内存管理

内存分层

地址空间和地址生成

逻辑地址生成：变量[编译]—>连续地址[链接]—>全局连续地址[载入]—>（内存中）逻辑地址

物理地址生成：逻辑地址[MMU映射]—>物理地址—>内存（命中）—>[未命中][置换]I/O

连续内存分配

内存碎片问题（内碎片、外碎片）

分区的动态分配

第一适配

最佳适配

最差适配

压缩式碎片整理

交换式碎片整理

非连续内存

分段

分段地址：段号+偏移

段表：段号 | 长度 | 基址

分页

帧：非连续的物理内存（物理内存被分割为大小相等的帧）

帧地址：帧号+帧內偏移

页：连续的虚拟内存

页地址：页号+页内偏移

页表： 页号 | 帧号

快表（TLB）：页号 | 帧号 -----命中/未命中（常用缓存，解决间址速度慢）

二级页表（解决页表空间大）：

逻辑地址：外层页号+外层页内地址+偏移

一级页表：外层页号 | 指定页表基址

二级页表：外层页内地址 | 帧号

虚拟内存

覆盖技术

分时共享内存：常驻区、覆盖区i

缺点：设计开销（程序员自己管理）、效率问题（换入换出IO）

粒度：程序内部无调用关系的代码段

调用者：程序员

交换技术

关键点：交换时机、交换区的大小、程序换入时的重定位

缺点：力度太大、cpu负担太大

粒度：程序间

调用者：系统，对程序员透明

虚拟内存技术：

程序局部性原理：程序执行的较短时间内，指令地址和操作数地址局限于一定区域

时间局部性

空间局部性

基础：在页式或段式内存管理的基础上实现

基本特征

大的用户空间：物理内存和外存结合

部分交换：页或段为单位

不连续性：物理内存分配不连续、虚拟地址空间使用不连续

实现方式：在页式存储管理的基础上，增加请求调页和页面置换功能

页表表项：逻辑页号 | 访问位+修改位+保护位+驻留位 | 物理页帧号

驻留位：表示该页在不在内存中

保护位：表示允许对该页作何种类型的访问

修改位：表示此页在内存中是否被修改过

访问位：表示该页被访问过

缺页中断：

1. if内存有空闲页面，则分配一物理页帧f，then go 4，else go 2.

2.采用某种页面置换算法，选择一将被置换的物理页帧f（对应逻辑页为q），

如过该页在内存期间被修改过，则须把它写回外存

3.对q对应的页表项进行修改，驻留位改为0

4.将需要访问的页p装入到物理页面f当中

5.修改p对应的页表项，驻留位改为1，物理页帧号置为f

6.重新运行被中断的指令

页面置换算法：

最优页面置换算法：置换的页面是将来最长时间不需要的页面、

FIFO：在内存中驻留时间最长的页面并淘汰之

缺点：调出页面经常是要经常访问的页面，并有belady现象

最近最久未使用算法LRU（考虑时间）： 选择最久未使用的那个页面

时钟页面置换算法：

当页面装入内存时访问位置0，访问时置1

把页面组织成环形链表，把指针指向最老的页面

缺页时，考察指针指向的页面，若访问位为0，立即淘汰，然，把该位置

0，指针下移，如此下去，直到找到被淘汰的页面

缺点:替换“脏”页代价巨大

二次机会法

思想：修改clock算法，运行脏页在一次时钟扫描中保存下来

同时使用脏位和使用位来指导置换 11->01->00(替换)

