

操作系统A

Principles of Operating System

北京大学计算机科学技术系 陈向群

Department of computer science
and Technology, Peking University

2020 Autumn

文件管理

- 文件和文件目录
- 文件系统的实现
- 文件系统实例
- 文件系统的管理
- 文件系统的性能
- 文件系统结构
- Windows NTFS 简介

文件系统的一致性、文件系统的安全、

文件系统的管理

1. 文件系统的可靠性

可靠性：

抵御和预防各种物理性破坏和人为性破坏的能力

► 坏块问题

► 备份

通过转储操作，形成文件或文件系统的多个副本

文件系统备份

全量转储：

定期将所有文件拷贝到后援存储器

增量转储：

只转储修改过的文件，即两次备份之间的修改，减少系统开销

物理转储：

从磁盘第0块开始，将所有磁盘块按序输出到磁带

逻辑转储：

从一个或几个指定目录开始，递归地转储自给定日期后所有更改的文件和目录

2. 文件系统一致性(1/2)

问题的产生:

磁盘块 → 内存 → 写回磁盘块

若在写回之前，系统崩溃，则文件系统出现不一致

解决方案:

设计一个实用程序，当系统再次启动时，运行该程序，**检查磁盘块和目录系统**

例子：**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1：记录了每个磁盘块在文件中出现的次数

表2：记录了每个磁盘块在空闲块表中出现的次数

文件系统一致性(2/2)

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
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

使用中的块

0	0	1	0	1	0	0	0	0	1	1	0	0	0	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

空闲块

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
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

使用中的块

0	0	1	0	2	0	0	0	0	1	1	0	0	0	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

空闲块



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
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

使用中的块

0	0	0	0	1	0	0	0	0	1	1	0	0	0	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

空闲块



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
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

使用中的块

0	0	1	0	1	0	0	0	0	1	1	0	0	0	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

空闲块



文件系统写入方式

考虑文件系统
一致性和
性能

(1) 通写 (write-through)

内存中的修改立即写到磁盘

缺点：性能差

例：FAT文件系统

(2) 延迟写 (lazy-write)

利用回写 (write back) 缓存的方法得到高速
可恢复性差

(3) 可恢复写 (transaction log)

采用事务日志来实现文件系统的写入
既考虑安全性，又考虑速度性能

例：NTFS

3. 文件系统的安全性

安全性

确保未经授权的用户不能存取某些文件

◆ 数据丢失

灾难

硬件或软件故障

人的失误

→ 可通过备份解决

◆ 入侵者

积极的 或 消极的

- 非技术人员的偶然窥视
- 入侵者的窥视
- 明确的偷窃企图
- 商业或军事间谍活动

设计安全时要考虑是哪一类入侵者

文件的保护机制 (1/4)

(1) 文件保护

- ▶ 用于提供安全性、特定的操作系统机制
- ▶ 对拥有权限的用户，应该让其进行相应操作，否则，应禁止
- ▶ 防止其他用户冒充对文件进行操作

实现：

- * 用户身份验证
- * 访问控制

文件数据
不能被随意访问

文件的保护机制(2/4)

(2) 用户身份验证

当用户登录时，检验其身份

(用户是谁，用户拥有什么，用户知道什么)

- ▶ 口令、密码

- ▶ 物理鉴定

磁卡，签名分析

- ▶ 基于生物特征信息的认证

指纹，虹膜，视网膜，人脸识别，声纹，手掌血管，
红外线人脸，步态，笔迹

CAPTCHA测试

文件的保护机制(3/4)

(3) 访问控制

主动控制：访问控制表

- ✓ 每个文件一个
- ✓ 放在内核空间
- ✓ 记录用户ID和访问权限
- ✓ 用户可以是一组用户
- ✓ 文件可以是一组文件

能力表(权限表)

- ✓ 每个用户一个
- ✓ 放在内核空间
- ✓ 记录文件名及访问权限
- ✓ 用户可以是一组用户
- ✓ 文件可以是一组文件

文件的保护机制(4/4)

UNIX的文件保护

审查用户的权限，审查本次操作的合法性
采用文件的二级存取控制

第一级：对访问者的识别
对用户分类：

- ✓ 文件主 (owner)
- ✓ 文件主的同组用户 (group)
- ✓ 其它用户 (other)

第二级：对操作权限的识别
对操作分类：

- ✓ 读操作 (r)
- ✓ 写操作 (w)
- ✓ 执行操作 (x)
- ✓ 不能执行任何操作 (-)

例子：rwx rwx rwx

chmod 711 file1 或 chmod 755 file2

数据恢复技术

► 数据恢复的原理

当磁盘、分区、文件遭到破坏时，其数据未真正被覆盖，只是数据在磁盘上的组织形式被破坏，以至于操作系统或用户不能访问

► 哪些情况下数据不能被恢复？

► 数据恢复包括系统数据恢复和用户数据恢复

► 数据恢复手段：工具和手工

各种提高文件系统性能的方法

文件系统的性能

文件系统的性能问题

磁盘服务 → 其速度和可靠性成为系统性能和可靠性的主要瓶颈

设计文件系统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、磁盘(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... ..

1. 磁盘高速缓存

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——磁盘高速缓存

- 当出现一个对某一特定块的I/O请求时，首先检测以确定该块是否在磁盘高速缓存中
- 如果在，则可直接进行读操作；否则，先要将数据块读到磁盘高速缓存中，再拷贝到所需的地方
- 由于访问的局部性原理，当一数据块被读入磁盘高速缓存以满足一个I/O请求时，很有可能将来还会再访问到这块数据
- 有些系统称为文件缓存、块高速缓存、缓冲区高速缓存

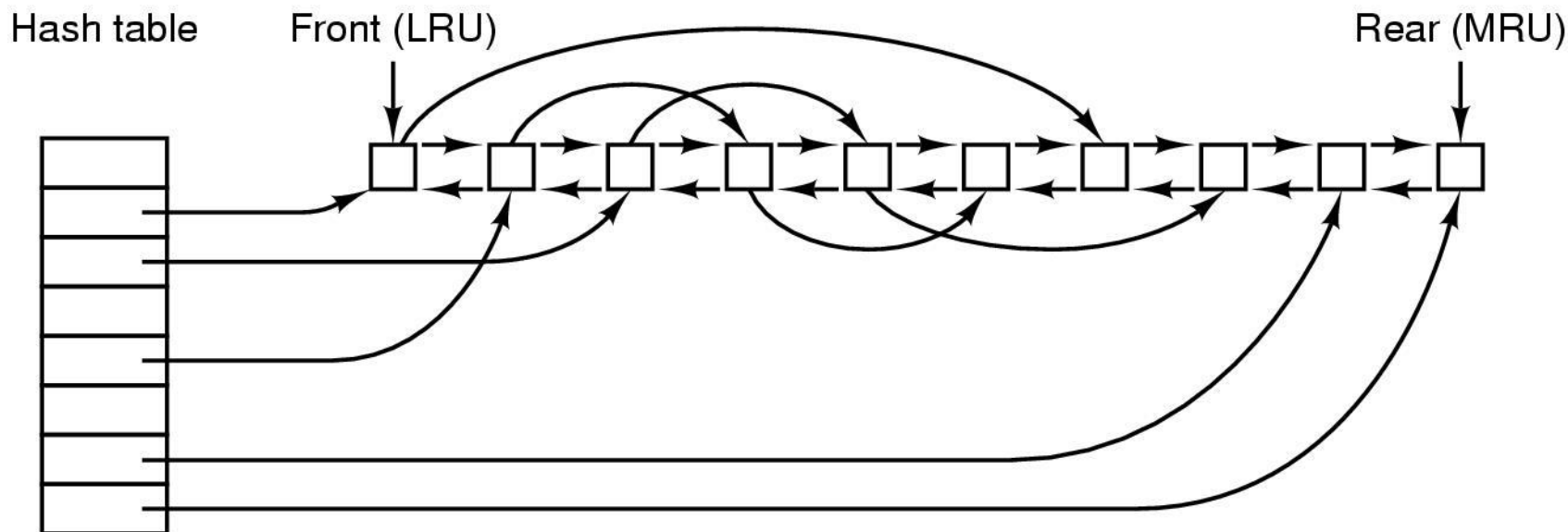
有关问题

块高速缓存满时需要进行置换

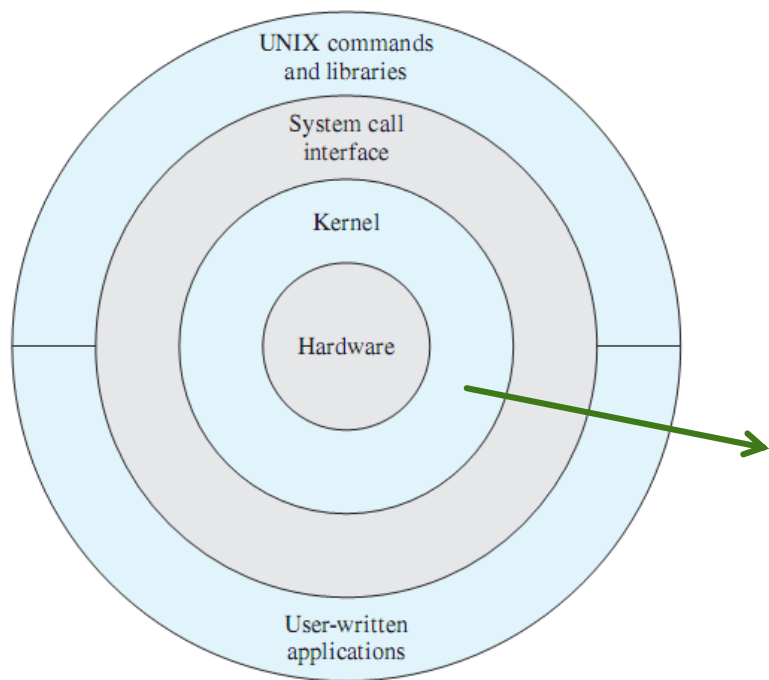
- 块高速缓存的组织
- 块高速缓存的置换（例如LRU）
- 如何考虑文件系统一致性？

