操作系统 Operating System

第七章 文件系统

目录的实现

原仓周

yuancz@buaa.edu.cn

内容提要

- 文件系统基本概念
- 文件系统实现方法
  - 文件的实现
  - 目录的实现
  - 硬链接和软链接
  - 内存中的文件系统
  - 文件的保护
  - 文件系统的性能
- 文件系统实例分析

目录的主要功能和实现

目录的主要功能：

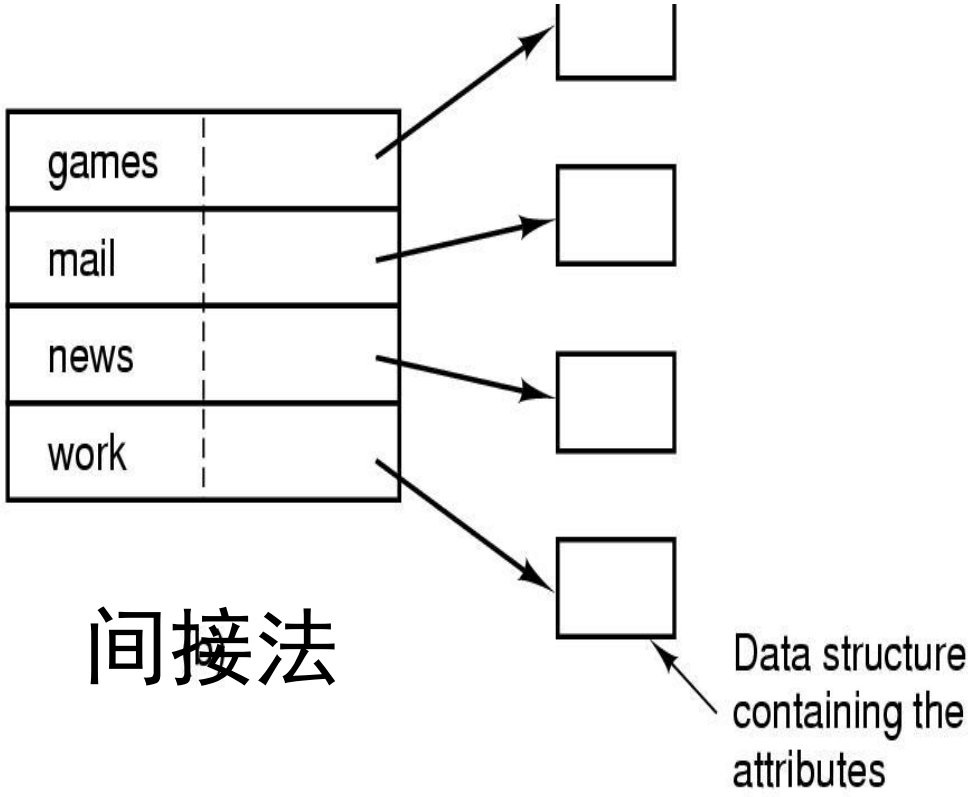
- 根据用户给出的ASCII形式的文件名（路径名），迅速地定位到相应的文件控制块
- 目录的实现需解决以下三个问题：
  - 目录项的内容
  - 长文件名问题
  - 目录的搜索方法

目录项的两类实现方法

- 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

两种实现目录项的方法

直接法



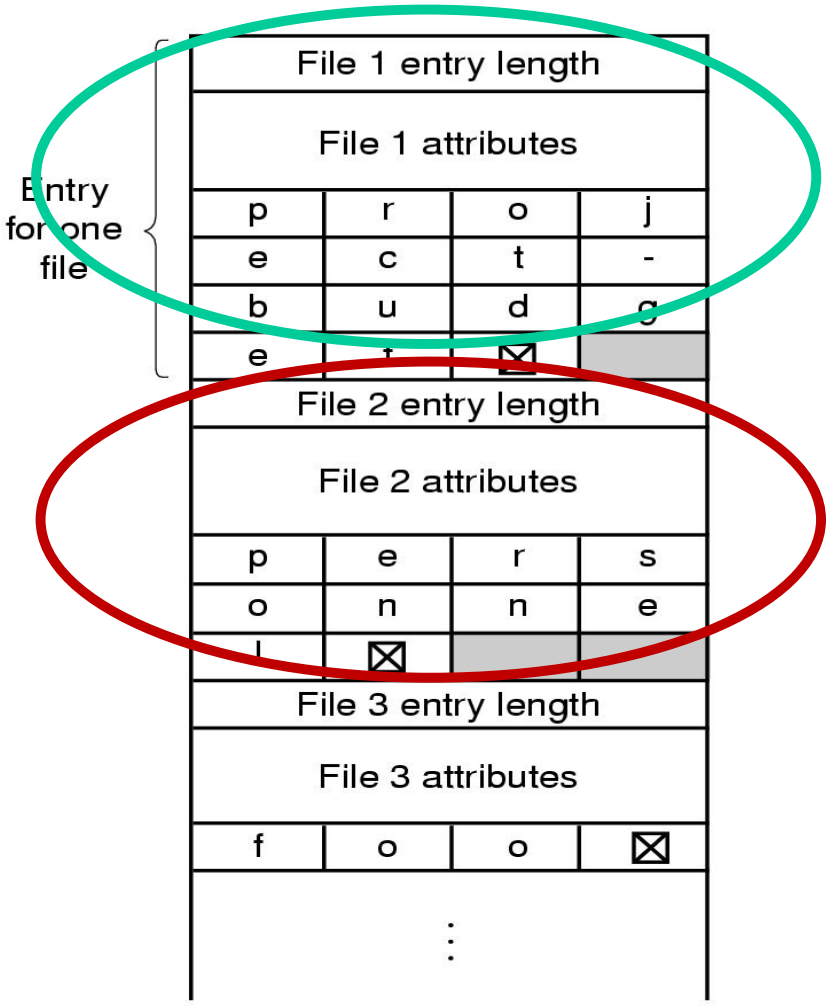
间接法

长文件名的三种实现方法

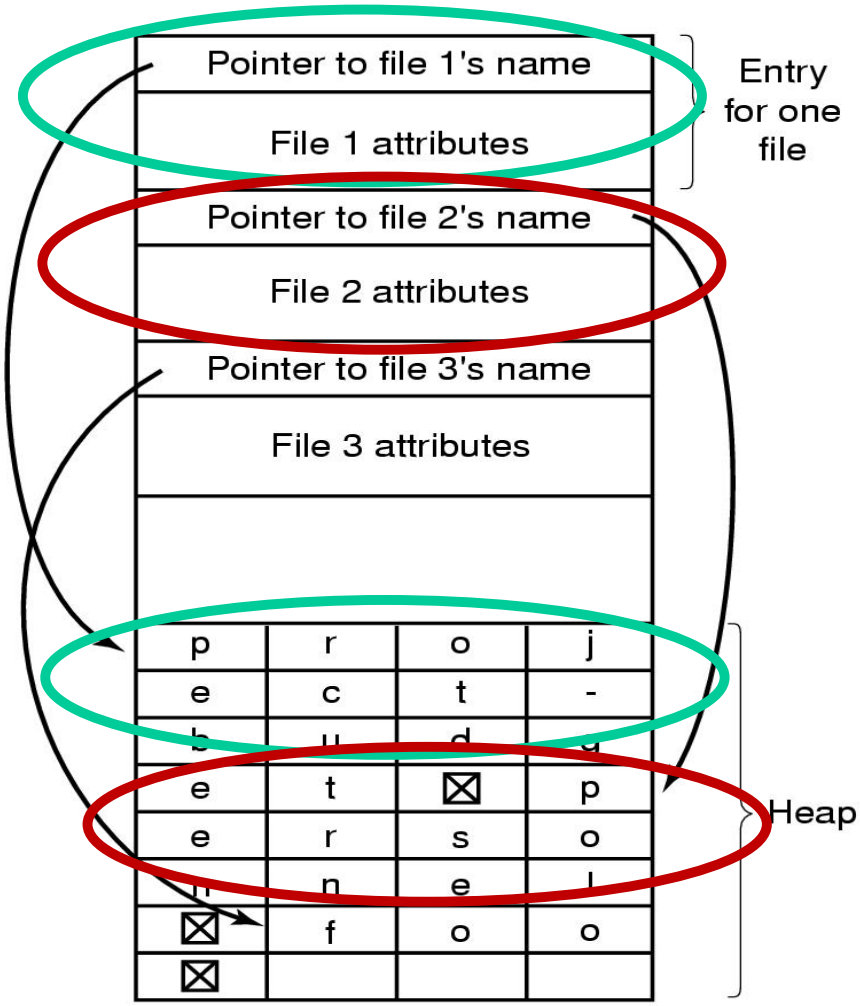
- MS-DOS：文件名的最大长度为8个字符，扩展名的最大长度为3个字符
- Unix V7：文件名的最大长度为14个字符

- 现代OS一般都支持可变长度的文件名。实现方法：
  - 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

长文件名的第2,3种实现方法



方法 2



方法 3

文件目录的两种查询技术

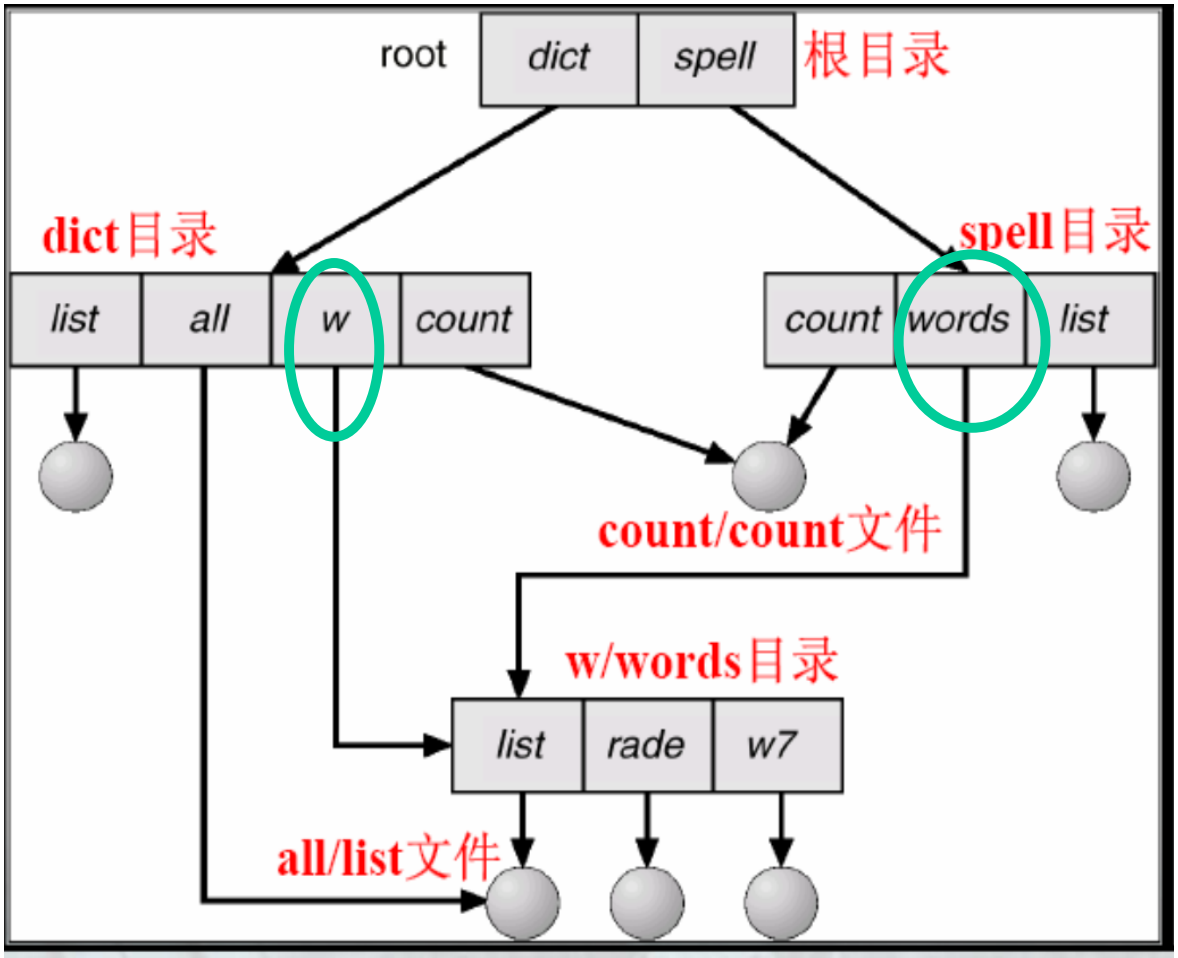
- 顺序查寻法
  - 依次扫描符号文件目录中的表目，将表目中的名字字段与查找的符号名NAME进行比较（只在表目不多时适用）
- Hash方法
  - 即“散列法”或成“杂凑法”
  - 是一种构造符号表、查询符号表常用的技术
  - 其基本思想是：利用一个易于实现的变换函数（即Hash函数），把每个符号名唯一的变换成符号表中的表目索引

内容提要

- 文件系统基本概念
- 文件系统实现方法
  - 文件的实现
  - 目录的实现
  - 硬链接和软链接
  - 内存中的文件系统
  - 文件的保护
  - 文件系统的性能
- 文件系统实例分析

