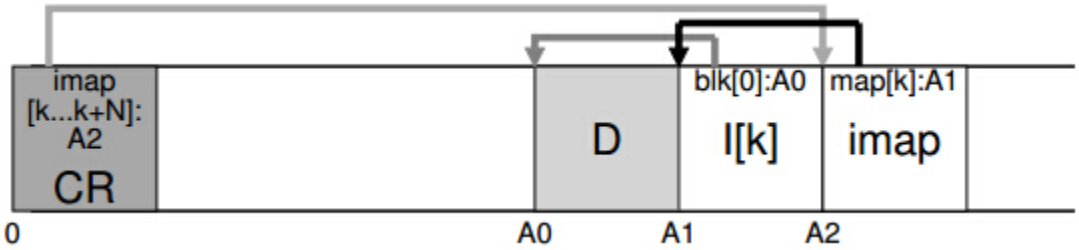
LFS：如何找到i-node

- LFS需要imap能够持久的保存，这样在系统崩溃的时候就能够找到inode的位置。
- 如果把imap保存在磁盘固定的位置上的话，每次更新的时候都要访问这个磁盘，就会增加磁盘读写时间，降低了系统性能，所以在LFS中，把更新相关的imap存放在更新数据所写入磁盘位置的右边。



LFS：如何寻找i-node map

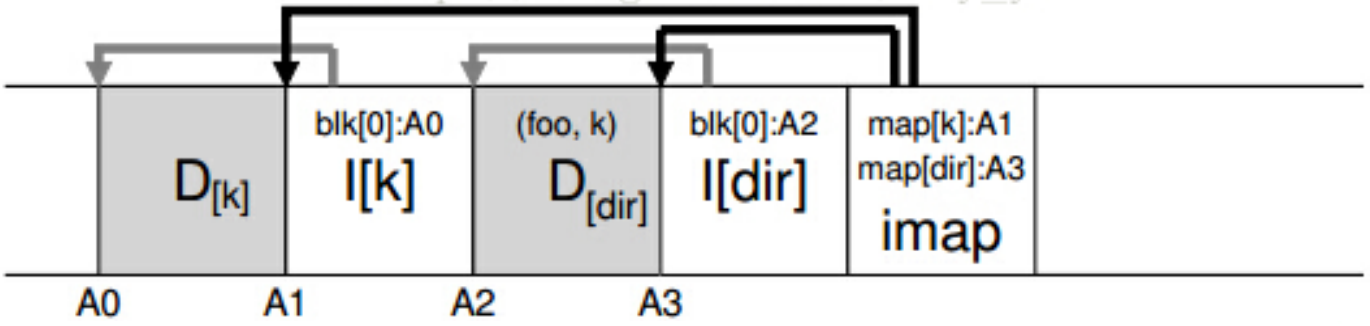
- imap分散保存在磁盘的各个位置上，那么使用时该如何找到这些imap呢？系统中必须在一个固定磁盘的位置存储这些imap的信息。在LFS中这个固定保存imap地址信息的块称为CR（Check-point Region）。



- **在CR中保存指向最新版本的imap的指针，系统就可以通过读取CR查找获取想要的imap信息。**
- **LFS最终在磁盘上的整体结构包含一个CR，imap（将inode number映射为Inode的地址），以及指向文件的inode.**

LFS：如何保存目录数据

- 当在磁盘上创建一个文件，LFS必须写入一个新的inode，新的数据，以及指向这个文件的目录数据和目录inode(目录数据和Inode是被更新了，不过在LFS不会覆写数据，所以这些数据都要在磁盘新的位置中写入)。

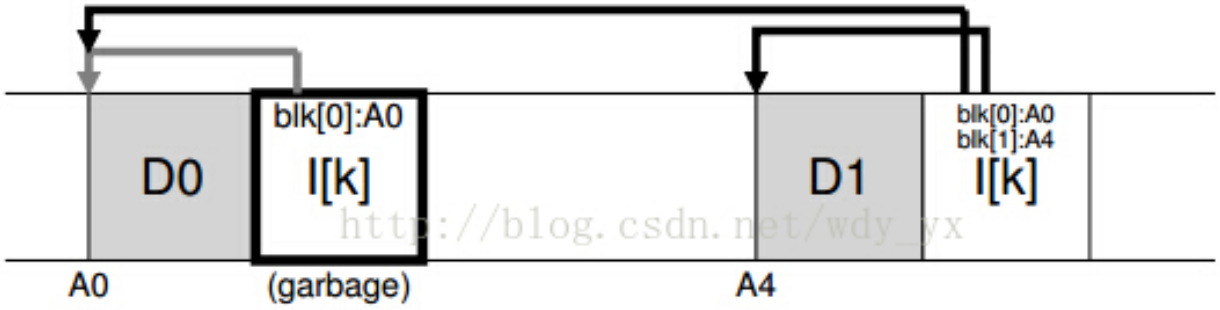


LFS：文件版本

- LFS不覆写数据，只是在新的磁盘位置写入数据，磁盘中同时有大量的老版本的数据，LFS怎么解决这个问题？
- 假设有个inode number为k的inode指向磁盘块D0，我们现在更新了文件数据，那么在磁盘中生成一个新的inode和数据磁盘块。磁盘中就会出现这种情况，inode,数据磁盘块有两个版本，一个老版本，一个新版本。

LFS：文件版本

- 在一个文件中添加数据，在磁盘中为原始文件增加了一个新的磁盘块用于保存添加的数据，可以看到，磁盘中增加了一个新的inode，但是原有的数据块除了被新的inode指向外，还依然被老版本的inode指向。



- **在LFS中只保存最新的版本和之前的老版本。对于这些老版本的inode，数据块等应该怎么处理？一个方法是保存这些老版本的数据然后允许用户恢复到老版本。这样的文件系统称为versioning file system。**

LFS：磁盘清理

- LFS只是在新的磁盘位置写入数据，磁盘总有写满的时候，LFS怎么解决这个问题？
- 在LFS中只保存最新的版本和之前的老版本，LFS周期性的扫描磁盘检测老的版本数据然后清除掉，为后续数据的腾出磁盘空间。
- LFS最终是基于segment进行清理操作。基本的清理过程如下：LFS首先读入一些老版本的segment，然后判断在segment中那些磁盘块中的数据是Live的，在把这些live的数据写入到一个新的segment集合中，释放原有segment占用的磁盘空间。
- 读入N个旧的Seg，清理后变为M个Seg，N>M。

LFS：失效恢复

- 为了确保CR数据写入的原子性，LFS其实保存了两个CR,保存在磁盘的两端，CR更新到磁盘中时是交替写入。
- LFS同样实现了一个非常小心的协议来更新CR。
  - 首先写入一个header(附带写入一个timestamp)，然后写入CR的主要内容，然后写入最后一个磁盘块数据以及一个timestamp。
  - 如果在CR更新时系统崩溃了，LFS通过观察到一对不一致的timestamp检测到出现了系统崩溃，LFS就会采用含有一致性timestamp的离崩溃时间最接近CR。

LFS：失效恢复

- 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;
  - LFS恢复只主要数秒的时间，而Unix文件系统的恢复可能需要几个小时。

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；
- 需要清理进程来回收空闲磁盘空间，会由此带来附加的CPU开销。

问题？