该块是否不久后会再次使用

该块是否会影响文件系统的一致性

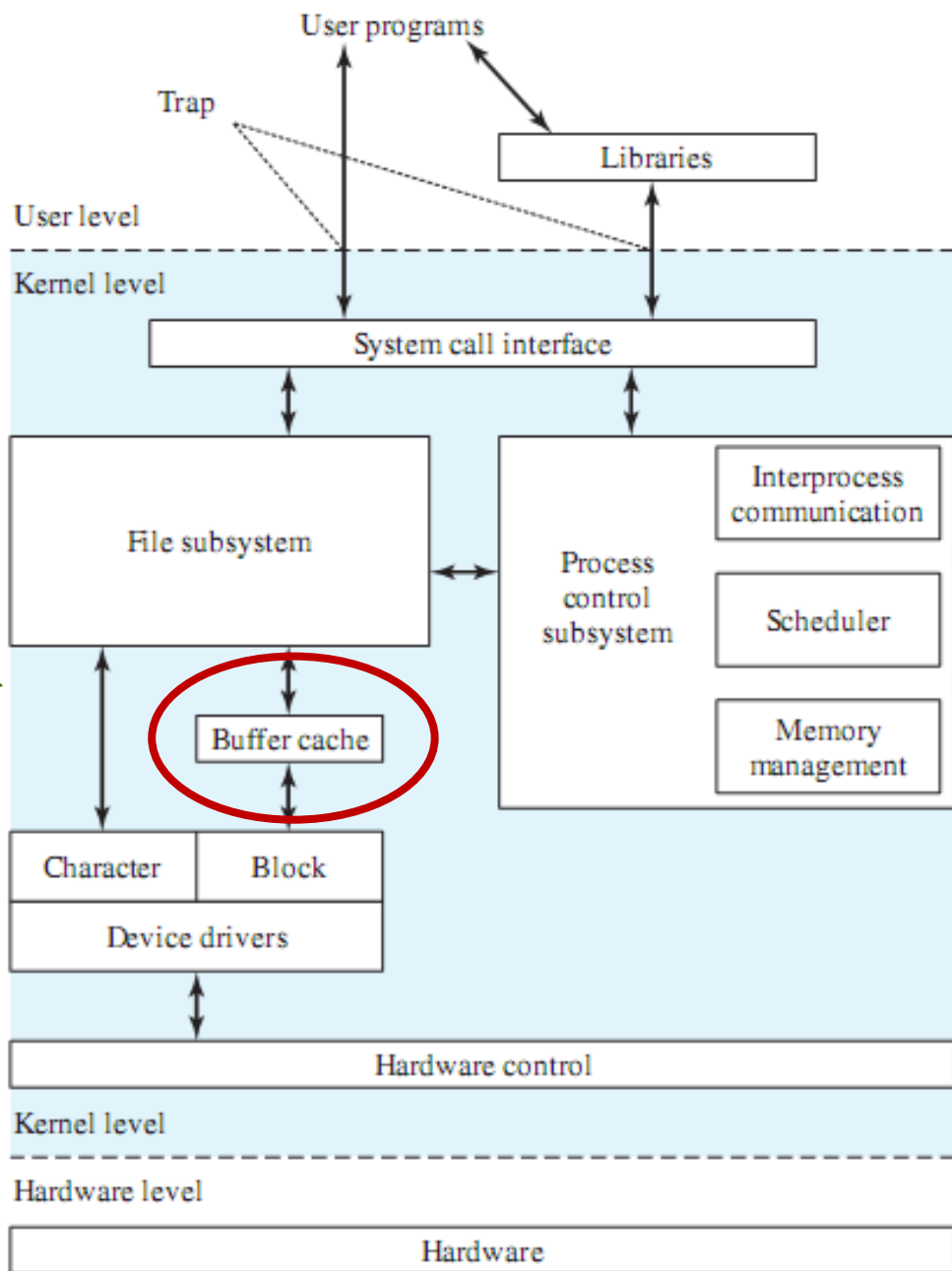


UNIX操作系统



层次结构

内核结构



2. 提前读取

- ▶ 思路：每次访问磁盘，多读入一些磁盘块
- ▶ 依据：程序执行的空间局部性原理
- ▶ 开销：较小(只有数据传输时间)
- ▶ 具有针对性

Windows 的文件访问方式 (1/3)

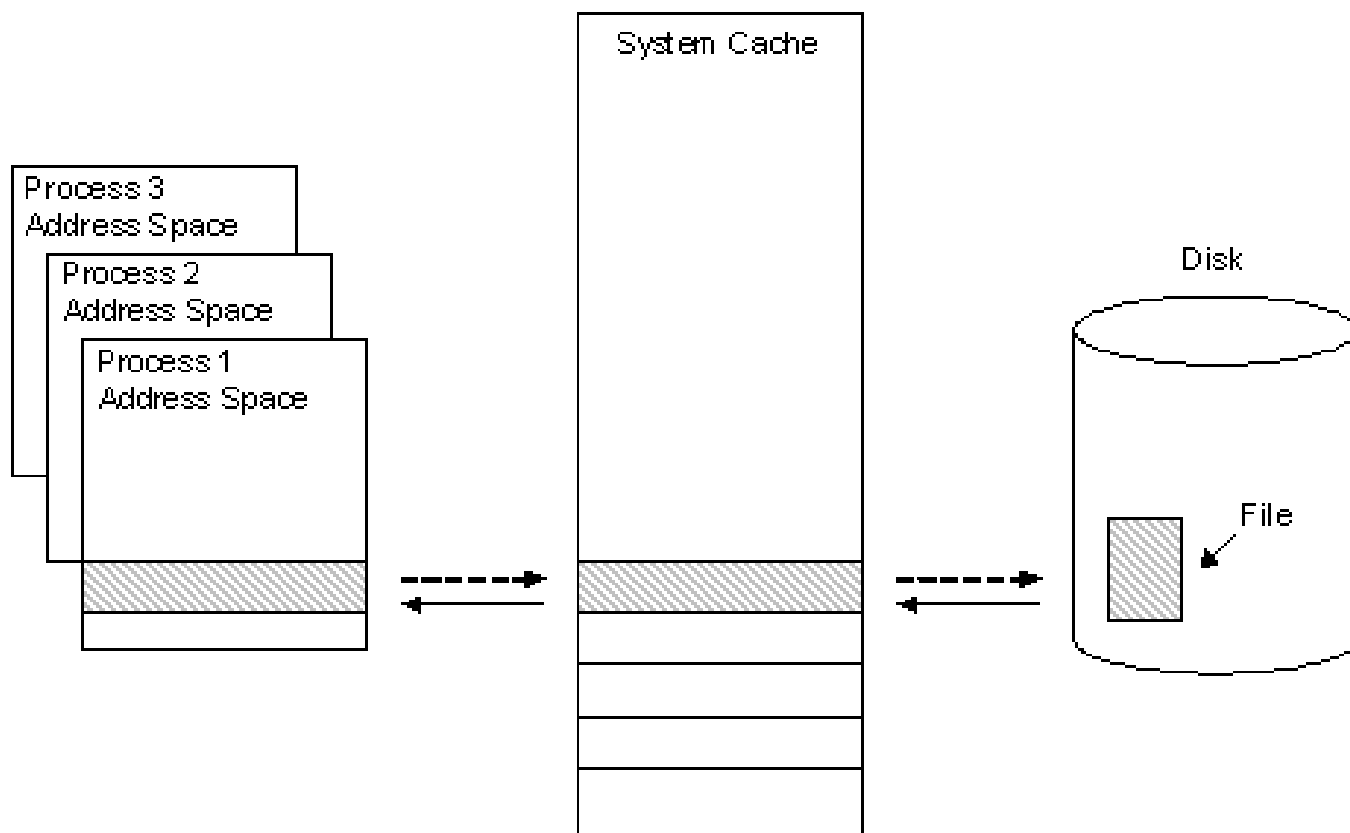
- ▶ 不使用文件缓冲
 - ▶ 普通的方式
 - ▶ 通过Windows提供的FlushFileBuffer函数实现
- ▶ 使用文件缓冲
 - ▶ 预读取。每次读取的块大小、缓冲区大小、置换方式
 - ▶ 写回。写回时机选择、一致性问题
- ▶ 异步模式
 - ▶ 不再等待磁盘操作的完成
 - ▶ 使处理器和I/O并发工作

Windows 的文件访问方式(2/3)

用户对磁盘的访问通过访问文件缓存来实现

- ▶ 由系统的cache manager来实现对缓存的控制
 - ▶ 读取数据的时候预取(prefetch)
 - ▶ 在cache满的情况下，根据LRU原则清除缓存的内容
 - ▶ 定期地更新磁盘上的内容使其与Cache一致（1秒）
- ▶ Write-back机制
 - ▶ 在用户要对磁盘写数据时，只更改Cache中的内容，由Cache Manager来决定何时将更新反映到磁盘

Windows 的文件访问方式 (3/3)



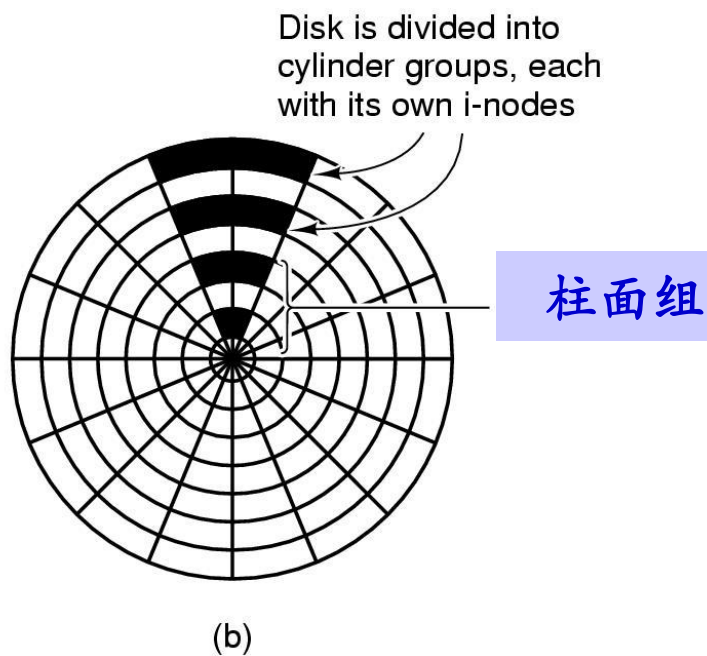
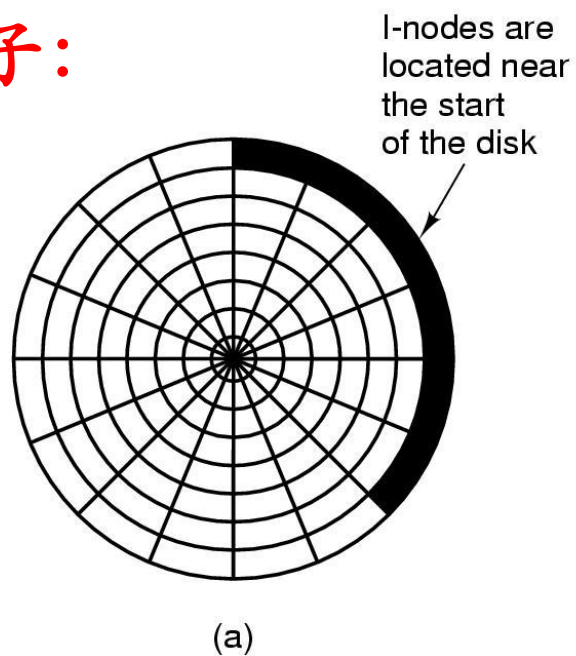
阴影部分为需要访问的数据，因此数据在磁盘、系统cache空间和进程空间有3份拷贝，一般情况下用户对数据的修改并不直接反映到磁盘上，而是通过write-back机制由lazy writer定期地更新到磁盘