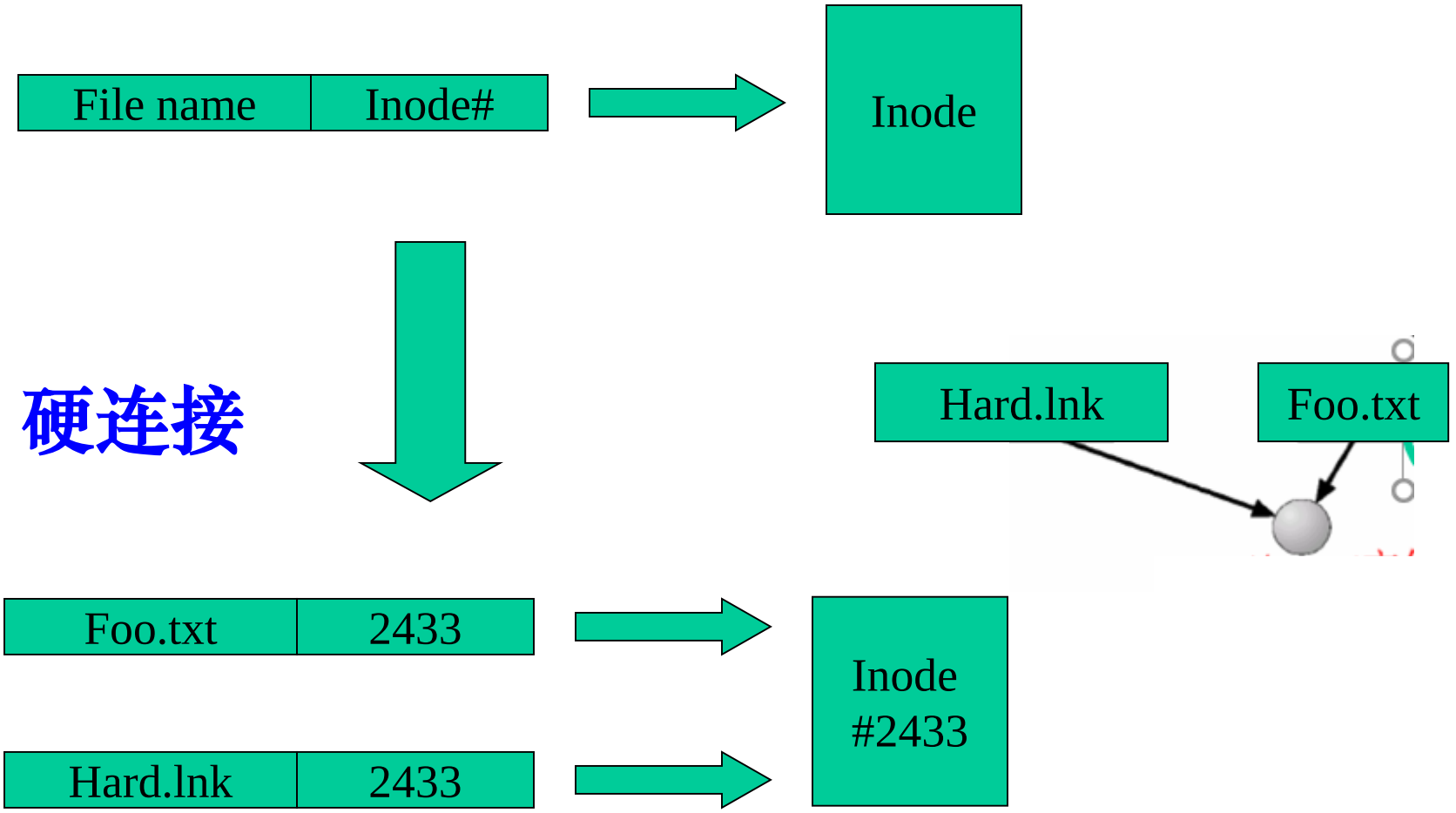
便于共享的目录组织

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



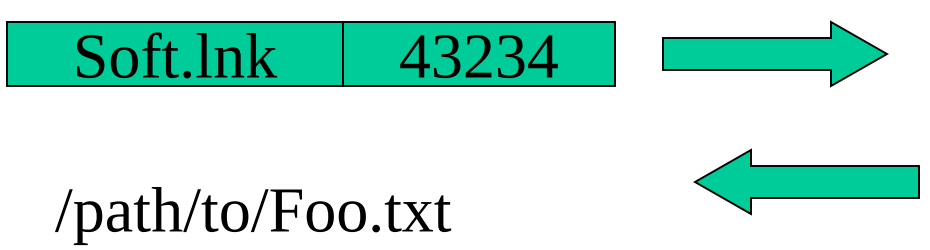
有向无环图

目录项 v.s. 硬连接

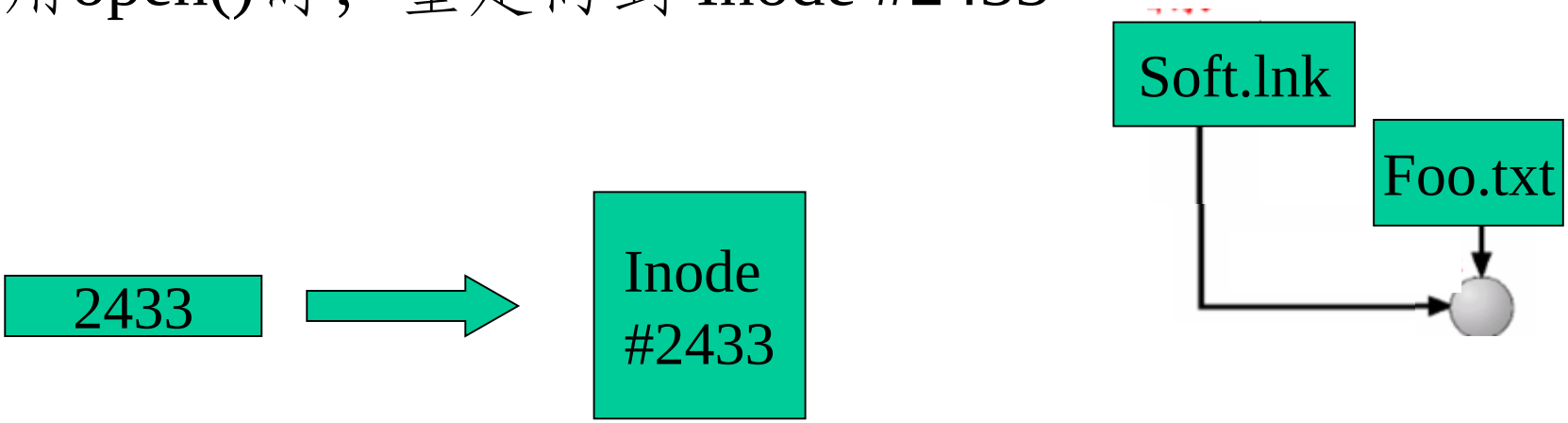

File name
Inode#
Inode
硬连接
Hard.lnk
Foo.txt
Foo.txt
2433
Hard.lnk
2433
Inode
#2433

目录项 v.s. 软连接

软连接/符号连接



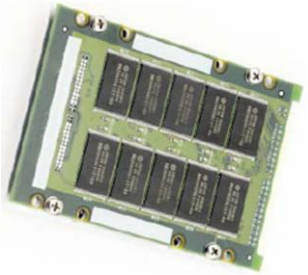
当调用open()时，重定向到 Inode #2433

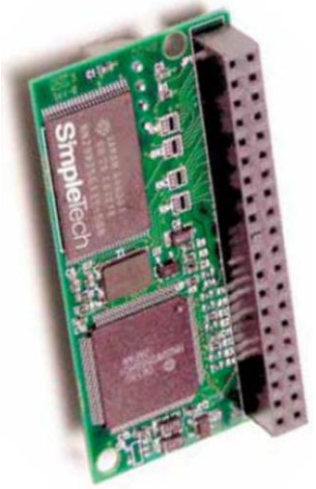


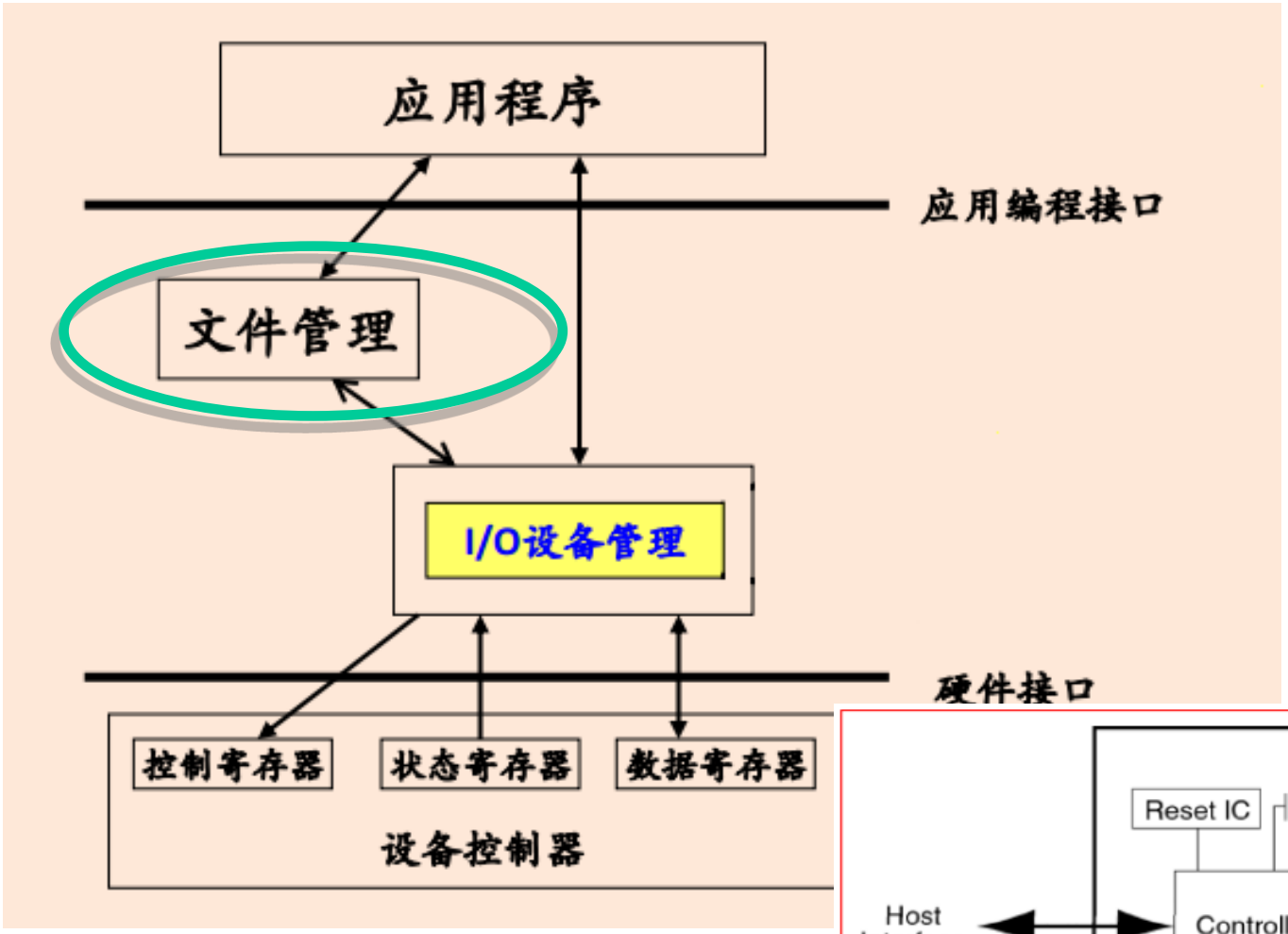
内容提要

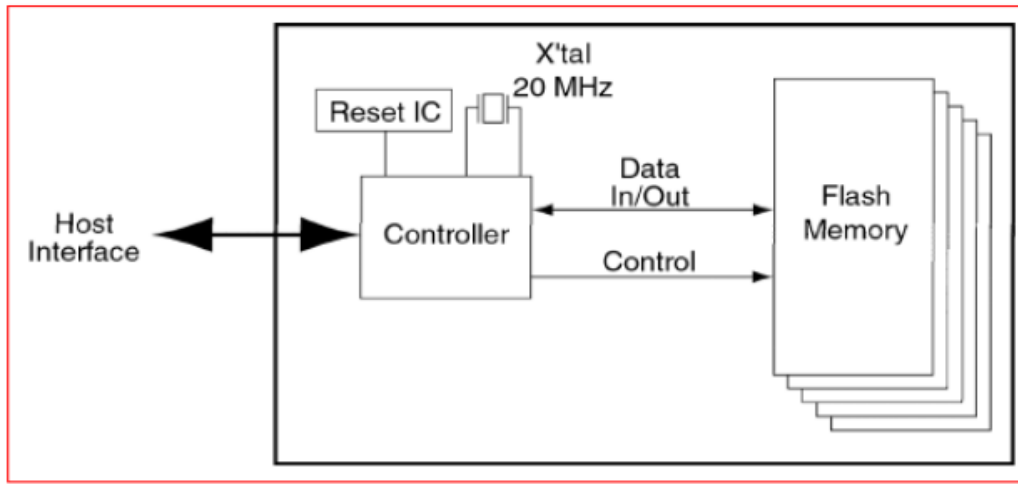
- 文件系统基本概念
- 文件系统实现方法
  - 文件的实现
  - 目录的实现
  - 硬链接和软链接
  - 内存中的文件系统
  - 文件的保护
  - 文件系统的性能
- 文件系统实例分析