最不常用算法LFU（考虑频度）：

思路：选择访问次数最少的那个页面，并淘汰之。

Belady现象：分配物理页面数增加，缺页率反而提高的异常现象

原因：置换特征与进程访问内存的动态特征是矛盾的

进程管理（资源管理+执行功能（线程））

进程的描述

定义：一个具有一定独立功能的程序在一个数据集合上的一次动态执行过程

组成：

程序的代码

程序处理的数据

程序计数器中的值，只是下一条将运行的指令

一组通用的寄存器的当前值，堆、栈

一组系统资源（如打开的文件）

特点

动态性：可动态的创建、结束进程

并发性：进程可以被独立调度并占用处理机运行

独立性：不同进程的工作不相互影响

制约性：因访问共享内存/资源或进程间同步而产生制约

进程控制结构

进程控制块PCB：操作系统控制进程运行所用的信息集合（唯一标识）

PCB包含三类信息

进程表示信息（进程标识，父进程标识，用户标识）

处理机状态信息保存区（保存进程的运行现场信息）

用户可见的寄存器（用户程序可用数据，如地址等寄存器）

控制和状态寄存器（如程序计数器）

栈指针（调用/中断处理和返回时需要用到它）

进程控制信息

调度和状态信息

进程间通信信息

存储管理信息

进程所用资源

有关数据结构链接信息

PCB组织方式：链表（常用---进程动态性）、索引表

进程状态

进程的生命周期管理：创建、就绪、运行、等待、结束



进程挂起

阻塞挂起状态：进程在外存并等待某事件的出现

就绪挂起状态：进程在外存，但只要进入内存，即可运行

挂起（把一个进程从内存转到外存）

阻塞到阻塞挂起

就绪到就绪挂起

运行到就绪挂起

就绪挂起到就绪

阻塞挂起到阻塞

在外存时的状态转换

阻塞挂起到就绪挂起：阻塞挂起的进程相关事件出现时，转化为就绪挂起进程

线程管理（共享资源平台）

定义：进程中的一条执行流程（线程=进程-共享资源）

从两方面理解进程：

资源组合角度：进程组合资源,构成资源平台（环境）,包括地址空间和打开的文件

运行的角度：代码在这个资源平台上的一条执行流程（线程）

优点：

一个进程可以同时存在多个线程

各个线程之间可以并发的执行

各个线程之间可以共享地址空间和文件资源

缺点：

一个线程崩溃，会导致其所属进程的所有线程崩溃

线程与进程的比较：

进程是资源分配的单位，进程是cpu调度的单位

进程拥有一个完整的资源平台，而线程之独享必不可少的资源，如寄存器和栈

线程同样具有就绪、阻塞和执行三种基本状态，同样具有状态之间的转换关系

线程能减少并发执行的时间和空间开销

线程的创建时间比进程短

线程的终止时间比进程短

同一进程内的线程切换时间比进程短（属于同一进程的线程拥有同一个页表）

由于同一进程的各线程共享资源，可直接进行不通过内核的通信

线程分类

用户线程

内核线程

进程控制

进程的创建---pid\_t fork(void)或pid\_t vfork(void) //创建子进程

fork创建的新进程被称为子进程，该函数被调用一次，但返回两次。两次返回的区

别是：在子进程中的返回值为0，而在父进程的返回值为新子进程的进程ID。

创建子进程，父子进程哪个先运行是根据系统调度且复制父进程的内存空间（数据

空间、堆、栈）。

fork创建子进程，子进程复制了父进程的内存空间和寄存器状态

vfork创建子进程，但子进程先运行且不复制父进程的内存空间。

等待子进程结束：wait

他使父进程去休眠来等待子进程的结果

当一个子进程调用exit()的时候，操作系统解锁父进程，并将通过exit传递得到的

返回值作为wait调用的一个结果（连同子进程pid）如果这里没有子进程存活

Wait（）立即返回

若有僵尸进程等待，wait立即返回一个值（并解除僵尸状态）

进程结束执行后，调用exit()