3. 合理分配磁盘空间

分配块时，把有可能顺序存取的块放在一起

→ 尽量分配在同一柱面上，从而减少磁盘臂的移动次数

例子：



4. 磁盘调度

当多个访盘请求在等待时，采用一定的策略，对这些请求的服务顺序调整安排

→ 降低平均磁盘服务时间，达到公平、高效

公平：一个I/O请求在有限时间内满足

高效：减少设备机械运动所带来的时间浪费

一次访盘时间 = 寻道时间 + 旋转时间 + 传输时间

- 减少寻道时间
- 减少延迟时间

磁盘调度算法 (1/9)

例子：假设磁盘访问序列：

98, 183, 37, 122, 14, 124, 65, 67

读写头起始位置：53

要求计算：

- (1) 磁头服务序列；
- (2) 磁头移动总距离（道数）

磁盘调度算法 (2/9)

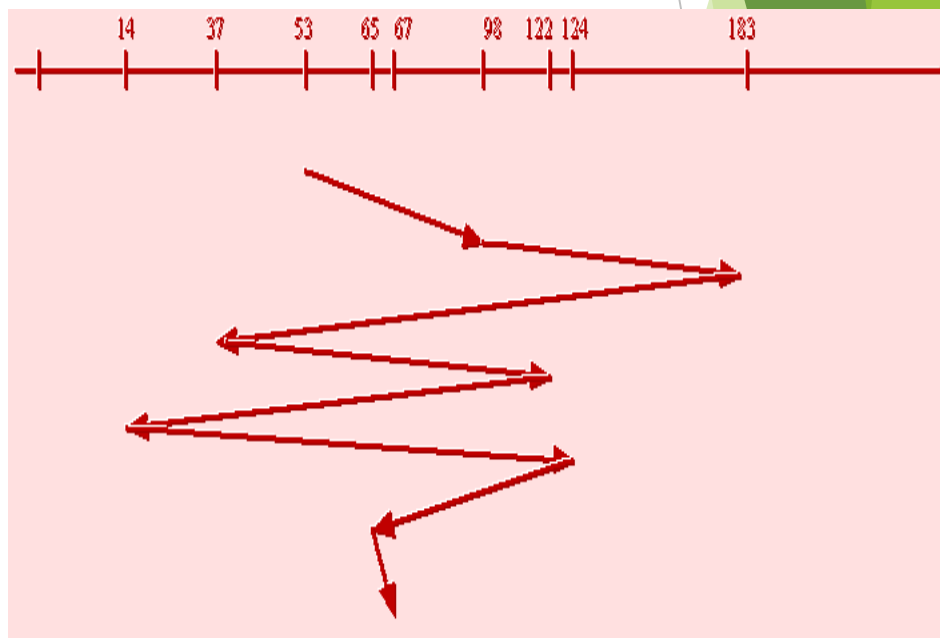
- ▶ 先来先服务：按访问请求到达的先后次序服务
- ▶ 优点：简单，公平；
- ▶ 缺点：效率不高，相临两次请求可能会造成最内到最外的柱面寻道，使磁头反复移动，增加了服务时间，对机械也不利

假设磁盘访问序列：

98, 183, 37, 122, 14, 124,
65, 67

读写头起始位置：53

640 磁道 (平均 80)



磁盘调度算法 (3/9)

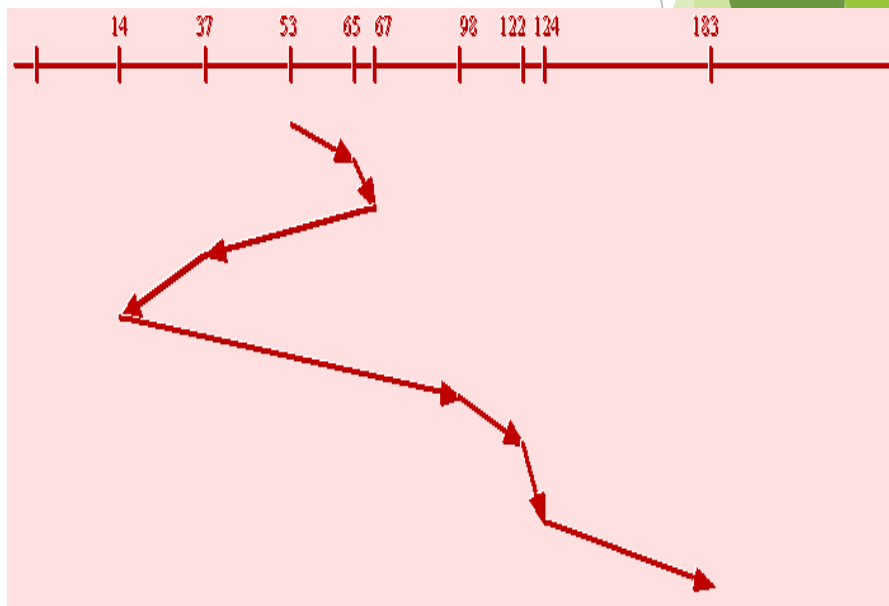
- ▶ 最短寻道时间优先：优先选择距当前磁头最近的访问请求进行服务，主要考虑寻道优先
- ▶ 优点：改善了磁盘平均服务时间；
- ▶ 缺点：造成某些访问请求长期等待得不到服务

假设磁盘访问序列：

98, 183, 37, 122, 14, 124,
65, 67

读写头起始位置：53

236 磁道 (平均 29.5)



磁盘调度算法 (4/9)

折中权衡
距离、方向

► 扫描算法 (SCAN, 电梯算法)

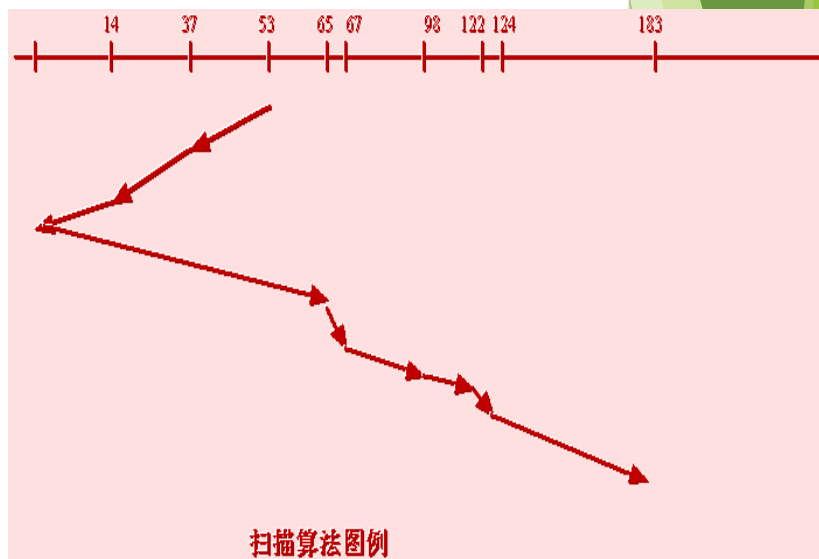
具体做法：当设备无访问请求时，磁头不动；当有访问请求时，磁头按一个方向移动，在移动过程中对遇到的访问请求进行服务，然后判断该方向上是否还有访问请求，如果有则继续扫描；否则改变移动方向，并为经过的访问请求服务，如此反复

假设磁盘访问序列：

98, 183, 37, 122, 14, 124,
65, 67

读写头起始位置：53


218 磁道 (平均 27.25)



磁盘调度算法 (5/9)

► 单向扫描调度算法C-SCAN

- 总是从0号柱面开始向里扫描
- 按照各自所要访问的柱面位置的次序去选择访问者
- 移动臂到达最后个一个柱面后，立即带动读写磁头快速返回到0号柱面
- 返回时不为任何的等待访问者服务
- 返回后可再次进行扫描



减少了新请求
的最大延迟

磁盘调度策略(6/9)

► N-step-SCAN策略

- 把磁盘请求队列分成长度为N的子队列，每一次用SCAN处理一个子队列
- 在处理某一队列时，新请求必须添加到其他某个队列中
- 如果在扫描的最后剩下的请求数小于N，则它们全都将在下一次扫描时处理
- 对于比较大的N值，其性能接近SCAN；当 $N=1$ 时，即FIFO



克服“磁头臂的粘性”

磁盘调度策略(7/9)

► FSCAN策略

- 使用两个子队列
- 扫描开始时，所有请求都在一个队列中，而另一个队列为空
- 扫描过程中，所有新到的请求都被放入另一个队列中
- 对新请求的服务延迟到处理完所有老请求之后



克服“磁头臂的粘性”

磁盘调度算法 (8/9)

► 旋转调度算法

旋转调度：根据延迟时间来决定执行次序的调度

分析：

- 若干等待访问者请求访问同一磁头上的不同扇区
- 若干等待访问者请求访问不同磁头上的不同编号的扇区
- 若干等待访问者请求访问不同磁头上具有相同的扇区

磁盘调度算法(9/9)

► 解决方案:

- ✓ 对于前两种情况: 总是让首先到达读写磁头位置下的扇区先进行传送操作
- ✓ 对于第三种情况: 这些扇区同时到达读写磁头位置下, 可任意选择一个读写磁头进行传送操作

例子:

请求顺序	柱面号	磁头号	扇区号
①	5	4	1
②	5	1	5
③	5	4	5
④	5	2	8

课堂练习

请求顺序	柱面号	磁头号	扇区号
①	9	6	3
②	7	5	6
③	15	20	6
④	9	4	4
⑤	20	9	5
⑥	7	15	2

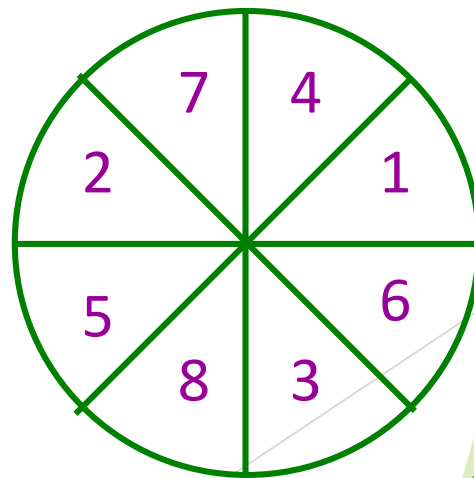
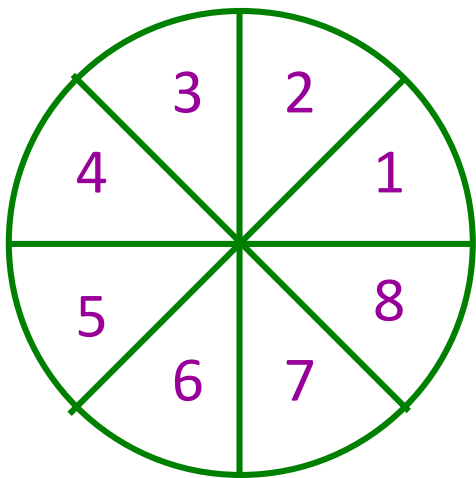
假设磁头在8柱面，求最省时间的响应次序

5. 信息的优化分布

记录在磁道上的排列方式也会影响输入输出操作的时间

例子：

处理程序要求顺序处理8个记录；磁盘旋转一周为20毫秒/周；花5毫秒对记录进行处理



6. 记录的成组与分解

- **记录的成组**：把若干个逻辑记录合成一组存放一块的工作
- 进行成组操作时必须使用内存缓冲区，缓冲区的长度等于逻辑记录长度乘以成组的**块因子**
- **目的**：提高了存储空间的利用率；减少了启动外设的次数，提高系统的工作效率
- **记录的分解**：从一组逻辑记录中把一个逻辑记录分离出来的操作

典型例子——
目录文件

7. RAID 技术

设计时要考虑的是：

磁盘存储系统的 速度、容量、容错、数据灾难发生后的数据恢复

解决方案：RAID（独立磁盘冗余阵列）

(Redundant Arrays of Independent Disks)

多块磁盘按照一定要求构成，操作系统则将它们看成一个独立的存储设备

目标：提高可靠性和性能

美国加州伯克利分校
D.A.Patterson教授1988
年提出

RAID技术的结构

通过把多个磁盘组织在一起，作为一个逻辑卷提供磁盘跨越功能

基本思路

数据是如何组织存储的？

- 1、通过把数据分成多个数据块，**并行**写入/读出多个磁盘，以提高数据传输率（**数据分条stripe**）
- 2、通过**镜像**或**校验**操作，提供容错能力（**冗余**）

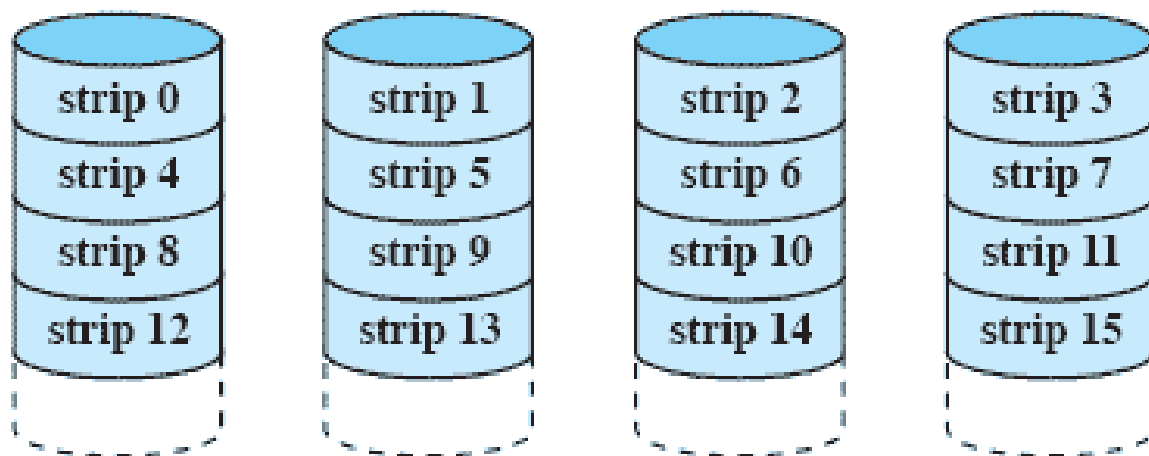
最简单的RAID组织方式：镜像

最复杂的RAID组织方式：块交错校验

条带化、镜像、校验 按“字节”或者“位”

RAID 0 - 条带化

- ▶ 数据分布在阵列的所有磁盘上
- ▶ 有数据请求时，同时多个磁盘并行操作
- ▶ 充分利用总线带宽，数据吞吐率提高，驱动器负载均衡

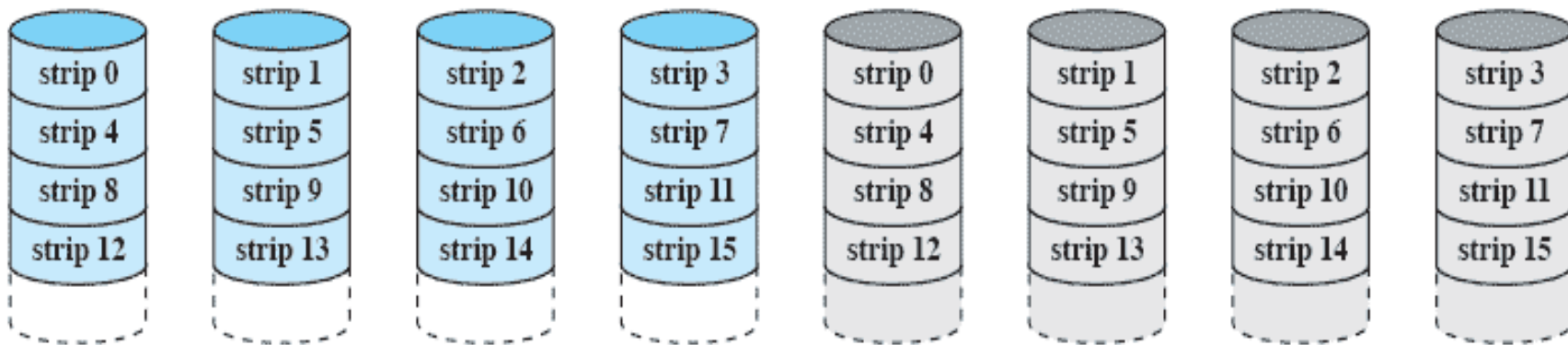


无冗余(即无
差错控制)
性能最佳

RAID 1 - 镜像

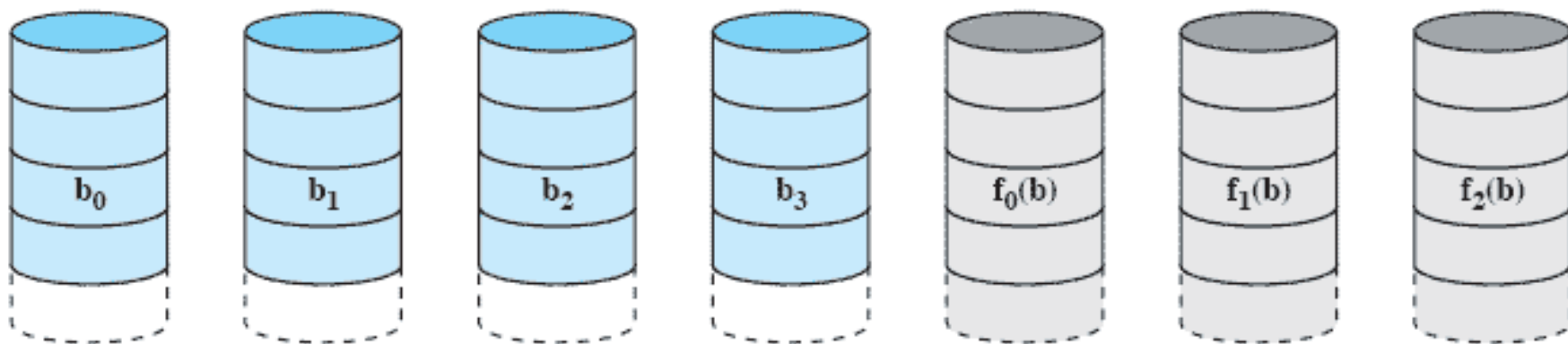
数据安全性
最好

- ▶ 最大限度保证数据安全及可恢复性
- ▶ 所有数据同时存在于两块磁盘的相同位置
- ▶ 磁盘利用率50%



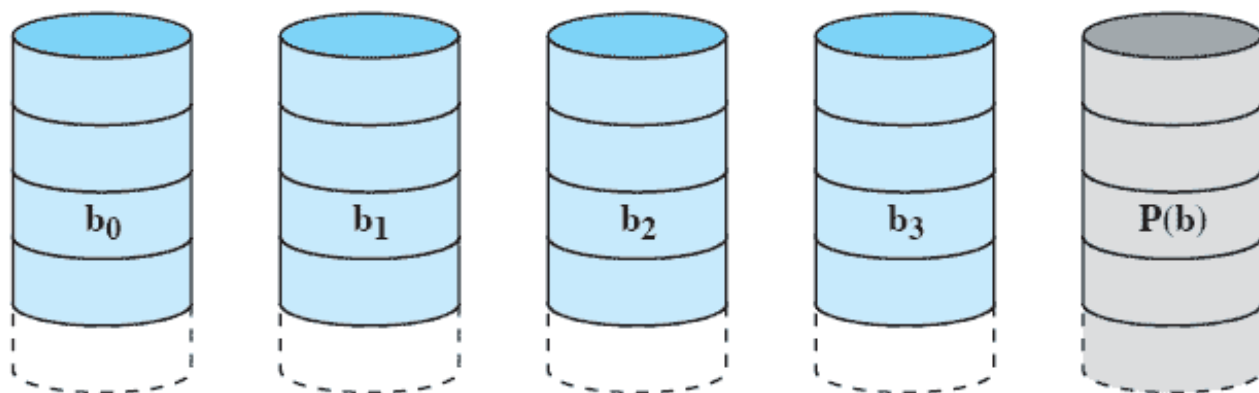
RAD 2 并行访问 — 海明码校验

- ▶ 将数据条块化分布于不同硬盘（字节或位为单位）
- ▶ 加入海明码，在磁盘阵列中间隔写入每个磁盘中
- ▶ 数据发生错误时可实施校正以保证输出正确数据
- ▶ 存取数据时，整个磁盘阵列一起动作，在各个磁盘的相同位置平行存取，所以有很好的存取时间