文件系统在内存中的管理

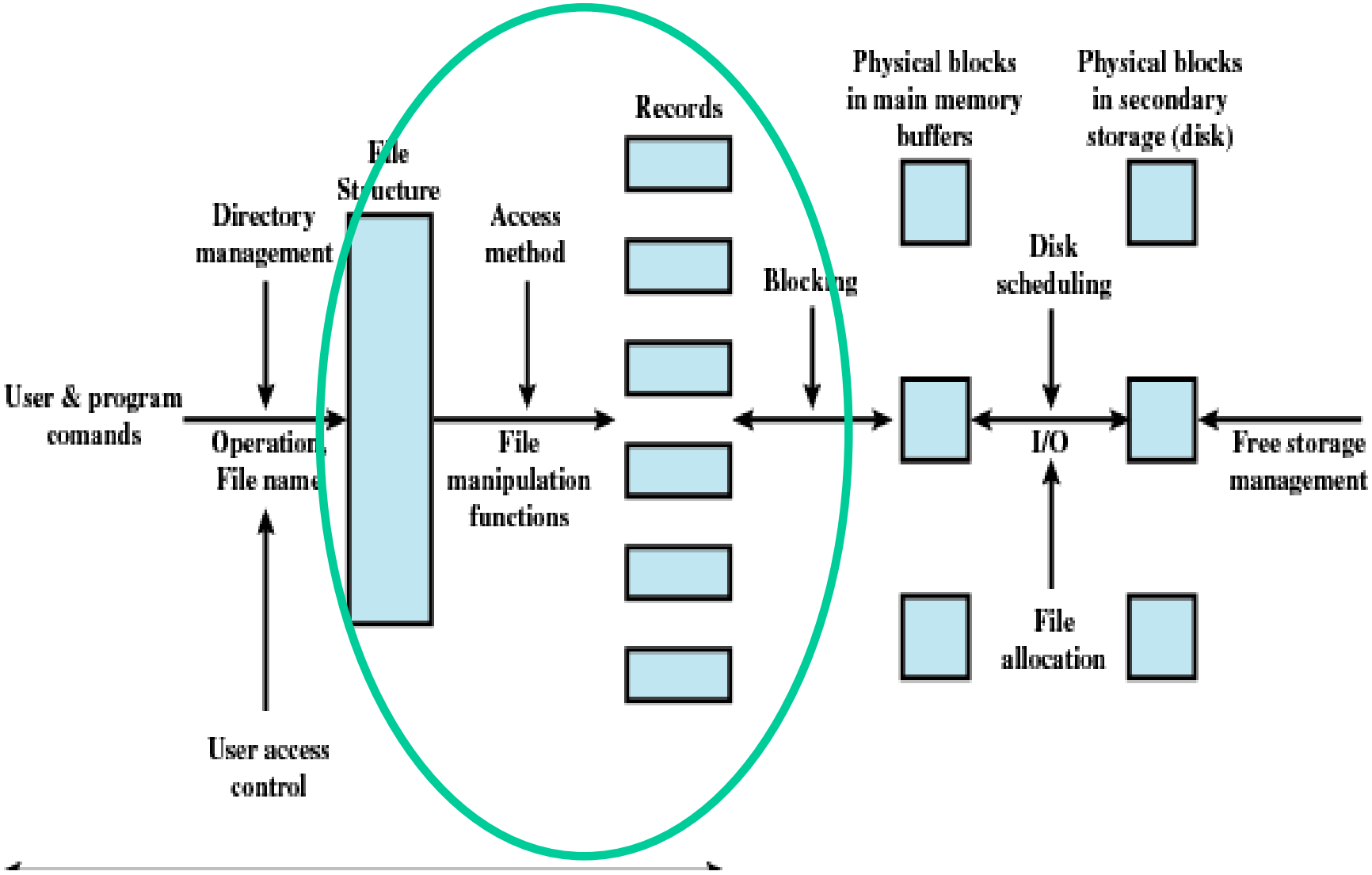




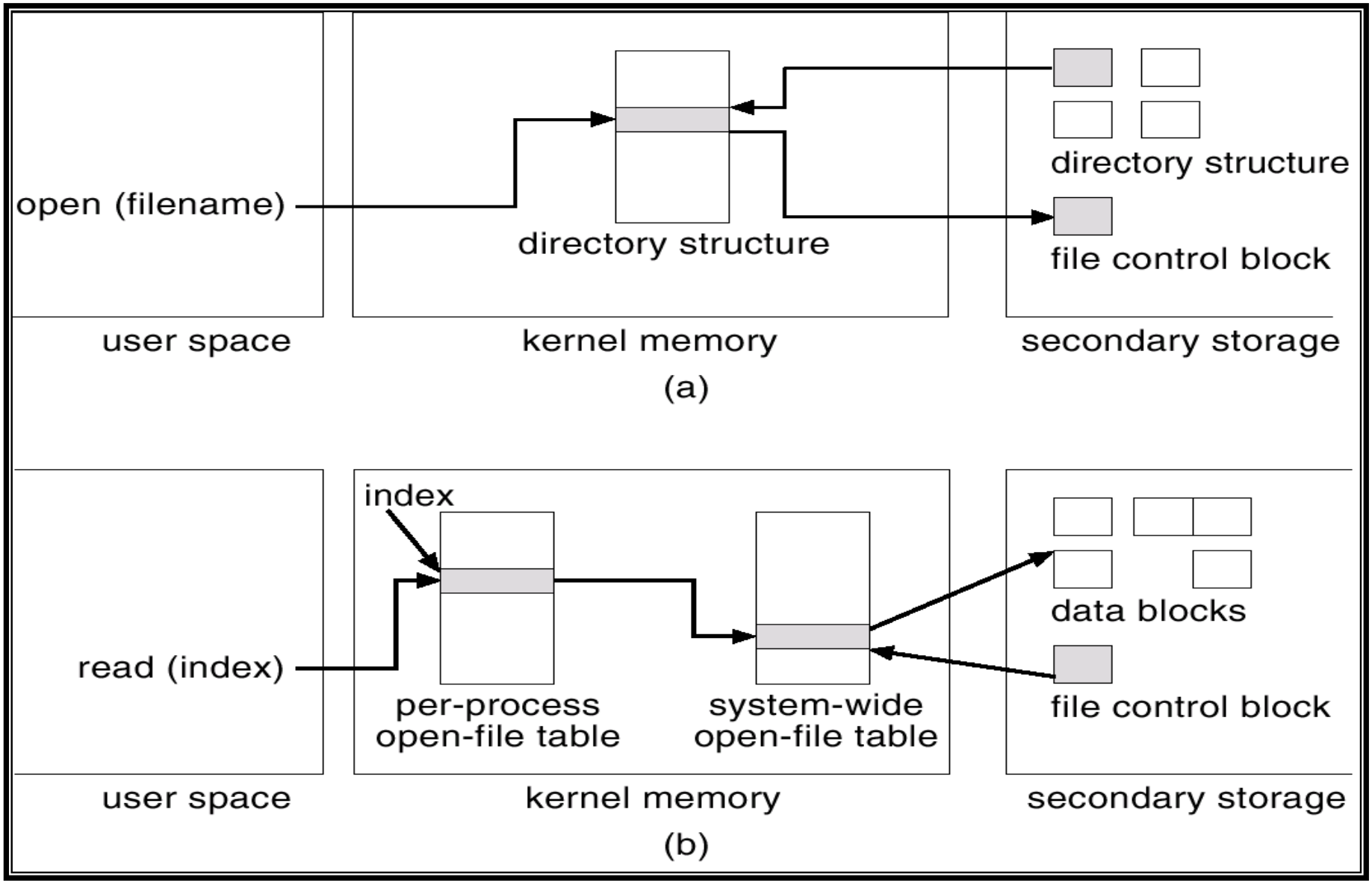




文件系统在内存中的管理结构



文件系统与进程的接口


open (filename)
directory structure
directory structure
file control block
user space
kernel memory
secondary storage
(a)
index
read (index)
per-process
open-file table
system-wide
open-file table
data blocks
file control block
user space
kernel memory
secondary storage
(b)

内容提要

- 文件系统基本概念
- 文件系统实现方法
  - 文件的实现
  - 目录的实现
  - 硬链接和软链接
  - 内存中的文件系统
  - 文件的保护
  - 文件系统的性能
- 文件系统实例分析

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

文件的存取权限控制

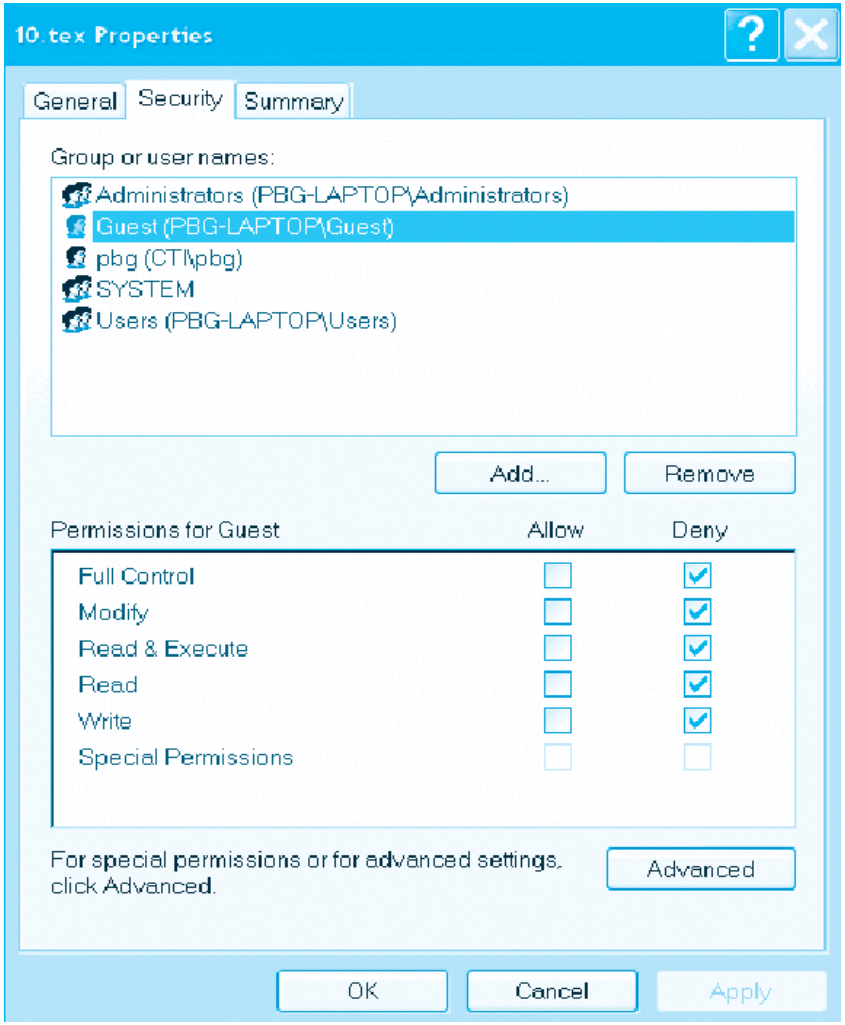
- **文件保护机制**
  - 防止未被核准的用户存取文件
  - 防止一个用户冒充另一个用户来存取
  - 防止核准用户（包括文件主）误用文件
- **存取权限验证步骤**
  - 审定用户的权限
  - 比较用户的权限与本次存取要求是否一致
  - 将存取要求和被访问文件的保密性比较，看是否有冲突

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



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

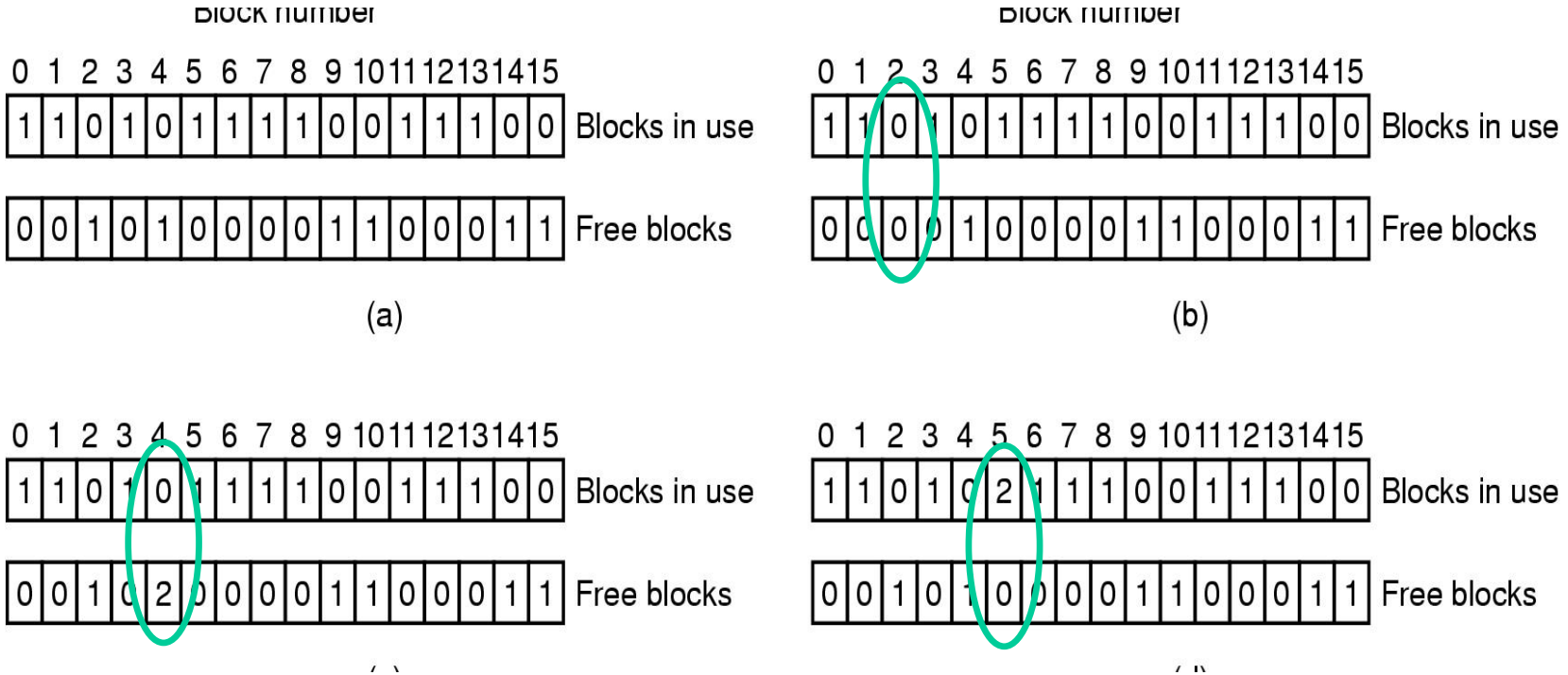
磁盘块不一致的例子

图（a）：一致状态

图（b）：丢失块2: add it to free list

图（c）：块4在空闲表中重复出现: rebuild free list

图（d）：数据块5重复出现: copy block and add it to one file



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

内容提要

- 文件系统基本概念
- 文件系统实现方法
  - 文件的实现
  - 目录的实现
  - 硬链接和软链接
  - 内存中的文件系统
  - 文件的保护
  - 文件系统的性能
- 文件系统实例分析

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
  - 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(intfildes, intfunction, 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