将程序的结果作为一个参数

关闭所有打开的文件，连接等

释放内存

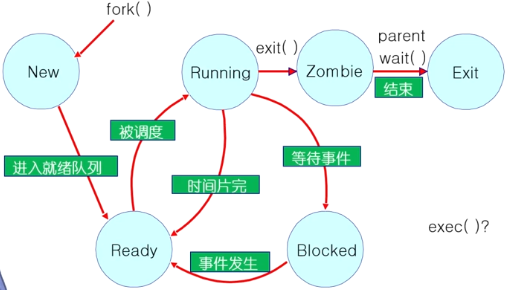
释放大部分支持进程的操作系统

检查父进程是否存活

是，保留结果值知道父进程需要它，这时，进程没有真正死亡，进入僵尸状态

否，它释放所有的数据结构，直到进程死亡

清理所有等待的僵尸进程



处理器CPU调度

上下文切换

切换cpu的当前任务，从一个进程/线程到另一个

保存当前进程/线程在PCB/TCP中的执行上下文（cpu状态）

读取下一个进程/线程的上下文

Cpu调度

从就绪队列中挑选一个进程/线程作为cpu将要运行的下一个进程/线程

调度程序：挑选进程/线程的内核函数（通过一些调度策略）

什么时候进行调度？（就绪-运行，运行-等待，运行-退出）

分类：不可抢占、可以抢占

调度的原则

Cpu使用率

吞吐量---单位时间内完成的进程数量

周转时间---从初始化到结束，包括等待时间和服务时间

等待时间---进程在就绪队列中的总时间

响应时间---从一个请求被提交到产生第一次响应所花费的总时间

调度算法

FCFS---先来先服务

SPN（SJF） SRT---短进程优先（短作业优先）短剩余时间优先---导致饥饿

HRRN---最高响应比优先

Round Robin---轮询（使用时间切片和抢占来轮流执行任务）

Multilevel Feedback Queues---多级反馈队列

Fair Share Scheduling---公平共享调度

同步

独立的线程

不和其他线程共享资源或状态

确定性—>输入状态决定结果

可重现—>能够重现起始条件，I/O

调度顺序不重要

合作线程

在多个线程中共享状态

不确定性

不可重现

Note：不确定性和不可重现意味着bug可能是间歇性发生的

Race condition（竞态条件）

系统缺陷：结果依赖于并发执行或者时间的顺序/时间

不确定性

不可重现

怎样避免竞态？

让指令不被打断

Atomic Operation（原子操作）

原子操作是指一次不存在任何中断或者失败的执行

该执行成功结束

或者根本没有执行

并且不应该发现任何部分执行的状

Critical section（临界区）

临界区是指进程中的一段需要访问共享资源并且当另一个进程处于相应代码区域

时便不会被执行的代码区

实现方法：

禁止中断（仅限于单处理器）

软件方法（复杂）

原子操作指令（单处理器或多处理器均可）

Mutual exclusion（互斥）

当一个进程处于临界区并访问共享资源时，没有其他进程会处于临界区并访问任何

相同的共享资源

Dead lock（死锁）

两个及以上的进程，在相互等待完成特定任务，从而最终没发将自身任务进行下去

Starvation（饥饿）

一个可执行的进程，被调度器持续忽略，以至于虽然处于可执行状态却不被执行

信号量

抽象数据类型

一个整型（sem），两个原子操作

P（）：sem减一，如果sem<0，等待，否则继续

V（）：sem加一，如果sem<=0，唤醒一个等待的P

信号量是整数

信号量是被保护的变量

初始化完成后，唯一改变一个信号量的值的办法是通过P（）和V（）

操作必须是原子

P（）能够阻塞，V（）不会阻塞

我们假定信号量是“公平的”