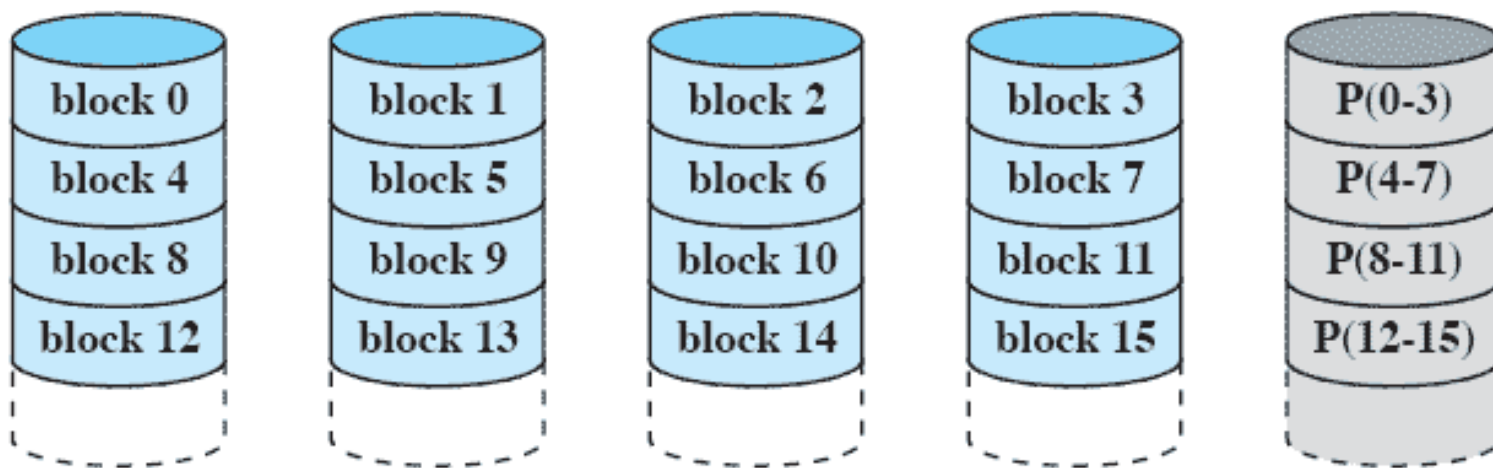
RAID 3 交错位奇偶校验

- ▶ 类似RAID2，以字节为单位将数据拆分，并交叉写入数据盘
- ▶ 专门设置一个存储校验盘，保存校验码（奇偶校验）



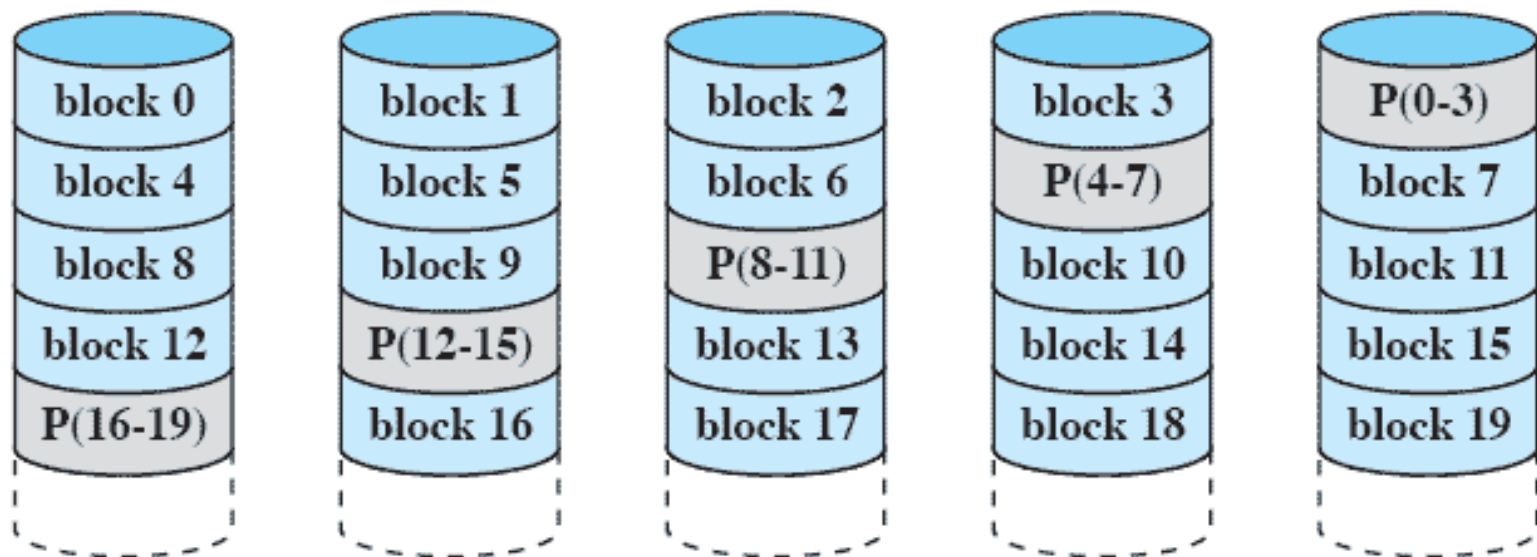
RAID 4 交错块奇偶校验

- ▶ 带奇偶校验
- ▶ 与RAID3相似，但以数据块为单位



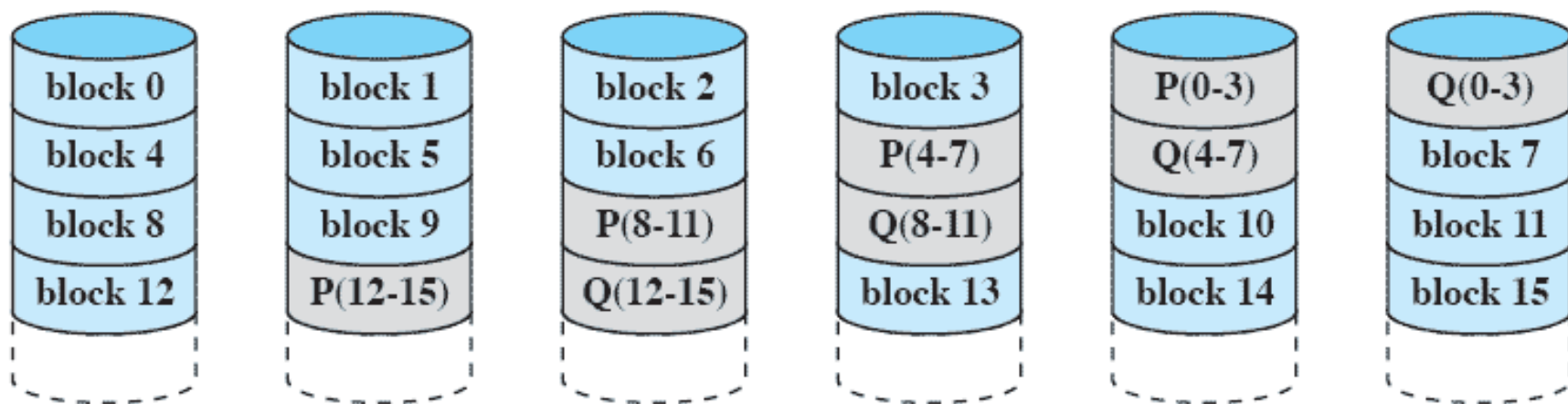
RAID 5 交错块分布式奇偶校验

- ▶ 与RAID4类似，奇偶校验分散在各个磁盘
- ▶ 数据读出效率高，写入效率一般
- ▶ 磁盘利用率较好，提高了可靠性
- ▶ 有写损失



RAID 6 交错块双重分布式奇偶校验

- ▶ 在RAID5的基础上，设立两个校验码，并将校验码写入两个驱动器
- ▶ 数据恢复能力增强
- ▶ 磁盘利用率降低，写能力降低

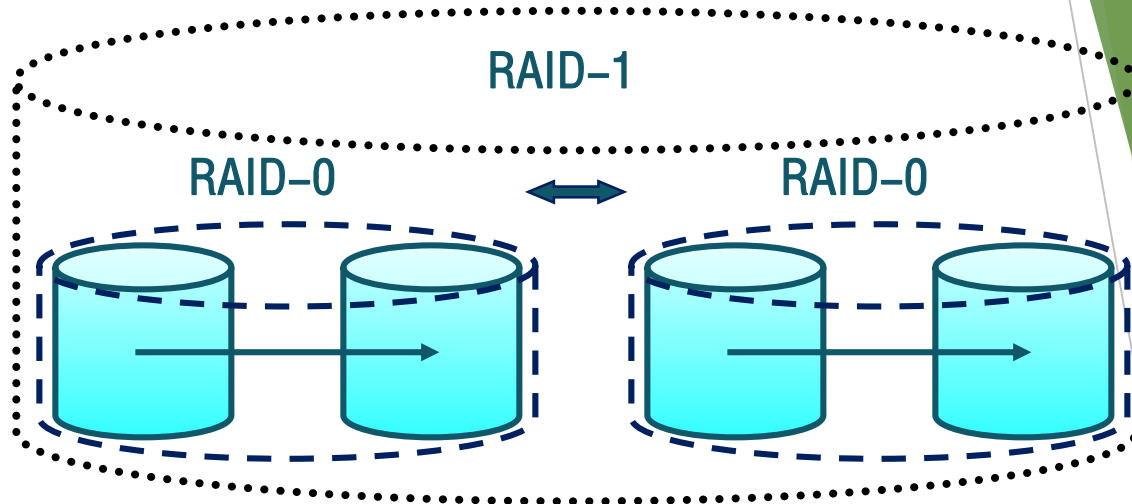


RAID 7 最优化异步高I/O速率及高数据传输率

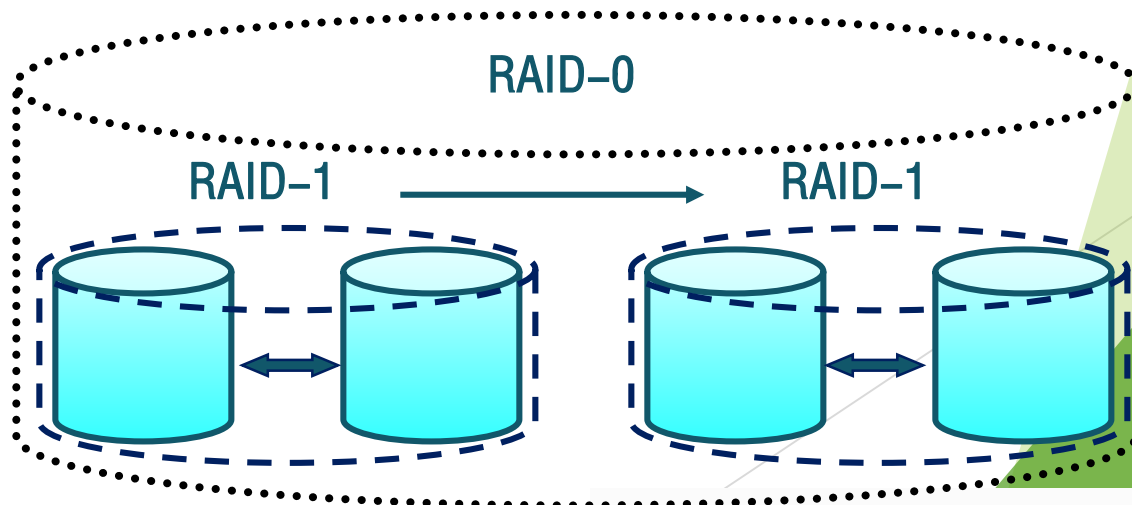
- ▶ 自身带有智能化实时操作系统和用于存储管理的管理工具，独立于主机运行
- ▶ 每个磁盘有独立的I/O通道，与主通道连接
- ▶ 操作系统直接对每个磁盘的访问进行控制，可以让每个磁盘在不同的时段进行数据读写
- ▶ 价格高

RAID 嵌套

RAID 0+1



RAID 1+0



层次模型、虚拟文件系统、.....

文件系统的结构设计

文件系统分类

- ▶ 磁盘文件系统

实例系统：FAT、NTFS、ext2/3、ISO9660等