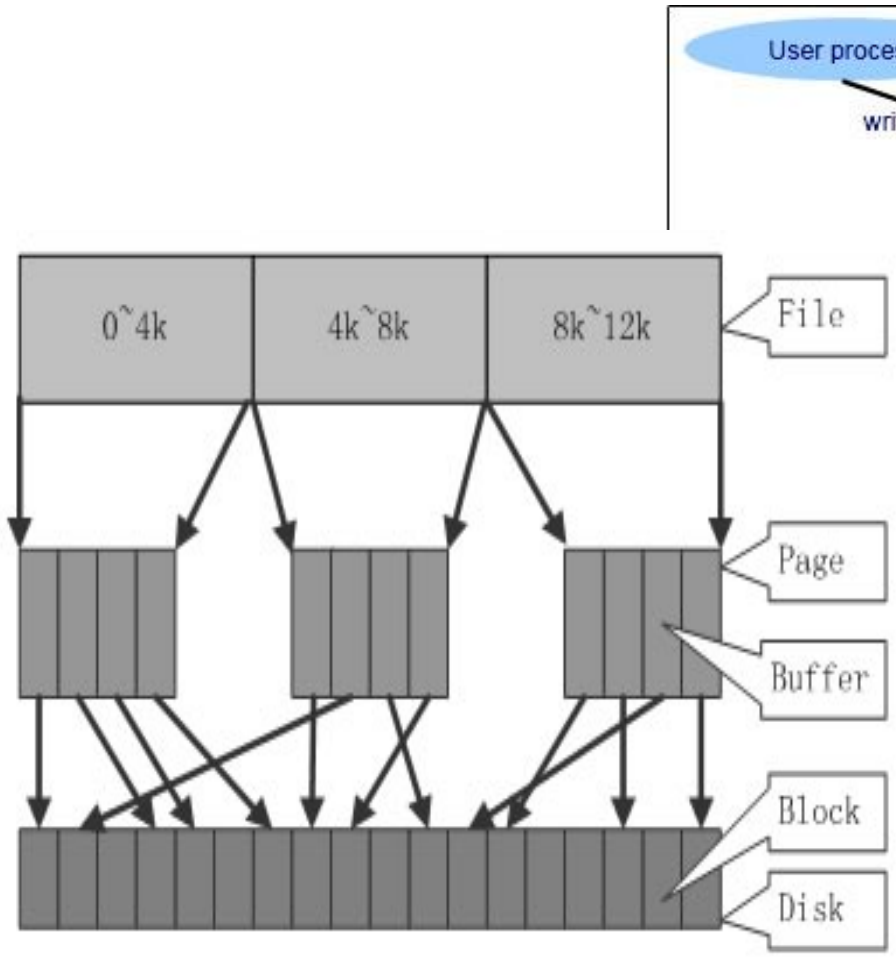
文件系统的性能问题

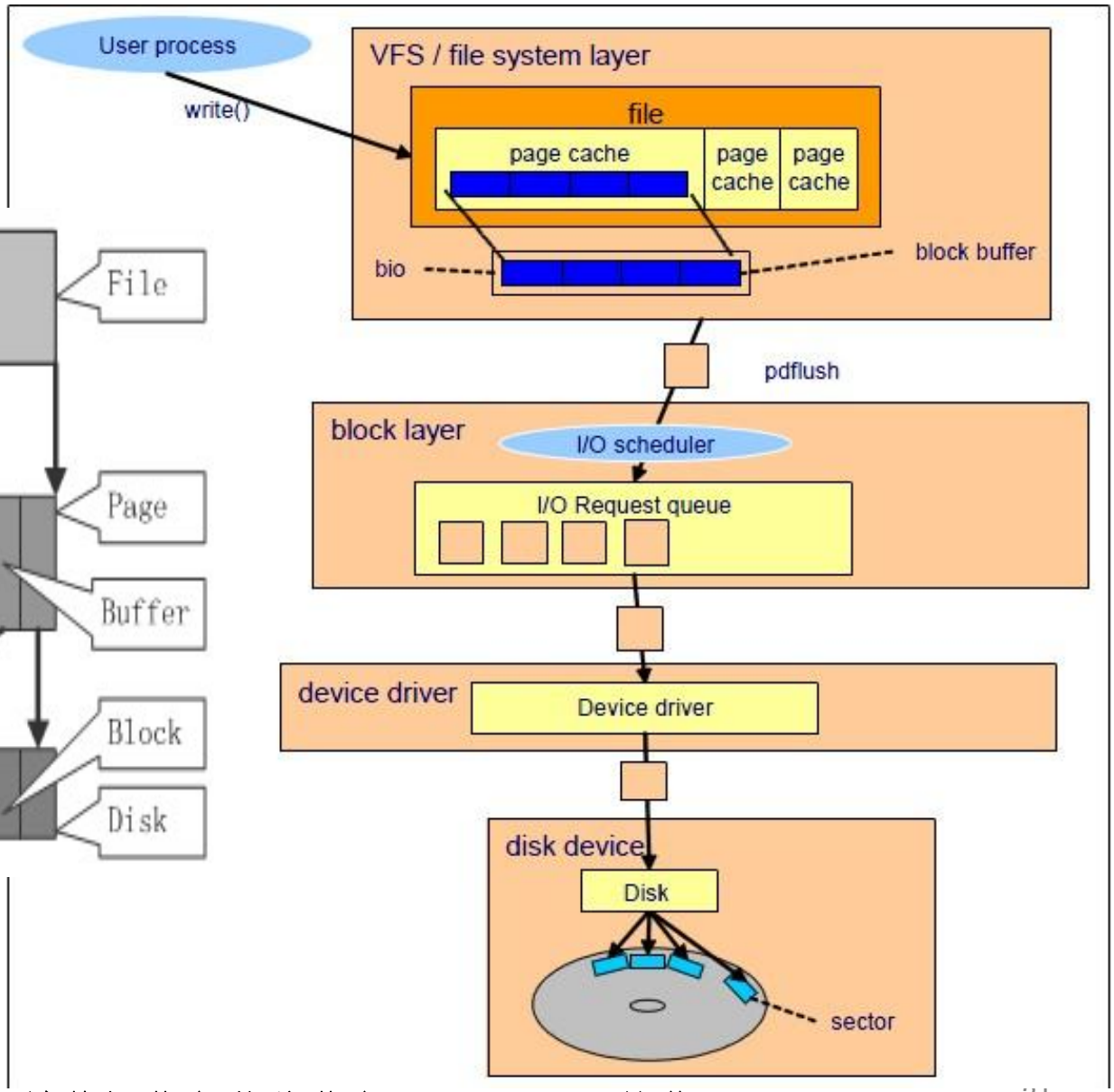
- 磁盘服务：速度成为系统性能的主要瓶颈之一
  - 因此，在设计文件系统时应尽可能减少磁盘访问次数
- 提高文件系统性能的方法：
  - 目录项（FCB）分解、当前目录、磁盘碎片整理
  - 块高速缓存、磁盘调度、提前读取
  - 合理分配磁盘空间、信息的优化分布
  - RAID技术等。

提高文件系统性能：块高速缓存

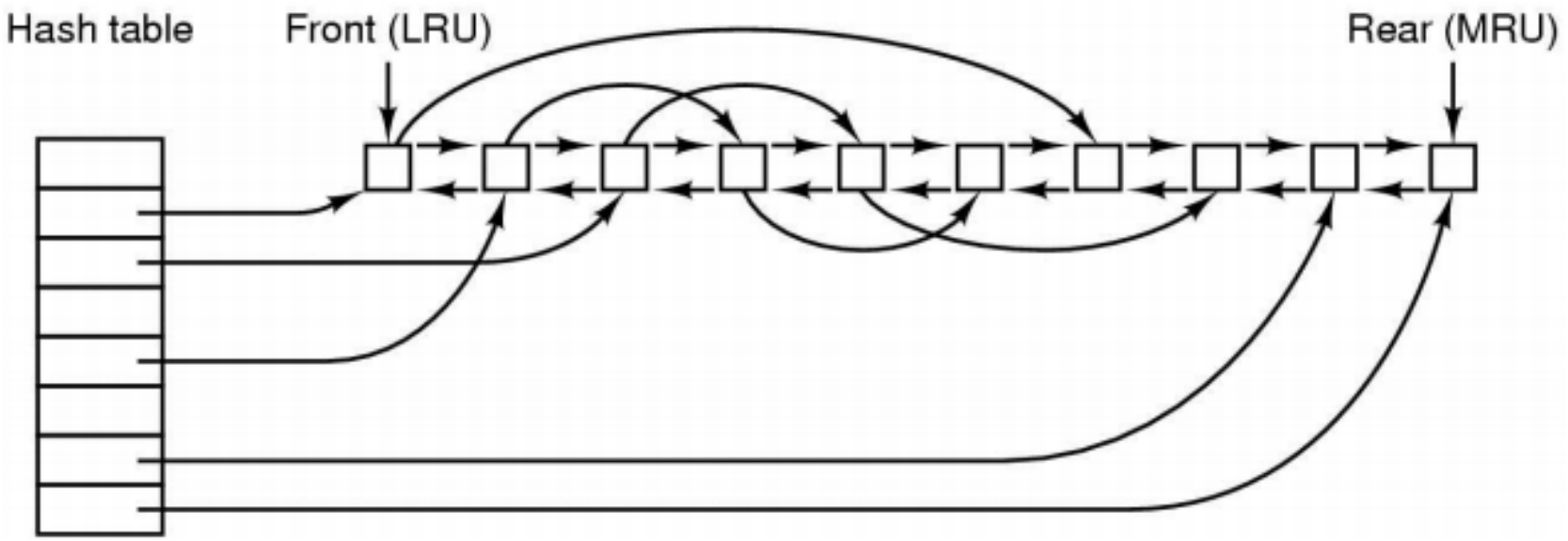
- 块高速缓存（BLOCK CACHE）又称为文件缓存、磁盘高速缓存、缓冲区高速缓存
  - 是指在内存中为磁盘块设置的一个缓冲区，保存了磁盘中某些块的副本
- 当对文件系统进行操作的时候：
  - 检查所有的读请求，看所需块是否在块高速缓冲中
    - 如果在，则可直接进行读操作
    - 否则，先将数据块读入块高速缓存，再拷贝到所需的地方
  - 由于访问的局部性原理
    - 当一数据块被读入块高速缓存以满足一个IO请求时
    - 可能将来还会再次访问到这一数据块

Block Buffer VS Page Cache





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，当其用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

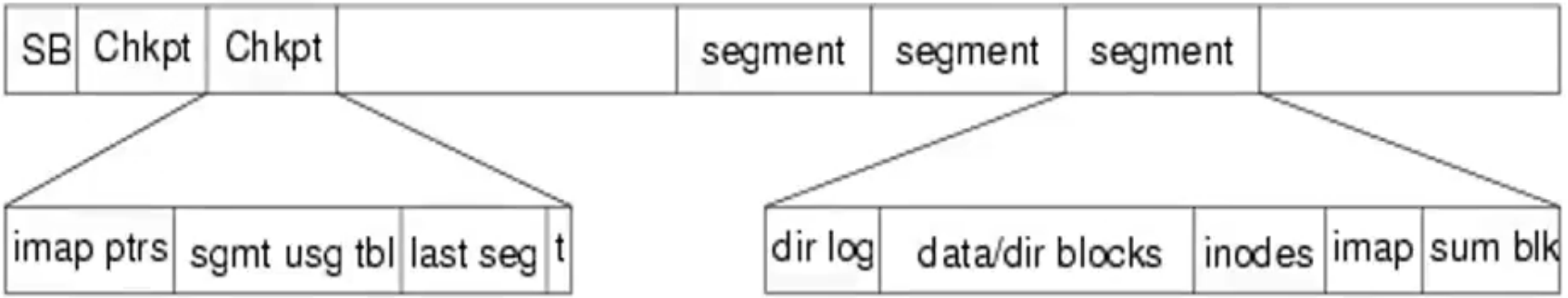
- 大多文件系统都使用缓存来存储最近访问的数据块
  - 大多读操作都可以基于缓存数据来处理，因而无需读取硬盘
- 从磁盘的角度来看，主要以写操作为主
  - 为了提高磁盘的I/O速度，需要提高写数据的速度
- 然而，磁盘的性能主要受到磁头的移动所带来的开销
- 当前文件系统中，写一个数据块需要几次写操作（写数据、修改元数据），因而需要多次移动磁头来进行寻道

LFS: 基本思路

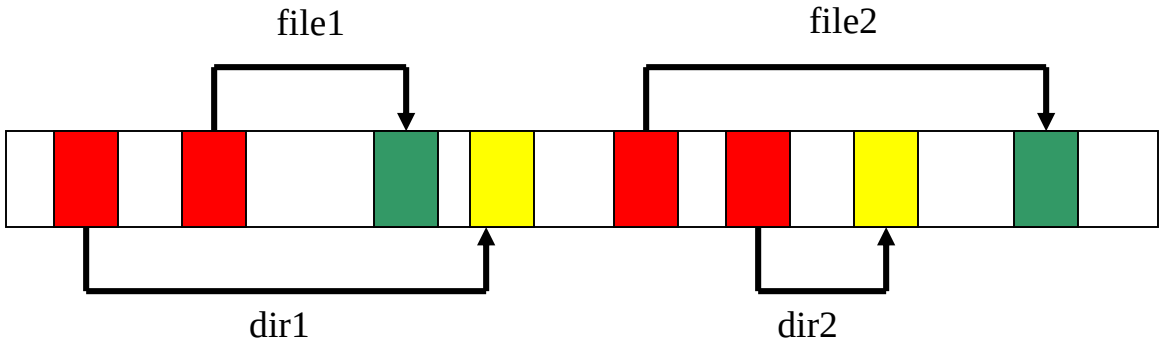
- 将磁盘看作一个日志系统： *log*
  - 日志log是一个数据结构
- 所有的写操作仅在日志头来完成，也就是对日志的追加(append)操作
  - 如果将磁盘按照日志进行管理，就能够避免为寻道而进行的磁头移动
  - “文件”总是顺序地添加到磁盘上
- 新的数据块和元数据 (inode、目录)先放在缓存中，然后一次性写入到硬盘
  - e.g., segments of .5M or 1M
- 这种方式能够有效提高磁盘的吞吐量