两种类型信号量

二进制信号量：可以是0或1

一般/计数信号量：可取任何非负值

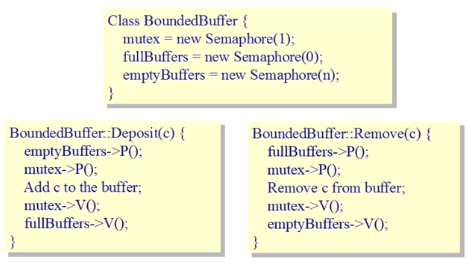
两者可以相互表现（给定一个可以实现另一个）

信号量可以用在两个方面

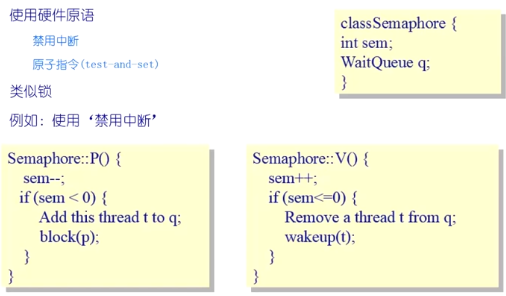
互斥

条件同步（调度约束—一个线程等待另一个线程的事情发生）

信号量的使用（生产者-消费者模型）



信号量的实现



管程

概念：管程 (英语：Monitors，也称为监视器) 是一种程序结构，结构内的多个子程序

（对象或模块）形成的多个工作线程互斥访问共享资源。

目的：分离互斥和条件同步的关注

什么是管程：

一个锁（Lock）：指定临界区

Lock：：Acquire()-等待直到锁可用，然后抢占锁

Lock：：Release()-释放锁，唤醒等待者（如果有）

0或者多个条件变量（Condition Variable）：等待/通知信号量用于管理并发访问共享数据

允许等待状态进入临界区

允许处于等待（睡眠）的线程进入临界区

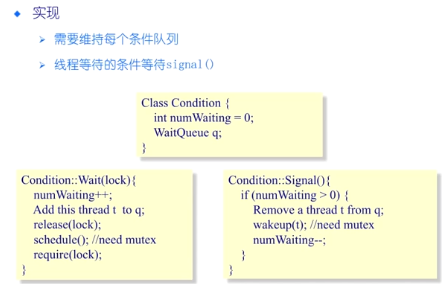
某个时刻原子释放锁进入睡眠

Wait（） operation

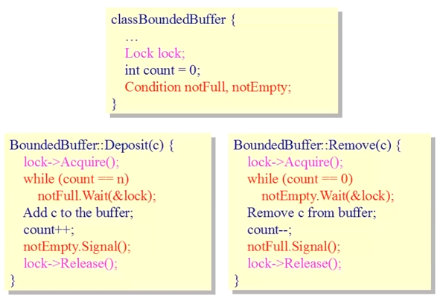
释放锁，睡眠，重新获得锁返回后

Signal（） operation（or broadcast（） operation）

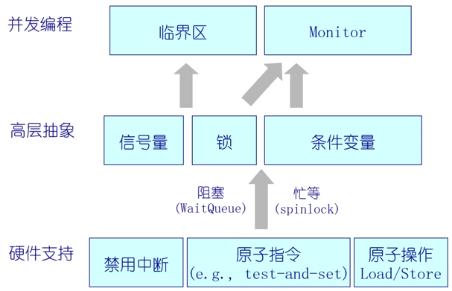
唤醒等待者（或者所有等待者），如果有



使用实例：



总结



死锁

定义：一组阻塞的进程持有一种资源等待获取另一个进程锁占有的一个资源

基本情况：

如果图中不包含循环🡪没有死锁

如果图中包括循环🡪

如果每个资源类只有一个实例，那么死锁

如果每个资源类有几个实例，死锁

死锁的特征：

互斥：在一个时间只能有一个进程使用资源

持有并等待：进程保持至少一个资源正在等待获取其他进程持有的额外资源

无抢占：一个资源只能被进程资源释放，进程已经完成了它的任务之后

循环等待

死锁的处理办法

死锁预防：

互斥—共享资源并不是必须的，必须占用非共享资源

占用并等待—必须保证当一个进程请求资源时，它不持有其他任何资源

要不一下全部申请资源，要不全部给

资源利用率低：可能发生饥饿

无抢占—如果进程占有某些资源，并请求其他不能被立即分配的资源，则释放已占有的资源

循环等待—对所有资源进程排序，并要求其他进程按照资源的顺序进行申请

死锁避免：动态检查资源分配状态，以确保永远不会有一个环形等待状态

银行家算法：

数据结构：

n=进程数量，m=资源类型数量

Max（总需求量）：n\*m矩阵。如果Max[i,j]=k,表示进程Pi最多请求

资源类型Rj的k个实例

Available(剩余空余量)：长度为m的向量。如果Available[j]=k,有k个

类型Rj的资源实例可以使用

Allocation(已分配量)：n\*m矩阵。如果Allocation[i,j]=k，则表示当前

分配了k个Rj的实例

Need（未来需求量）：n\*m矩阵。如果Need[i,j]=k,则Pi可能需要最

少k个Rj实例完成任务

Need[i,j] = Max[i,j] – Allocation[i,j]

Safety state estimating algorithm

1.work和finish分别是长度为m和n的向量

初始化：work=Available， finish[i] = false//线程i没结束

2.找这样的i

（a）finish[i]=false （b）Needi <= Work

没有这样的i，转到4

3.work = work + Allocation，finish[i] = true，转2

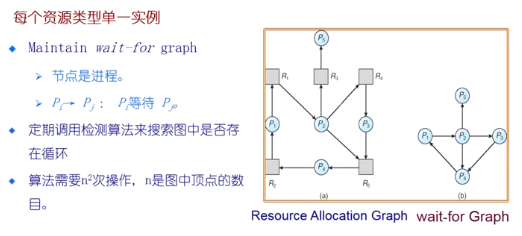
4.if finish[i] == true for all i, then the system is in a safe state