- ▶ 数据库文件系统

实例系统：WinFS

- ▶ 日志文件系统

- ▶ 网络/分布式文件系统

实例系统：NFS、SMB、AFS、GFS

- ▶ 虚拟文件系统

设计问题

- ▶ 如何定义文件系统对用户的接口?
 - ▶ 文件及属性
 - ▶ 文件操作
 - ▶ 目录结构
- ▶ 如何将逻辑文件系统映射到物理磁盘设备上?
 - ▶ 数据结构与算法
- ▶ 文件系统实现时如何分层

文件系统通用模型

应用程序

文件系统接口

逻辑文件系统层

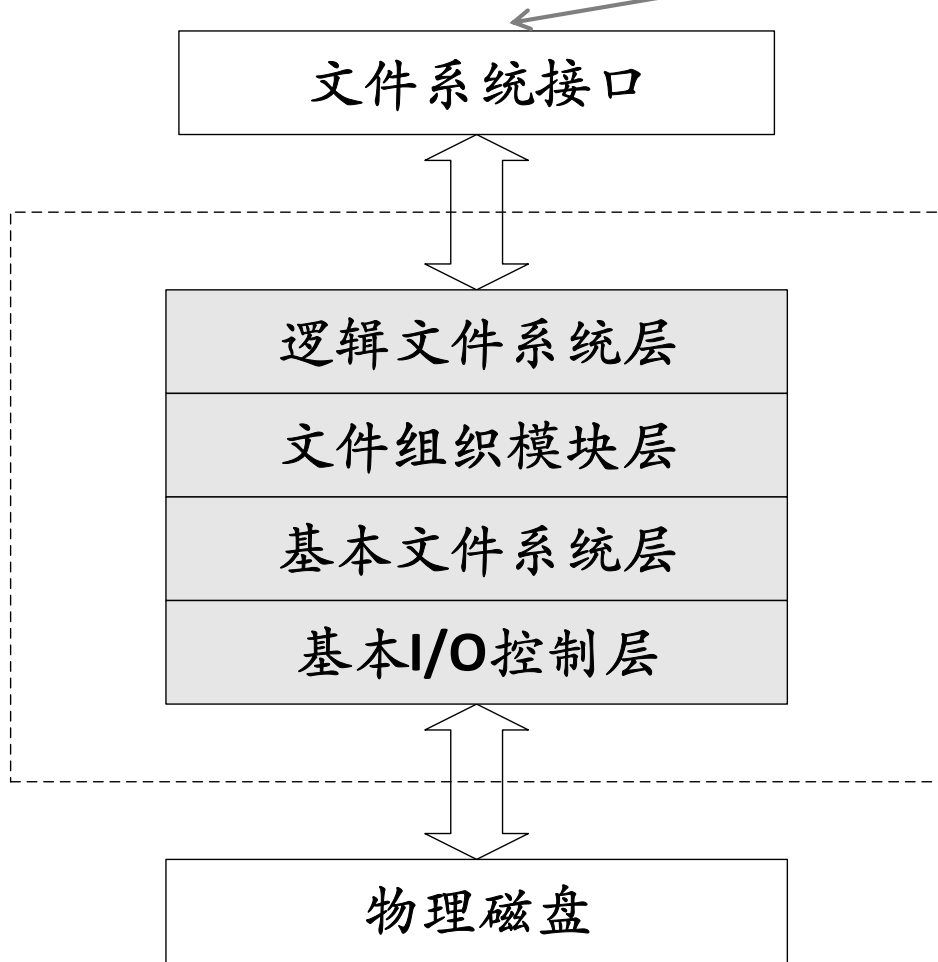
文件组织模块层

基本文件系统层

基本I/O控制层

物理磁盘

层次模型



各层的作用

► 文件系统接口

定义了一组使用和操作文件的方法

► 逻辑文件系统层

使用目录结构为文件组织模块提供所需的信息，并负责文件的保护和安全

► 文件组织模块层

负责对具体文件以及这些文件的逻辑块和物理块进行操作

► 基本文件系统层

主要向相应的设备驱动程序发出读写磁盘物理块的一般命令

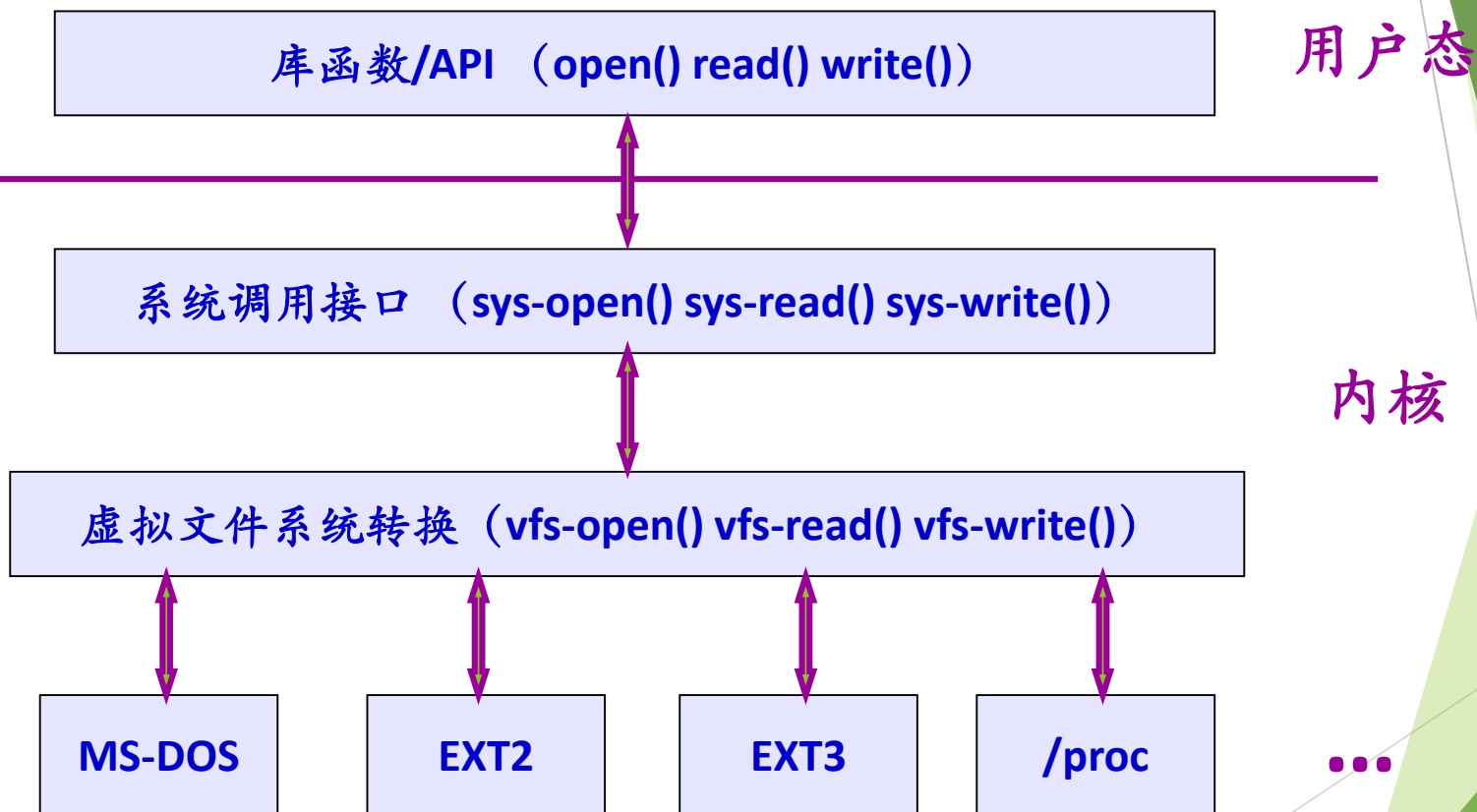
► 基本I/O控制层

由设备驱动程序和中断处理程序组成，实现内存和磁盘系统之间的信息传输

虚拟文件系统

- ▶ 对多个不同文件系统的抽象
- ▶ 功能
 - 提供相同的文件和文件系统接口
 - 管理所有文件和文件系统关联的数据结构
 - 高效查询例程，遍历文件系统
 - 与特定文件系统模块的交互

虚拟文件系统



日志结构文件系统

典型的写操作步骤：
文件目录i节点、目录项、文件的i节点、文件本身

► LFS- Log-structured File System

► 思路(明确问题在哪里?)