LFS的数据结构

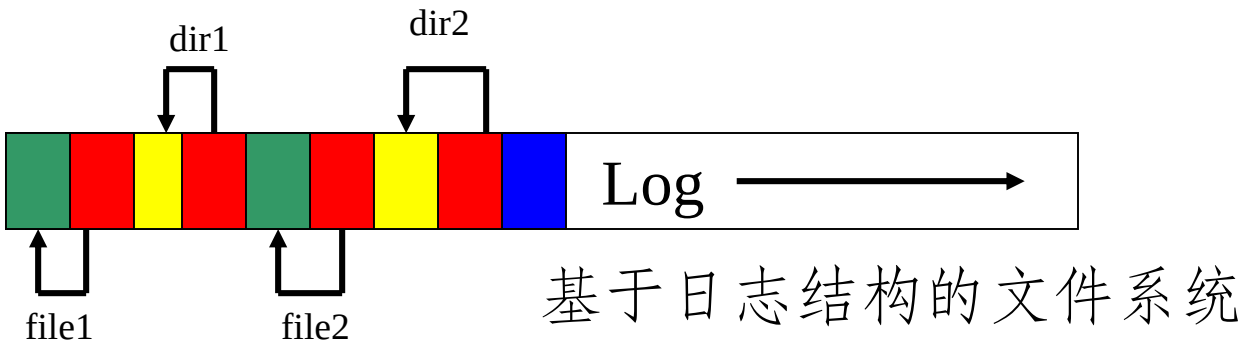
- inode: 与Unix的inode一样，包含文件的物理块指针
- inode map: 一个存放inode节点在磁盘上的位置的表
  - inode map blocks are written as part of the segment
  - a table in a fixed checkpoint region on disk points to those blocks
- 段（Segment）：包含数据块和元数据的日志
- 段摘要: 段中每一个数据块的信息
- 段使用情况表: 段数据块中的有效数据的量



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，也就是把没有填充满的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
  - FAT
  - UFS
  - EXT2
  - VFS
  - 文件系统的注册与挂卸载

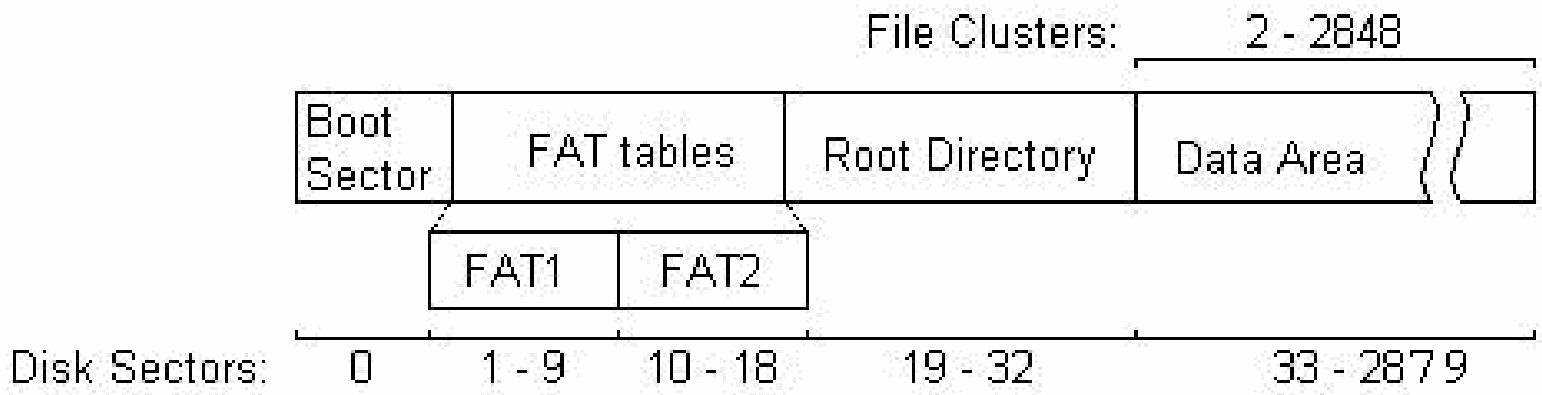
内容提要

- 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
  - UFS
  - EXT2
  - VFS
  - 文件系统的注册与挂卸载

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）

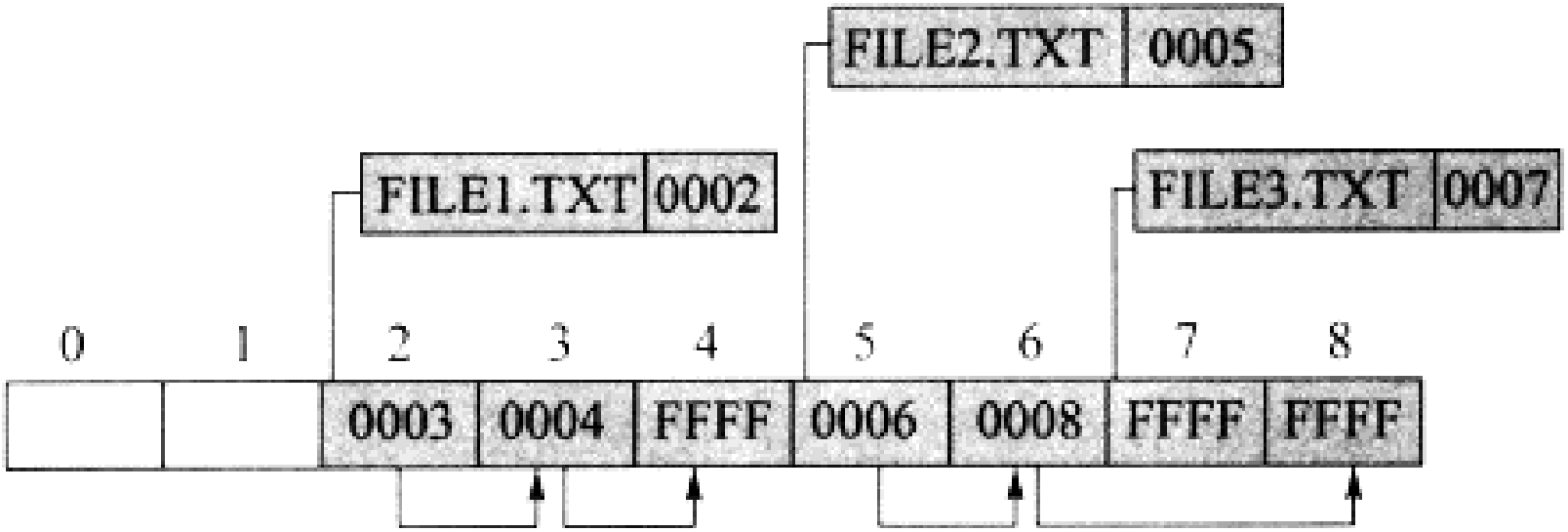
字节偏移量(16进制)	域长	样值(16进制)	含义
0B	2字节	00 02	每扇区字节数
0D	1字节	08	每簇扇区数
0E	2字节	01 00	保留扇区数：从分区引导扇区到第一个文件分配表开始的扇区数
10	1字节	02	文件分配表个数
11	2字节	00 02	根目录项数
13	2字节	00 00	扇区数(小)：卷上的扇区数，如果该数适合于16位(65535)的话
15	1字节	F8	介质类型：F8表明为硬盘，F0表明为软盘
16	2字节	C9 00	每个文件分配表的扇区数(FAT32不用)
18	2字节	3F 00	每磁道扇区数
1A	2字节	10 00	磁头数
1C	4字节	3F 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

- 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。



FAT的簇

- 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

- 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：簇编号位数/(8*512*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

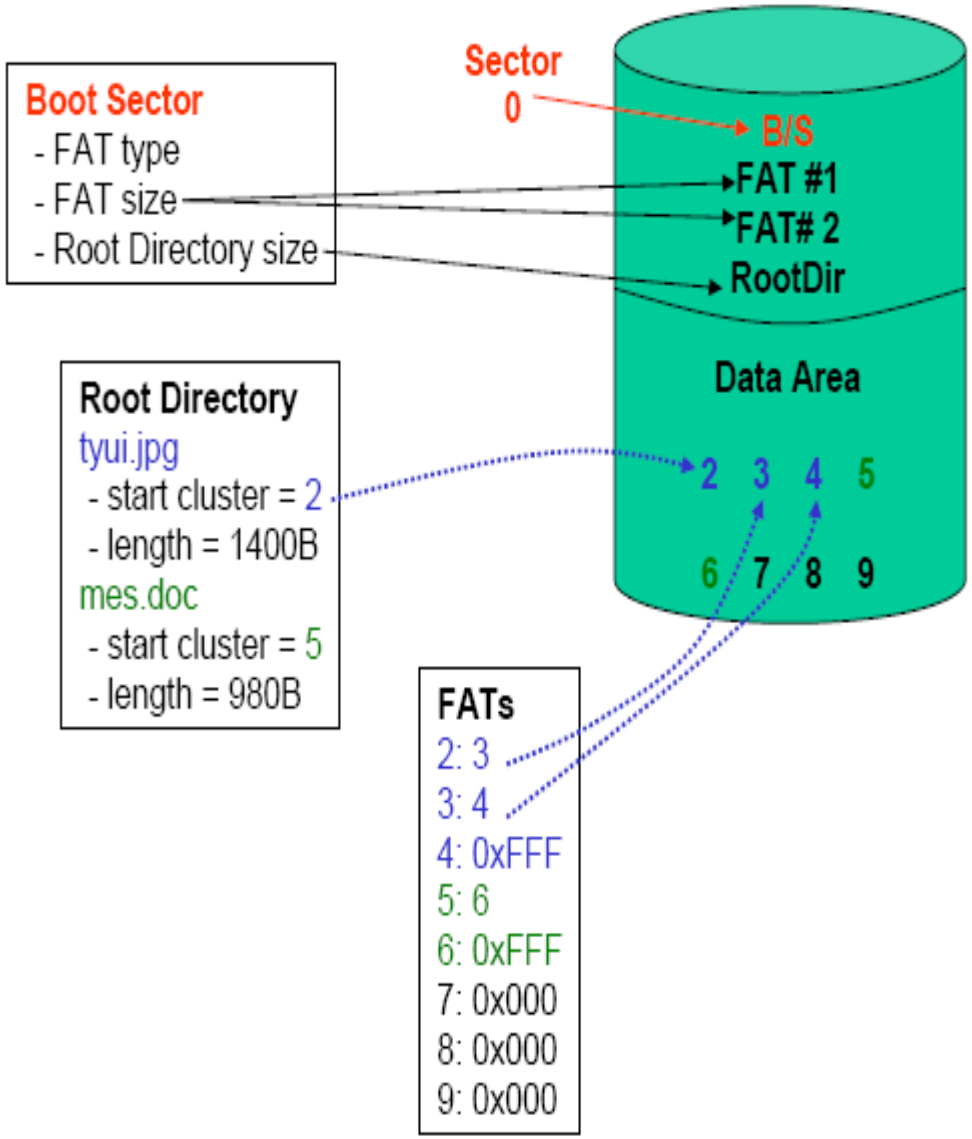
FAT32目录项

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.

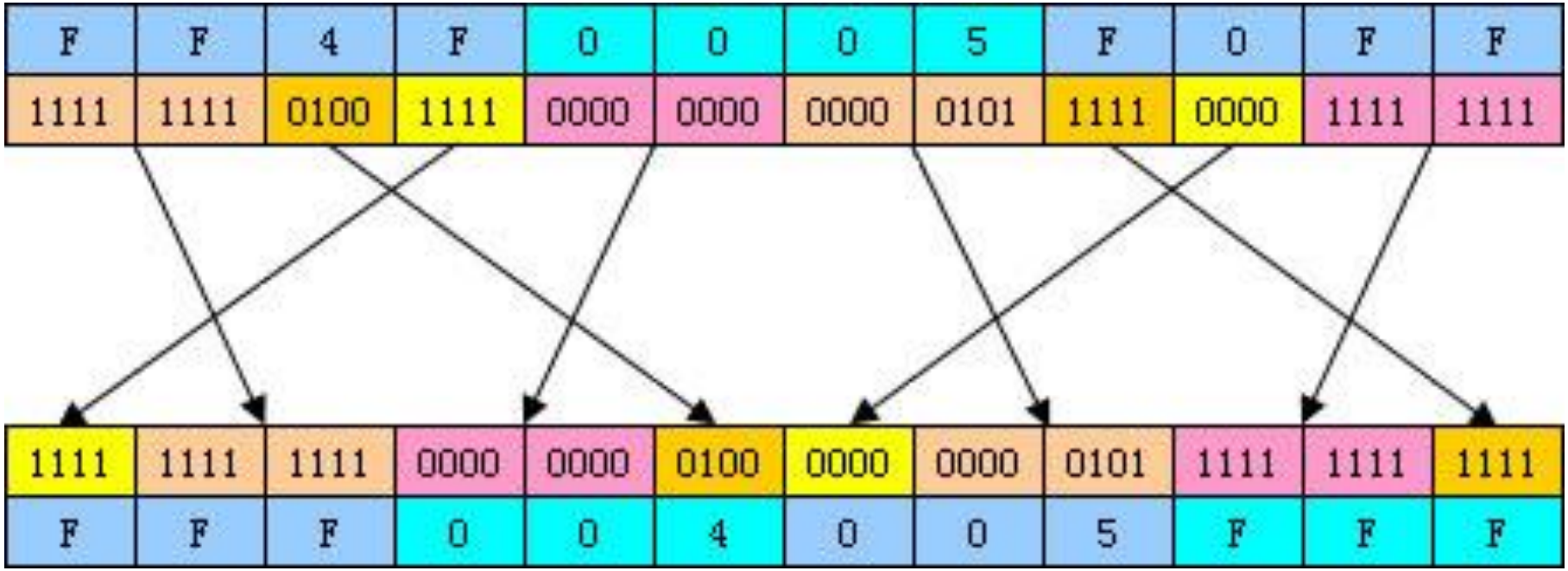


http://www.c-jump.com/CIS24/Slides/FAT/lecture.html

FAT12　FAT举例

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

小端存储格式，低位在前，高位在后



每12位一个块号　　每12位一个块号

小端格式的读取方式

- **xor eax, eax**

- **cmp　byte [bOdd], 1 ；判断是奇数还是偶数簇号**
- **jnz　LABEL_EVEN_2**
- **shr　ax, 4**
- **LABEL_EVEN_2:**
- **and　ax, 0FFFh**