Banker’s algorithm

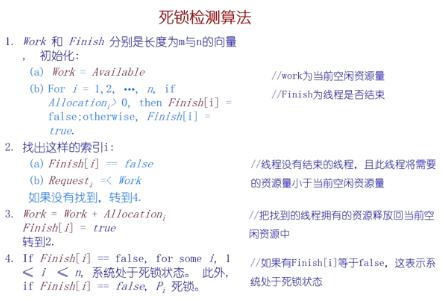


死锁检测

法一：



法二：



死锁恢复

终止所有的死锁进程（环）

在一个时间内终止一个进程直到死锁消除

终止进程的顺序应该是

进程的优先级

程序运行了多久以及需要多少时间才能完成

进程占用的资源

进程完成需要的资源

多少进程需要被终止

进程是交互还是批处理

进程通信

信号：软件中断，通知事件处理

接收到信号时会发生什么

Catch：指定信号处理函数被调用

Ignore：依靠操作系统的默认操作

Mask：闭塞信号因此不会传送

不足：不能传输要交换的任何数据

管道：

子进程从父进程继承文件描述符

进程不知道（或不关心）从键盘、文件、程序读取或写入到终端、文件、程序。

ls | more

消息队列：消息队列按照FIFO的来管理消息

共享内存：

进程：

每个进程都有一个私有地址空间

在每个地址空间内，明确的设置了共享内存段

优点：

快速、方便的共享数据

不足：

必须同步数据访问

Socket：它可用于不同主机间进程通信

文件系统

基本概念

文件系统：一种用于持久性存储的系统抽象

功能：

分配文件磁盘空间

管理文件块（哪一块属于哪一个文件）

管理空闲空间（哪一块是空闲的）

分配算法（策略）

管理文件集合

定位文件及其内容

命名：通过名字找到文件的接口

最常见：分层文件系统

文件系统类型（组织文件的不同方式）

提供的便利及特征

保护：分层来保护数据安全

可靠性/持久性：保持文件的持久即使发生崩溃、媒体错误、攻击等

文件：文件系统中一个单元的相关数据在操作系统中的抽象

文件属性：

名称、类型、位置、大小、保护、创建者、创建时间、最近修改时间…

文件头

在存储元数据中保存了每个文件的信息

保存文件的数据

跟踪哪一块存储块属于逻辑上文件结构的哪个偏移

文件描述符

文件使用模式：使用程序必须在使用前先“打开”文件

内核跟踪每一个进程打开的文件

操作系统为每一个进程维护一个打开文件表

一个打开文件描述符是这个表中的索引

需要元数据数据来管理打开文件