提高磁盘写操作的效率(读操作由文件缓存满足)

→ 避免寻找写的位置

- 把整个磁盘看作是一个日志，每次写到其末尾
- 集中(按一段)写入日志末尾
- 将i节点和文件内容一起写入，建立i节点表
- 清理线程：扫描日志，清理，生成新的段

日志文件系统

- ▶ 借鉴日志结构文件系统的设计思路：鲁棒性
- ▶ 保存一个日志：记录系统下一步将要做什么
- ▶ 系统出错后，恢复时查看日志，完成所作操作
- ▶ Windows的NTFS，Linux的ext3、ReiserFS

1. 在目录中删除文件
2. 释放i-节点到空闲i-节点池
3. 将所有磁盘块归还空闲磁盘块池

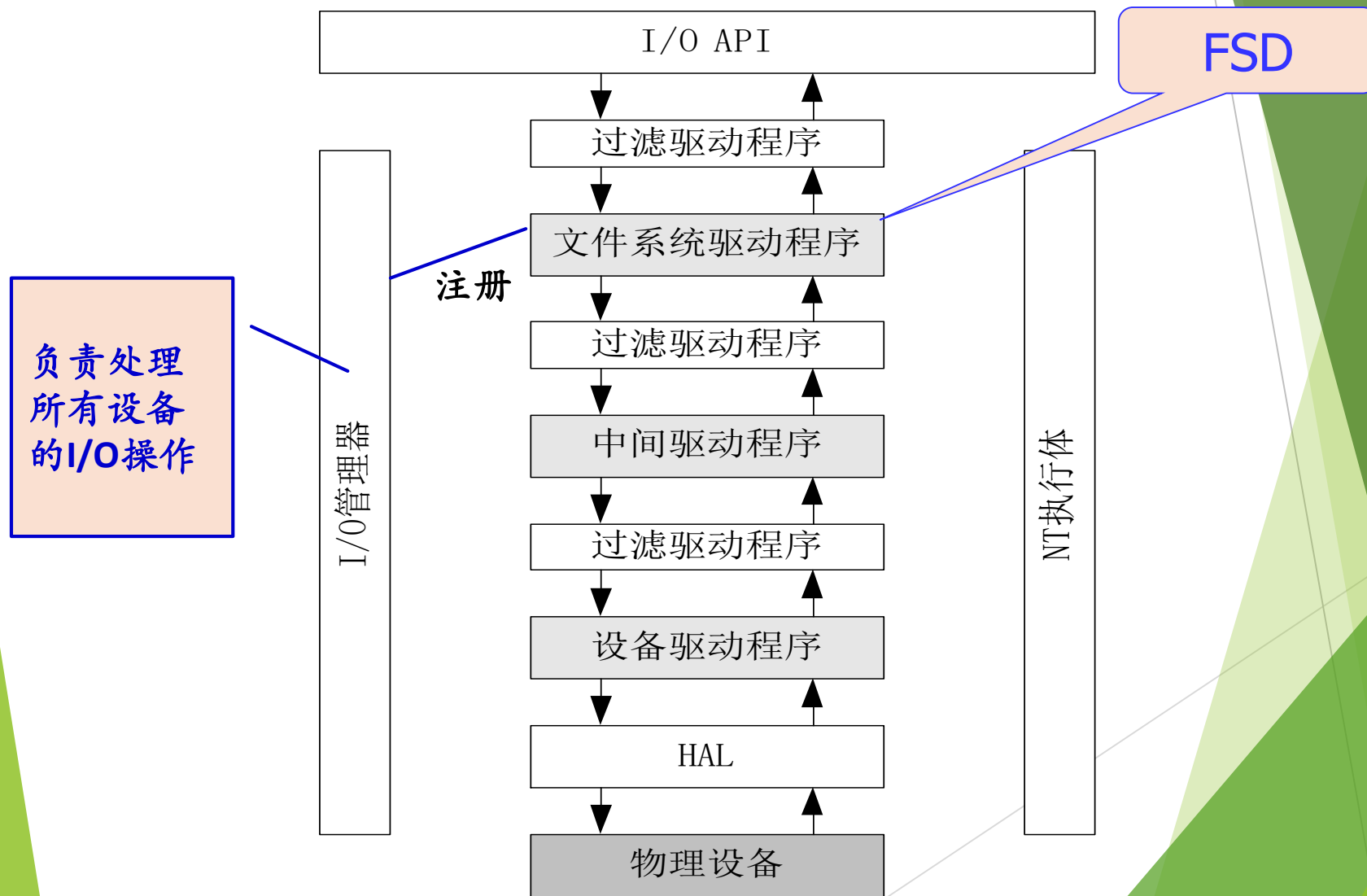
典型步骤

步骤：

- 写日志项(3个将完成的操作)
- 把日志项写入磁盘
- 执行操作
- 擦除日志项

*Windows*文件系统模型

Windows文件系统模型



FSD (文件系统驱动程序)

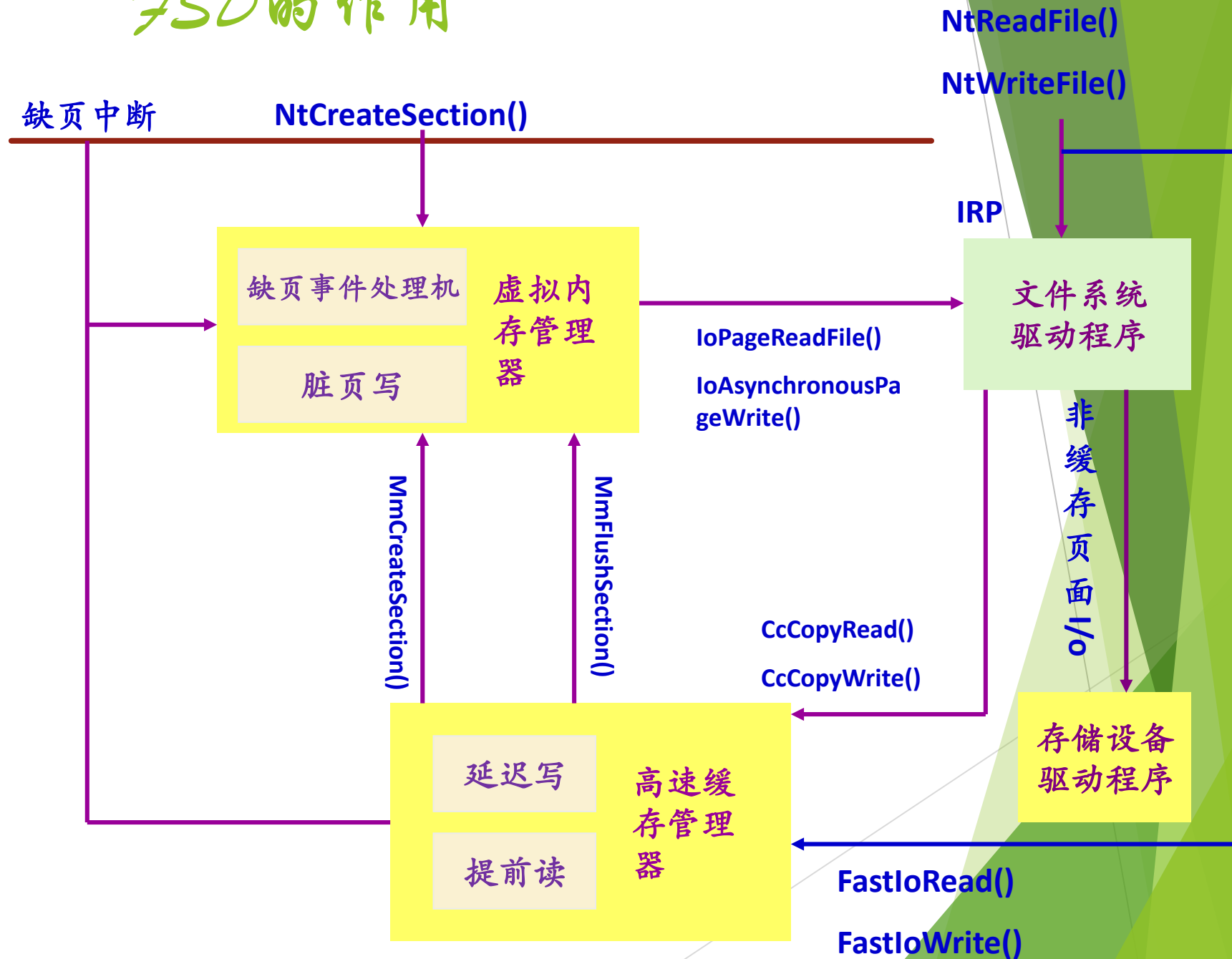
分为本地FSD和远程FSD

- ▶ **本地FSD**: 允许用户访问本地计算机的数据
- ▶ **远程FSD**: 允许用户通过网络访问远程计算机上的数据

作用: **Windows**文件系统的有关操作都通过FSD来完成

- ✓ 显式文件I/O
- ✓ 高速缓存滞后写
- ✓ 高速缓存提前读
- ✓ 内存“脏”页写
- ✓ 内存缺页处理

FSD的作用



(1) 显式文件I/O

- ▶ **CreateFile(ReadFile,WriteFile)** (Win32API, 在Kernel32.dll中)

=> **NtCreateFile**

=>通过ObOpenObjectByName解析名称字符串

=>通过IoParseDevice创建IRP (I/O request packet, I/O请求包)

=>通过IoCallDriver将IRP交给合适的FSD以创建文件

显式文件I/O

► ReadFile

=> NtReadFile

=> 将已打开文件的句柄转换成文件对象指针

=> 检查访问权限

=> 创建IRP读请求

=> 通过IoCallDriver将IRP交给合适的FSD

=> 如可放在高速缓存，则应检查PrivateCacheMap

=> 如有效则表示该文件已有私有高速缓存映射结构；

=> 如无效则表示尚没有私有高速缓存映射结构，需要调用CcInitializeCacheMap来初始化

=> 通过CcCopyRead从高速缓存中读取数据。如果数据还不es高速缓存中，CcCopyRead会引起缺页异常，并间接调用MmAccessFault

(2) 高速缓存延迟写、提前读

- ▶ 高速缓存管理器的滞后写线程定期地对高速缓存中已修改的页面进行写操作
 - 通过调用内存管理器的MmFlushSection函数来完成
 - 具体地说，MmFlushSection通过IoAsynchronousPageWrite将数据送交FSD
- ▶ 高速缓存管理器的提前读线程负责提前读数据：通过分析已作的读操作，来决定提前读多少
 - 提前读线程是通过缺页异常来完成的