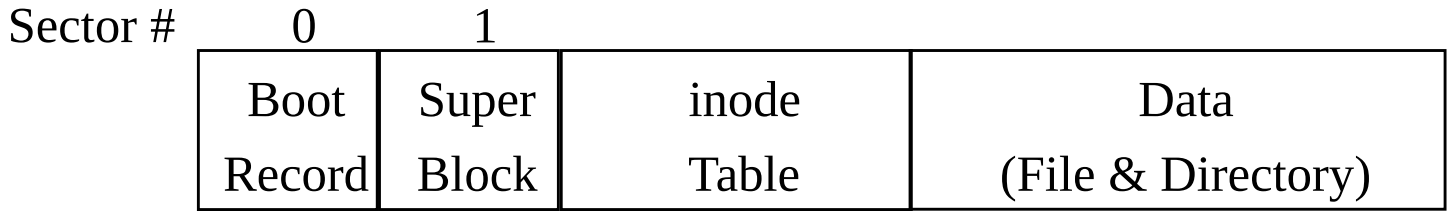
内容提要

- 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
  - UFS
  - EXT2
  - VFS
  - 文件系统的注册与挂卸载

UNIX的文件系统

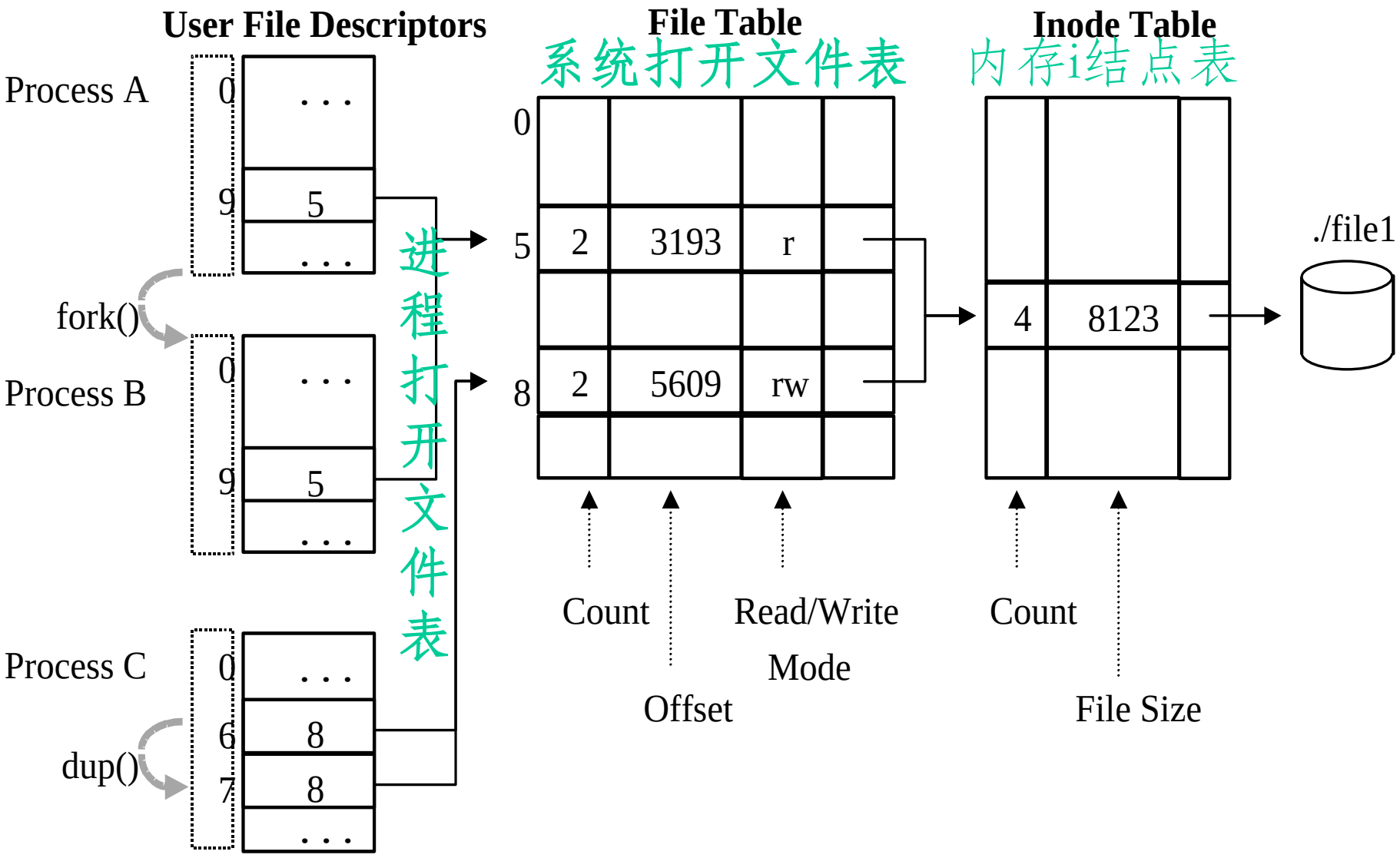
- 多级文件目录，可以建立文件别名（索引结点方式和符号链接方式），有用户访问权限控制（文件的读R、写W和执行X，相应于目录的检索文件、增删文件和进入目录）
  - 注意：如果对文件具有写权限，而对文件所在目录没有写权限，仍然可以改变该文件的长度（如添加数据），因为除文件名外的其他文件目录内容都存放在索引结点而不是在目录文件。
- 文件类型：常规文件(ordinary file)，目录文件(directory)，特殊文件(special file)如外设，先进先出文件(FIFO)如命名管道

UFS



Volume Structure in UNIX

进程和系统打开文件表


User File Descriptors
File Table
系统打开文件表
Inode Table
内存i结点表
Process A
0
9
5
fork()
Process B
0
9
5
进程打开文件表
Process C
0
6
8
dup()
7
8
0
5
2
3193
r
8
2
5609
rw
4
8123
./file1
Count
Read/Write
Mode
Offset
Count
File Size

进程和系统打开文件表

- 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

- 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

- 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

内容提要

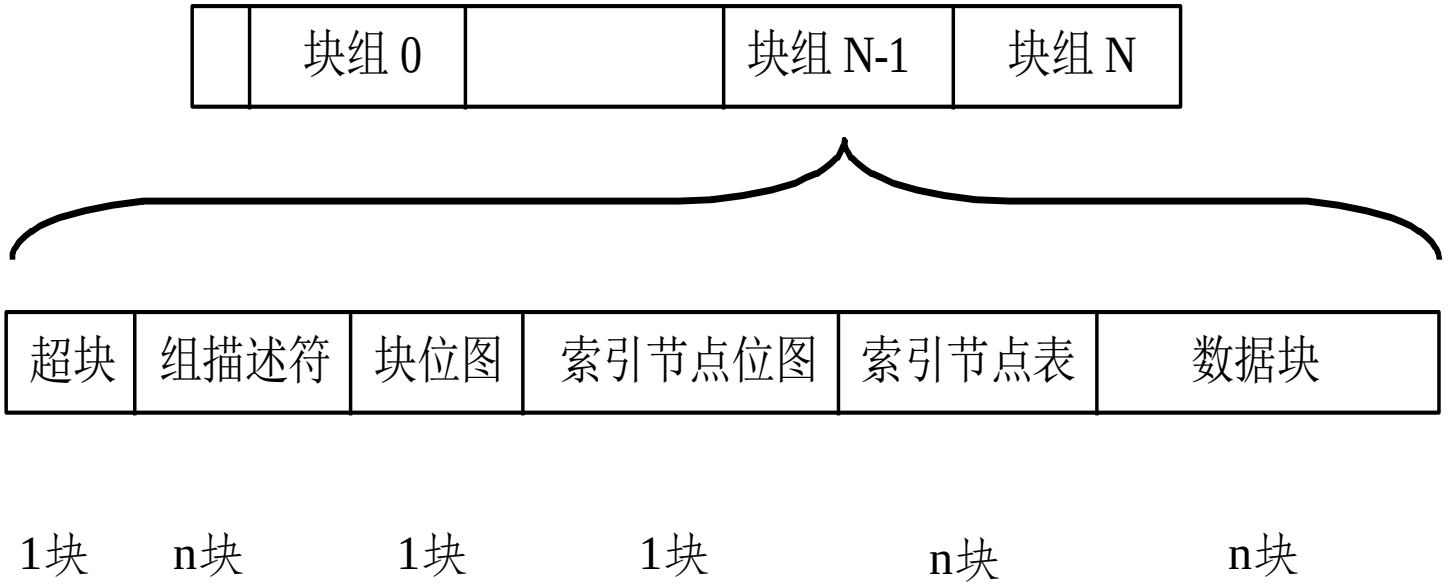
- 文件系统基本概念
- 文件系统实现方法
- 文件系统实例分析
  - FAT
  - UFS
  - EXT2
  - VFS
  - 文件系统的注册与挂卸载

Linux文件系统

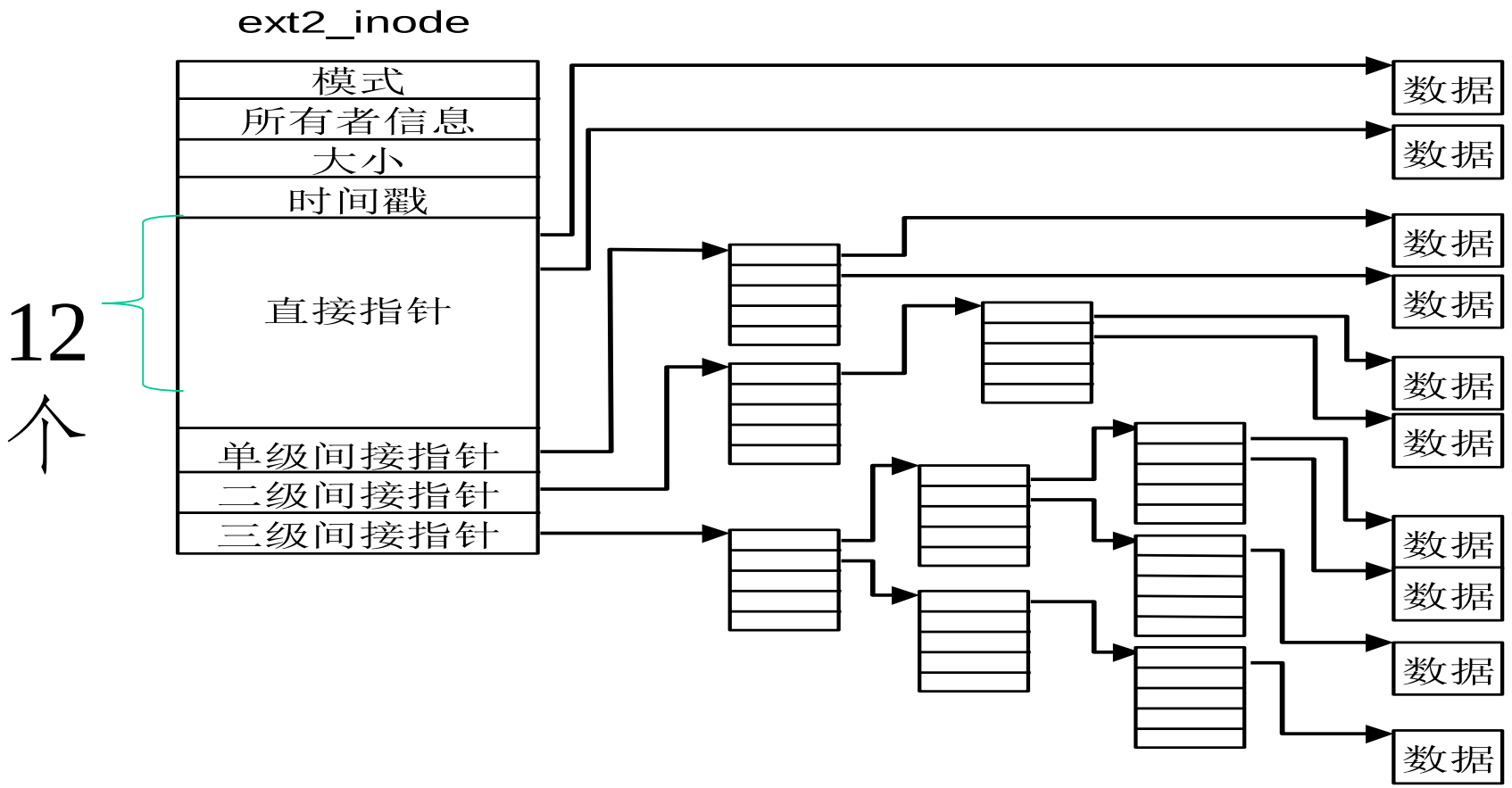
- Ext2 文件系统利用索引节点（inode）来描述文件系统的拓扑结构。在单个文件系统中，每个文件对应一个索引节点，而每个索引节点有一个唯一的整数标识符。文件系统中所有文件的索引节点保存在索引节点表中。
- Ext2 文件系统中的目录实际是一种特殊文件，它们也有对应的索引节点，索引节点指向的数据块中包含该目录中所有的目录项（文件、目录、符号链接等），每个目录项对应自己的索引节点。

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


ext2_inode
模式
所有者信息
大小
时间戳
直接指针
单级间接指针
二级间接指针
三级间接指针
12
个
数据
数据
数据
数据
数据
数据
数据
数据
数据
数据