(3) 内存“脏”页写

► 内存脏页写线程定期地清洗缓冲区

→ 该线程通过IoAsynchronousPageWrite来创建IRP写请求，这些IRP被标识为非缓存I/O，因此它们被FSD直接送交到磁盘存储驱动程序，将内存数据写到磁盘

(4) 内存缺页处理

- ▶ 在进行显式I/O操作与高速缓存提前读时，都会用到内存缺页处理
- ▶ 应用程序访问内存映射文件且所需页面不在内存时，也会产生内存缺页处理
- ▶ 内存缺页处理MmAccessFault通过IoPageRead向文件所在文件系统发送IRP请求包来完成

分布式文件系统

分布式文件系统1

► 分布式计算机系统

- 由多台分散的计算机互连而成的计算机系统
- 强调资源、任务、功能和控制的全面分布
- 各个资源单元（物理或逻辑的）既相互协作又高度自治，能在全系统范围内实现资源管理，动态进行任务分配或功能分配，并行运行分布式程序

► 工作方式

- 任务分布
- 功能分布

► 分布式文件系统

完成的功能类似于传统操作系统中的文件系统——永久性存储和共享文件，允许用户直接存取远程文件而不需要将它们复制到本地

分布式文件系统2

► 设计上满足一些透明性要求

► 存取透明性

► 位置透明性

► 故障透明性

► 并发存取透明性

► 故障透明性

► 性能透明性

► 复制透明性

► 迁移透明性

系统的透明性
(transparency)

系统的内部实现细节对用户是隐藏的

Hadoop Distributed File System (HDFS)实现机制概述

Hadoop Distributed File System

- ▶ 什么是HDFS
 - ▶ 分布式文件系统
 - ▶ 部署在低廉的 (low-cost) 硬件上
 - ▶ 高容错性 (fault-tolerant)
 - ▶ 高吞吐量 (high throughput)
 - ▶ 适用于超大数据集 (large data set)

HDFS实现上的特点

▶ 块存储

- ▶ 什么是块存储
- ▶ 为什么引入块存储

▶ 读写策略

- ▶ 流式读写
- ▶ 为什么引入流式读写

▶ 数据冗余

- ▶ 数据备份
- ▶ 负载均衡
- ▶ 机架感知

▶ 单点故障处理

- ▶ 体系结构与单点故障点
- ▶ 高可用模式
- ▶ 数据持久化机制
- ▶ 需要备份的数据
- ▶ 数据热备份

HDFS块存储

► 什么是块存储

- 最基本的存储单元是数据块
- 数据块默认是64M
- 如果文件大小大于数据块容量，文件将会被拆分
- 如果文件大小小于数据块容量，文件不独占数据块

► 为什么引入块存储

- 数据块比磁盘块（512K）更大，减少了寻址规模
- 文件以块为单位存储在不同的磁盘上，文件大小不受限于单个磁盘的容量（针对大文件）
- 相对于文件，以数据块为单位更便于数据备份

HDFS读写策略

▶ 流式读写

- ▶ 校验和 (checksum) 校验

- ▶ 只支持在文件末尾添加数据 (Append)

- ▶ 不支持在文件任意位置修改

- ▶ 并发读文件

- ▶ 不支持并发写文件

▶ 为什么引入流式读写

- ▶ 一次写入多次读取

- ▶ 高吞吐量

HDFS数据冗余

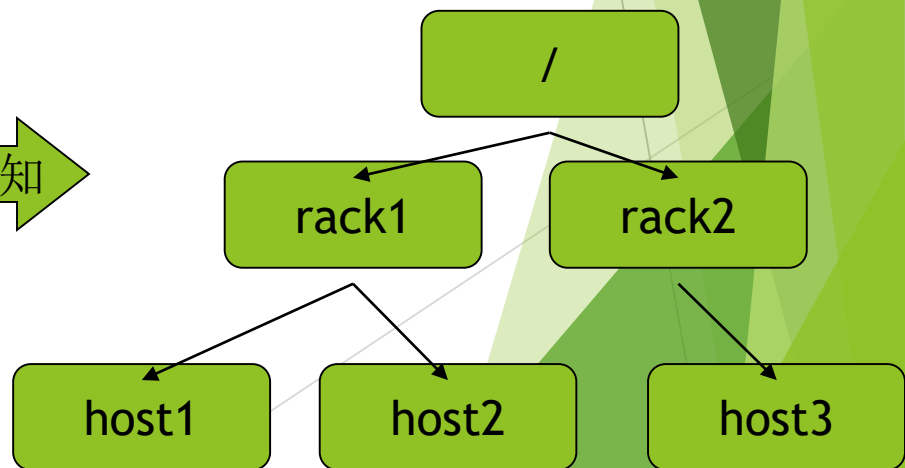
- ▶ 数据备份
 - ▶ 复制因子 (Replication factor) 默认为3
- ▶ 负载均衡
 - ▶ 本地节点一个副本
 - ▶ 同机架 (rack) 其他节点一个副本
 - ▶ 其他机架的节点一个副本
- ▶ 机架感知
 - ▶ IP地址-->拓扑位置

```
#!/usr/bin/python
#-*-coding:UTF-8 -*-
import sys
```

```
rack = {
    "9.111.158.199":"/rack1",
    "9.111.158.73":"/rack1",
    "9.111.158.76":"/rack2",
}
```

```
if __name__=="__main__":
    print "/" + rack.get(sys.argv[1],"rack2")
```

机架感知



HDFS单点故障处理

► HDFS体系结构与单点故障点

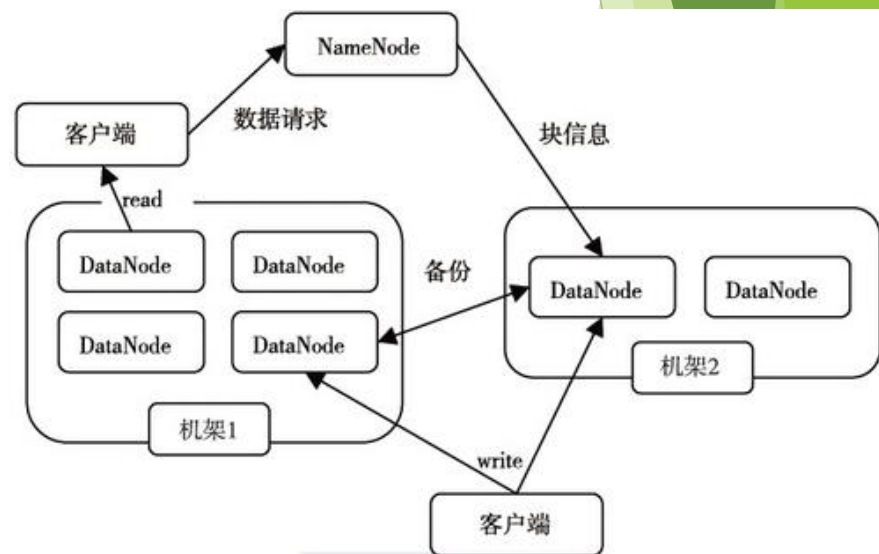
► 主从模式

► 名字节点 (NameNode)

- 唯一存放元数据的节点
- 执行对文件系统元数据的操作
- 负责数据块到数据节点的映射
- 单点故障点

► 数据节点 (Datanode)

- 管理存储的数据



HDFS单点故障处理

- ▶ 高可用模式 (High Availability)
 - ▶ 引入热备节点 (Standby NameNode)
 - ▶ 对名字节点元数据进行热备份
- ▶ 数据持久化 (checkpoint) 机制
 - ▶ 内存-->磁盘
 - ▶ 事务日志, 记录引发元数据改变的操作
 - ▶ 镜像文件, 元数据在磁盘上的副本
 - ▶ 镜像文件+事务操作=新的元数据镜像

HDFS单点故障处理

► 需要备份的数据

► 事务日志

► 镜像文件

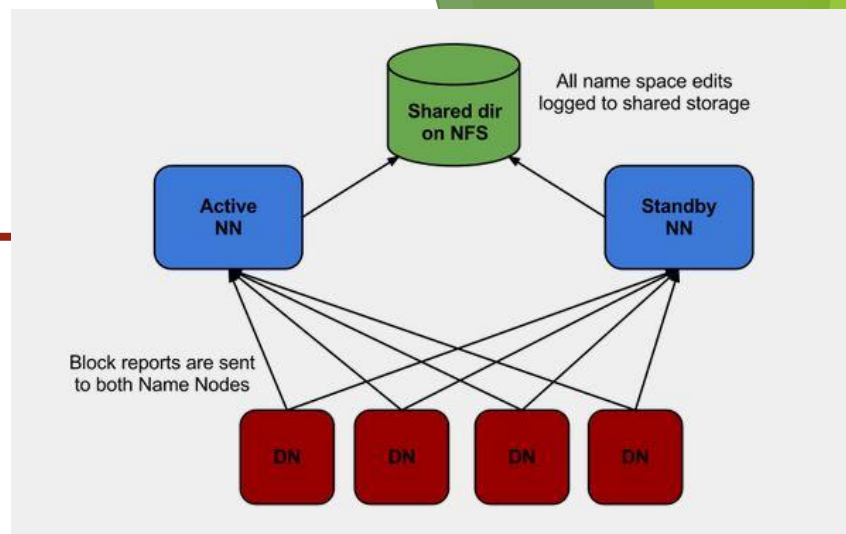
► 数据块到数据节点的映射关系

► 数据热备份

► 事务日志和镜像文件定期更新到网络文件系统

► 热备节点根据镜像和日志还原完整的元数据信息

► 块副本和数据节点的映射关系，数据节点像两个名字节点同时汇报



HDFS其他特性

- ▶ 安全模式
- ▶ 缓存一致性控制
- ▶ 文件删除后恢复
- ▶ Kerberos安全机制

重点小结

□ 文件系统的管理

- ▶ 文件系统的备份
- ▶ 文件系统的一致性

□ 文件系统的性能优化

- ▶ 块高速缓存
- ▶ 磁盘调度
- ▶ RAID技术

□ 文件系统的结构设计

- ▶ 文件系统的层次模型
- ▶ 虚拟文件系统
- ▶ 日志结构文件系统
- ▶ 日志文件系统

□ NTFS文件系统

- * Windows文件系统模型及文件驱动程序
- * 文件系统布局
- * MFT
- * 文件如何存放
- * 目录如何组织

作业9

作业提交时间：

2020年12月20日 晚23:30

-
- 1、总结“文件缓冲”技术
 - 2、总结虚拟文件系统
 - 3、总结RAID技术

The background features abstract, overlapping green geometric shapes, primarily triangles and polygons, in various shades of green, creating a modern and dynamic look.

Thanks

The End