Inode的解释

- 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。

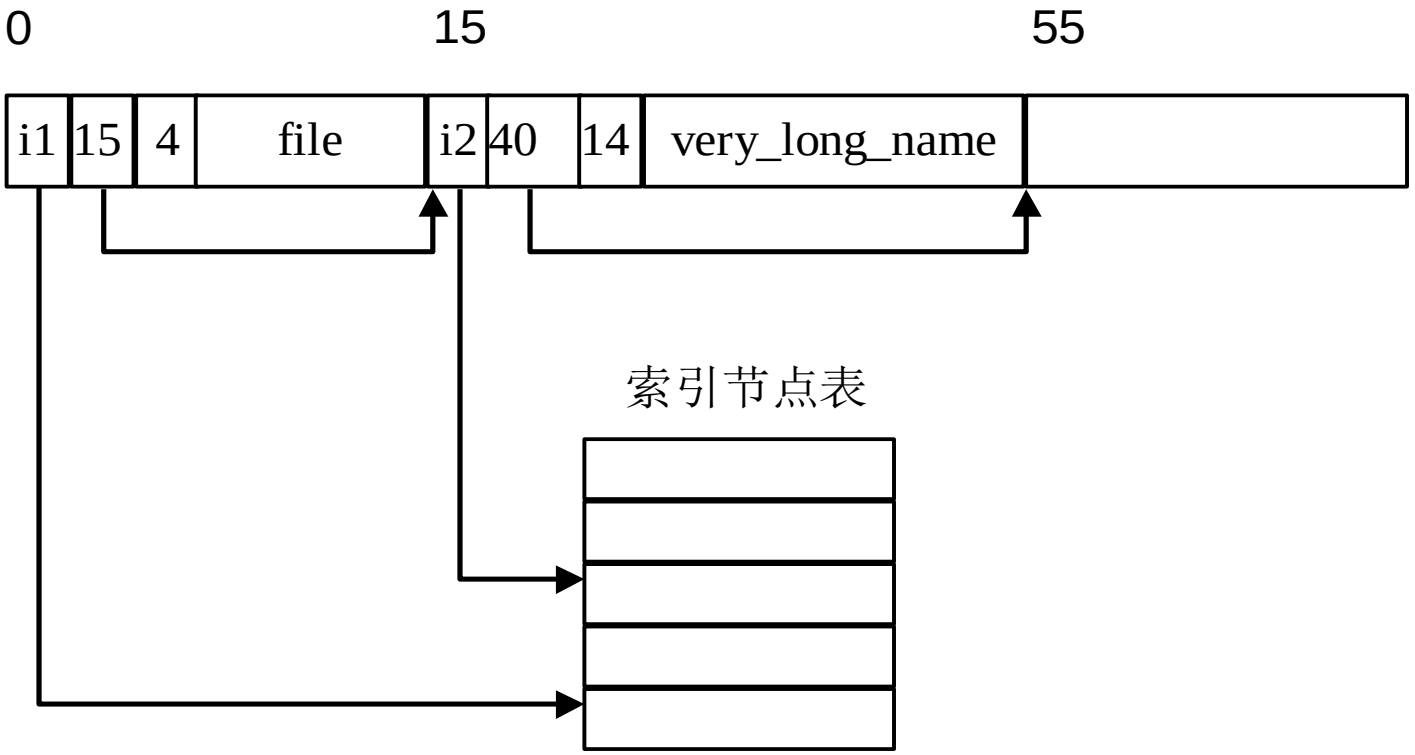
Ext2 文件系统的超块

- 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

Ext2目录项



访问/usr/WeiYM/mbox文件过程

Root目录总是位于1号索引节点

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
  - UFS
  - EXT2
  - VFS
  - 文件系统的注册与挂卸载

虚拟文件系统VFS

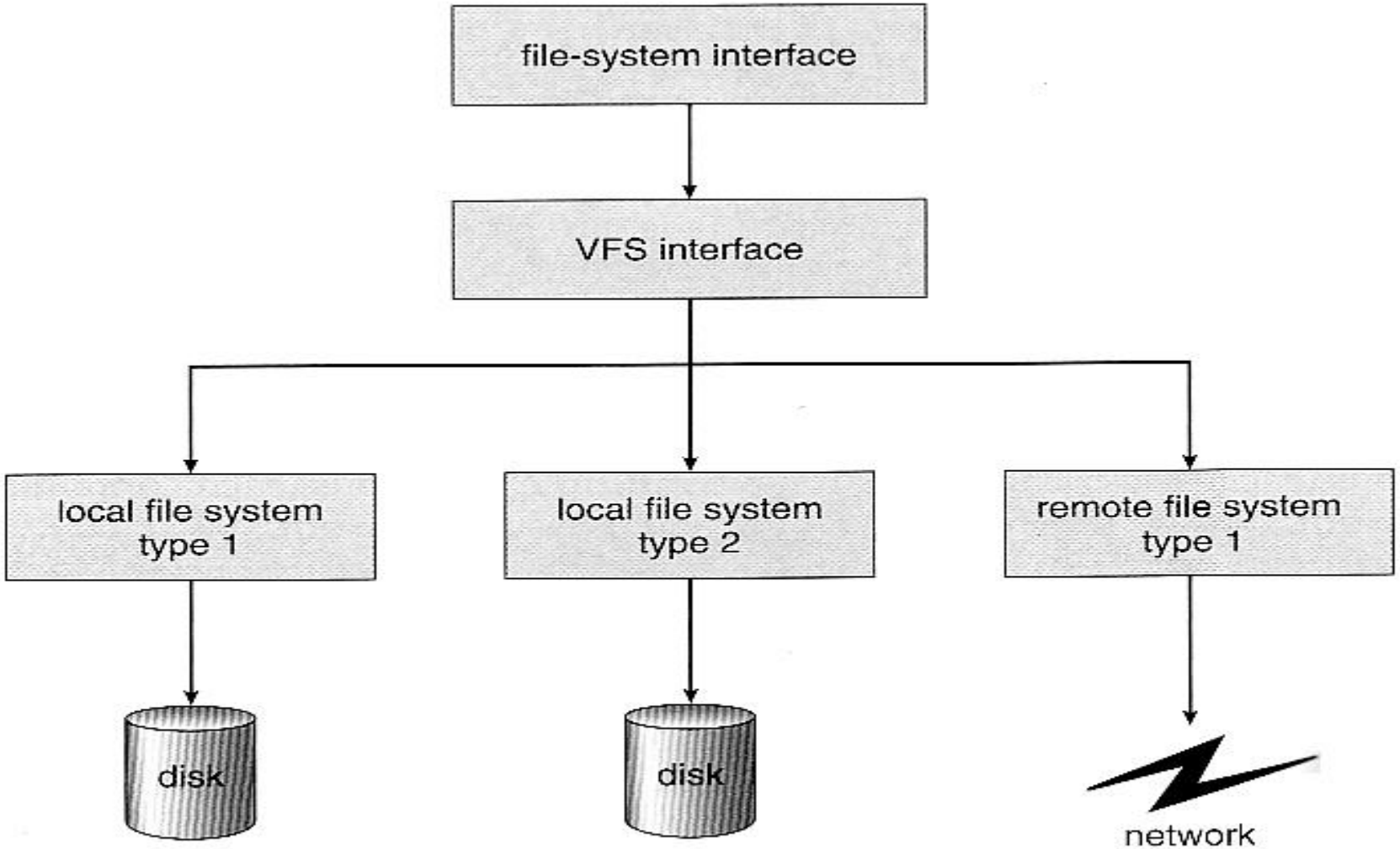
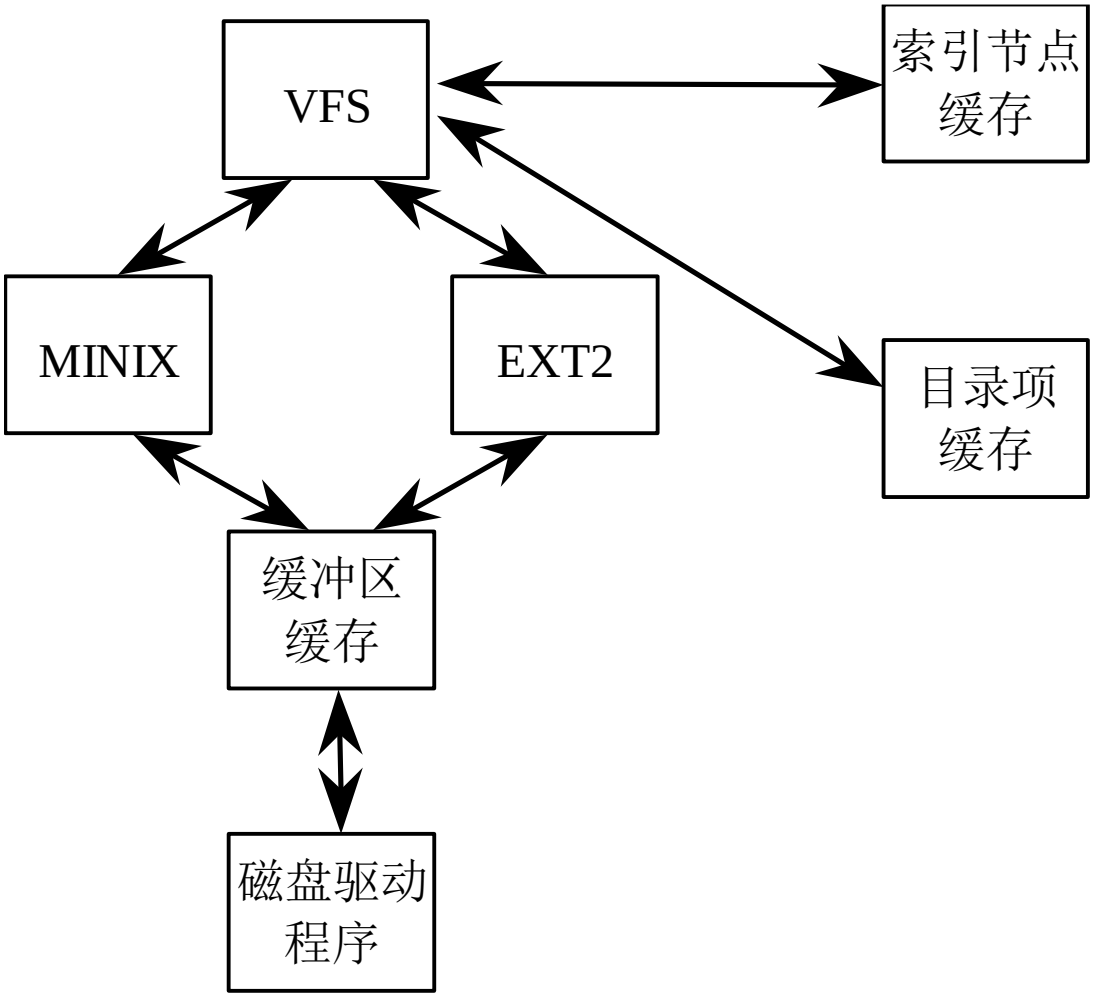
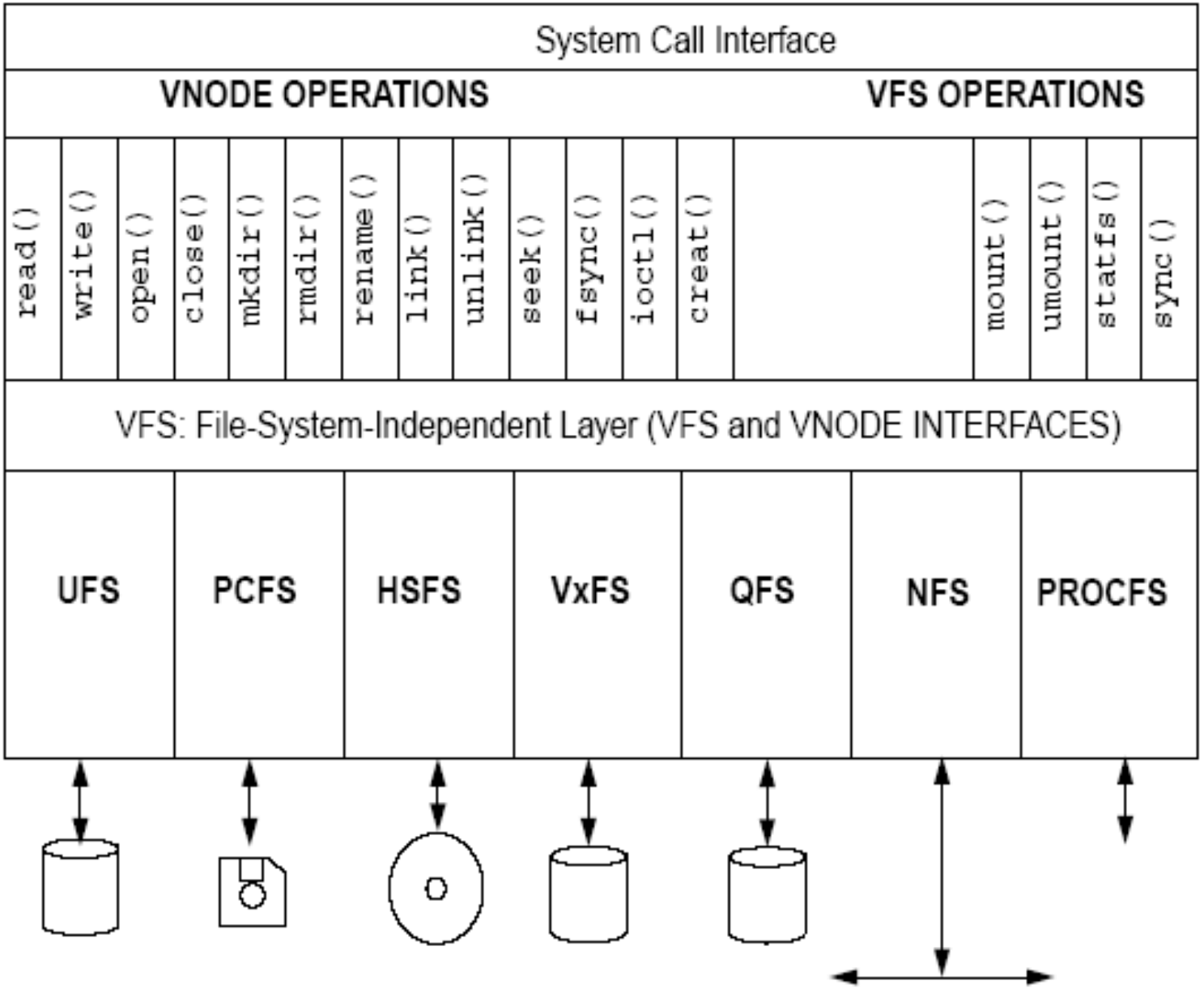


Figure 11.4 Schematic view of a virtual file system.

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。

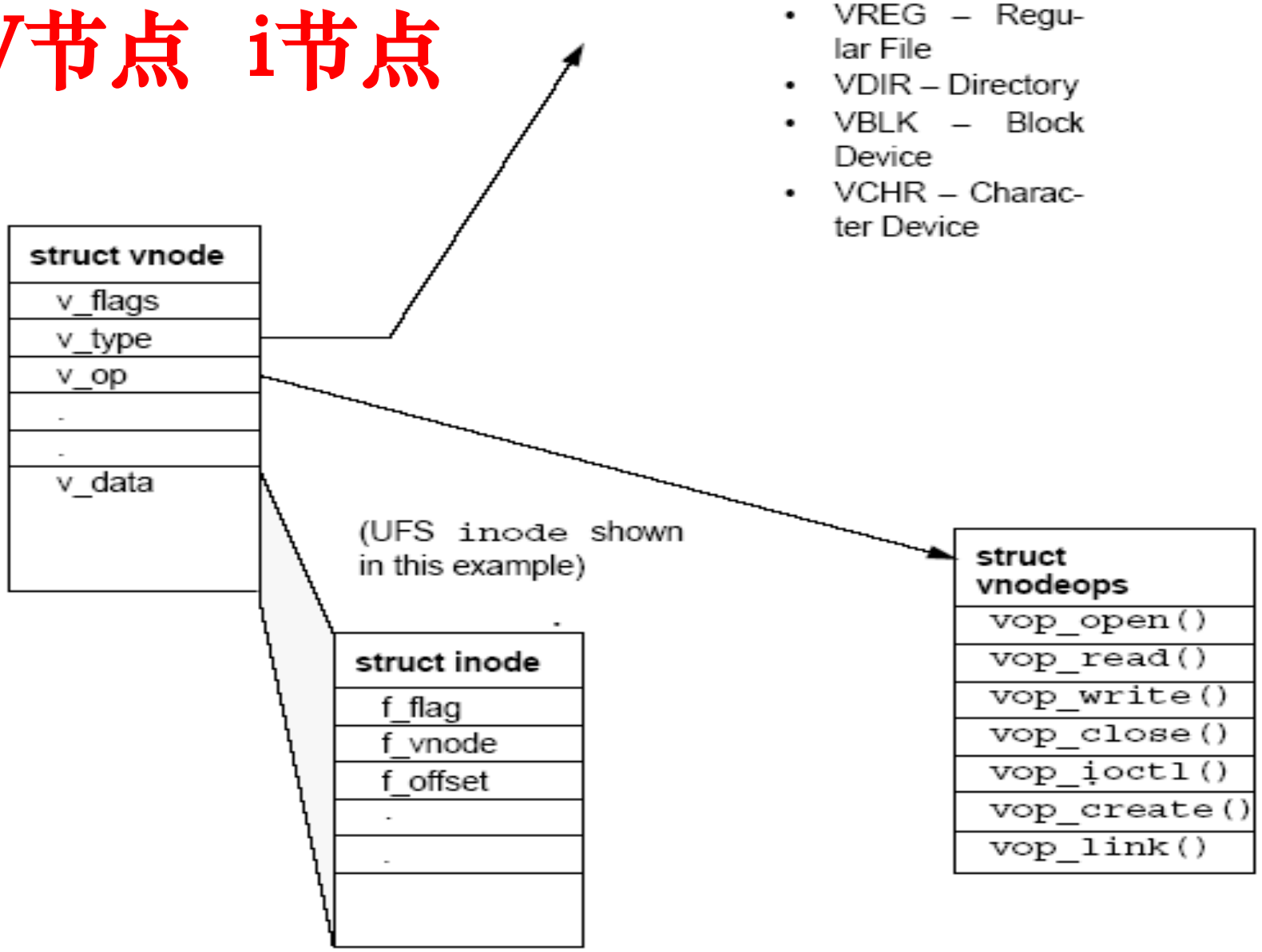
VFS 索引节点

- 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。

VFS 索引节点

- 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。

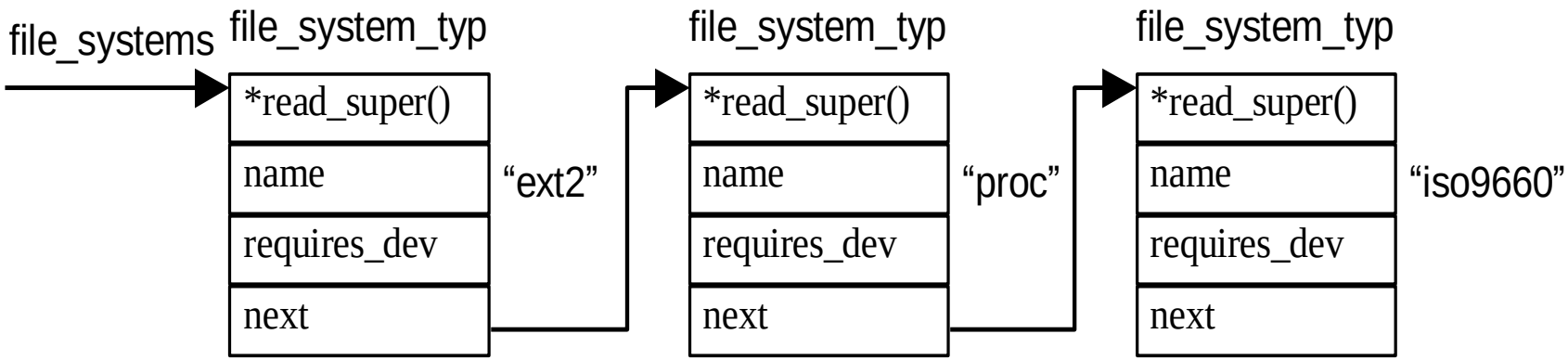
V节点 i节点


• VREG – Regular File
• VDIR – Directory
• VBLK – Block Device
• VCHR – Character Device
struct vnode
v_flags
v_type
v_op
.
.
v_data
(UFS inode shown in this example)
struct inode
f_flag
f_vnode
f_offset
.
.
struct vnodeops
vop_open()
vop_read()
vop_write()
vop_close()
vop_ioctl()
vop_create()
vop_link()

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
  - UFS
  - EXT2
  - VFS
  - 文件系统的注册与挂卸载

文件系统的注册


file_systems
file_system_typ
*read_super()
name
requires_dev
next
“ext2”
file_system_typ
*read_super()
name
requires_dev
next
“proc”
file_system_typ
*read_super()
name
requires_dev
next
“iso9660”

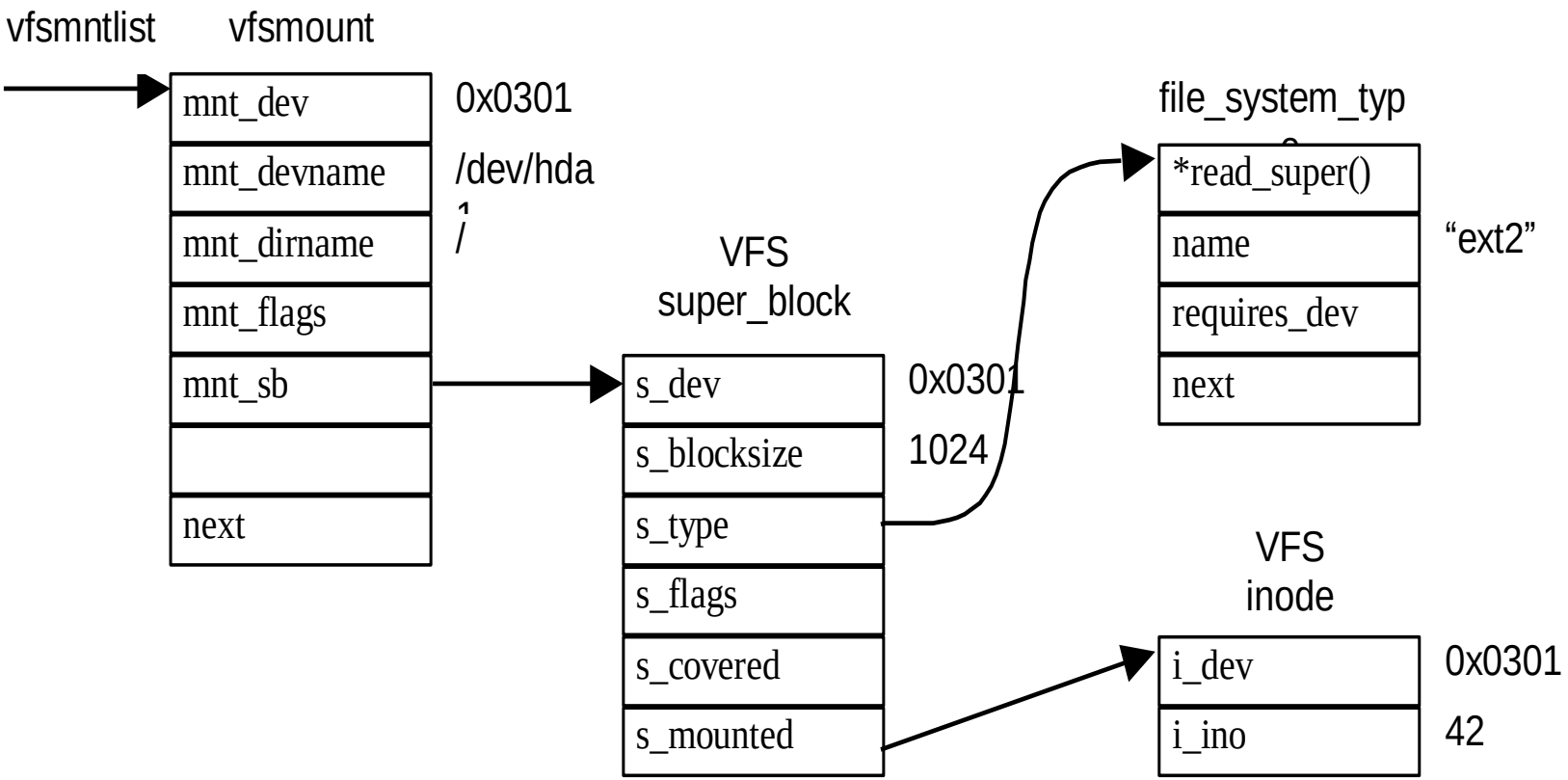
文件系统的挂装和卸装

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。

挂装点与super_block

- 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。

挂装点与super_block



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、…

LINUX已支持的文件系统

- 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

LINUX已支持的文件系统

- 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。

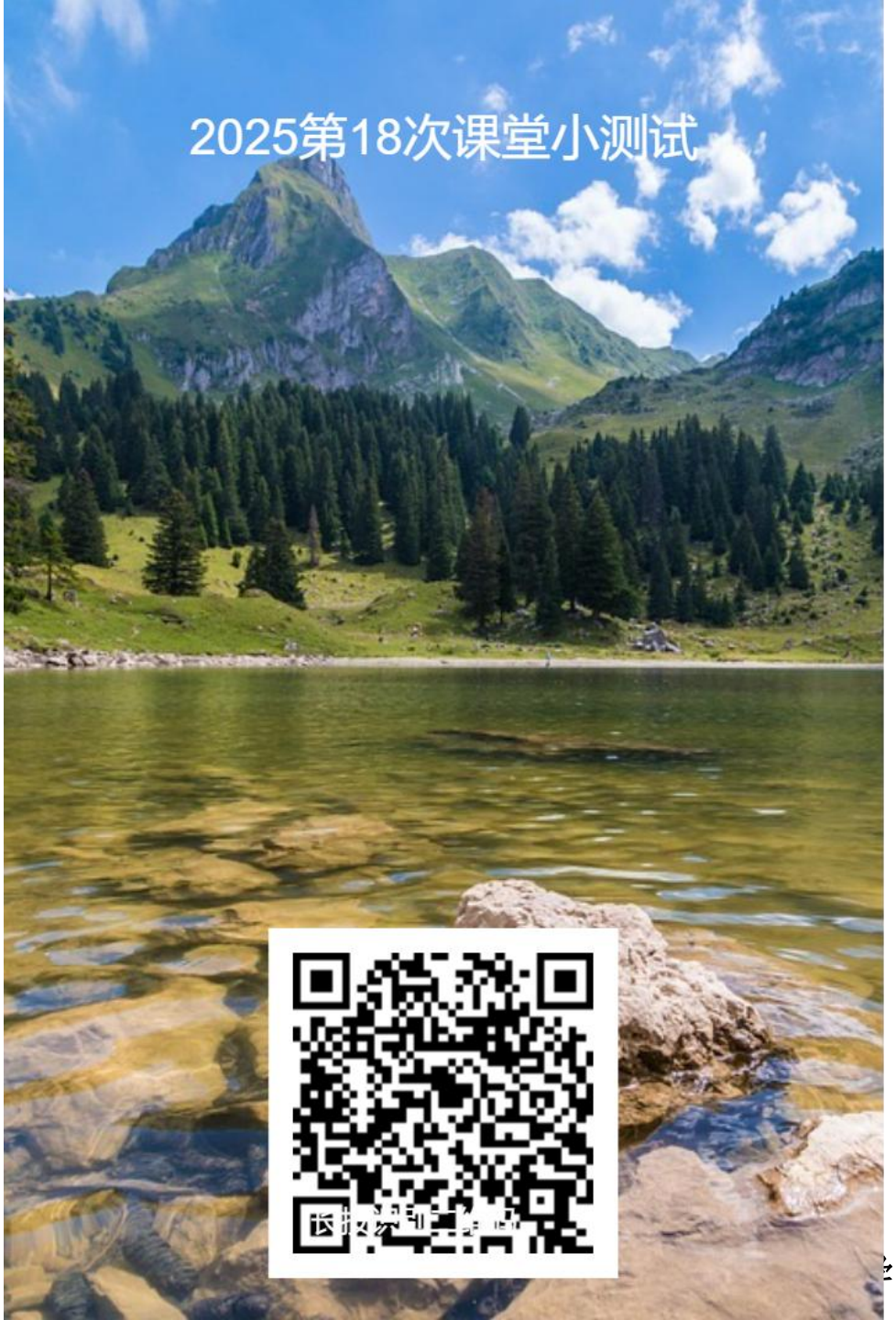
小结

- 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


2025第18次课堂小测试