文件指针：指向最近的一次读写位置，每个打开了这问文件的进程都有这个指针

文件打开计数：记录文件打开次数，最后一个进程关闭，允许将其从打开文件表移除

文件磁盘位置：缓存数据访问信息

访问权限：每个程序访问模式信息

在文件系统中的所有操作都是在整个块空间上进行的

目录

文件以目录的方式组织起来

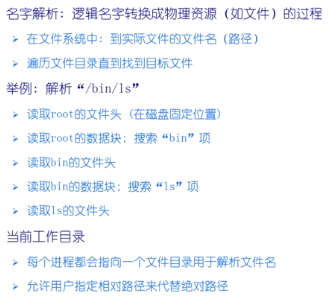
目录是一类特殊的文件

每一个目录都包含了一张表<name, pointer to file header>

目录和文件的树形结构

早期的文件系统是扁平的（只有一层目录）

层级名称空间



一个文件系统必须先挂载才能被访问

一个未挂载的文件系统被挂载在挂载点上

文件别名

硬链接：多个文件项指向一个文件

软连接：以“快捷方式”指向其它文件

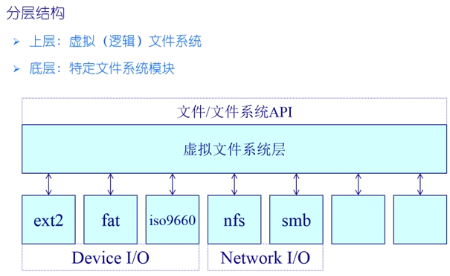
通过存储真实文件的逻辑名称来实现

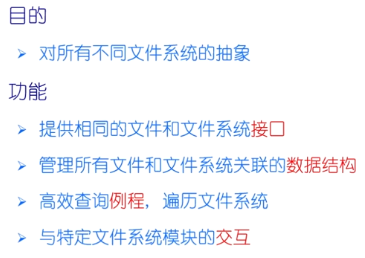


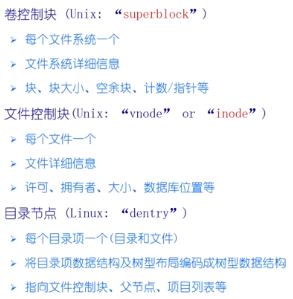
文件系统种类



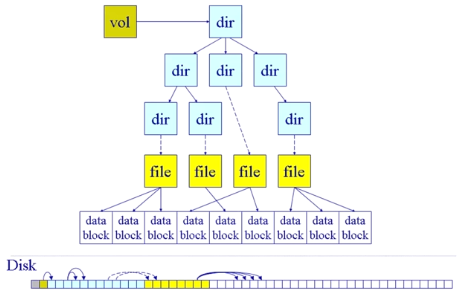
虚拟文件系统

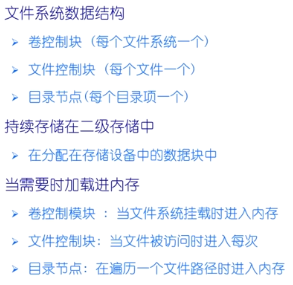




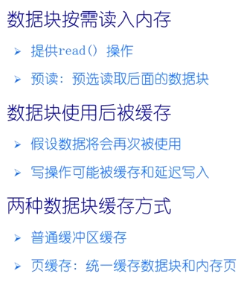


抽象文件系统示意图：

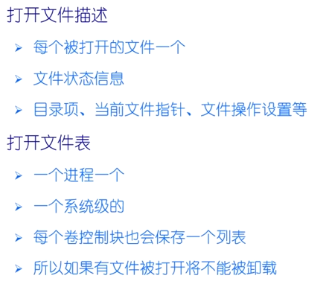


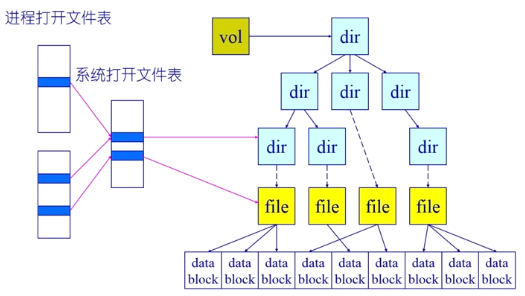


数据块缓存

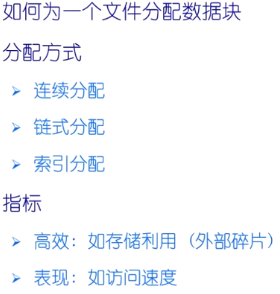


打开文件的数据结构

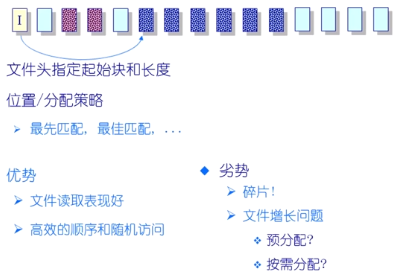




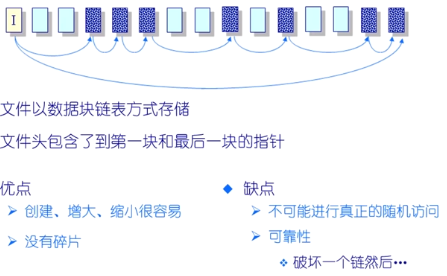
文件分配



连续分配



链式分配

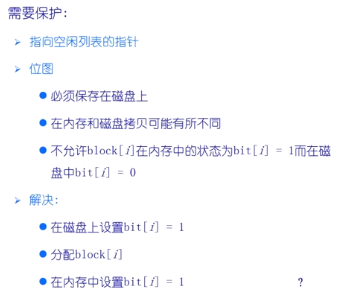


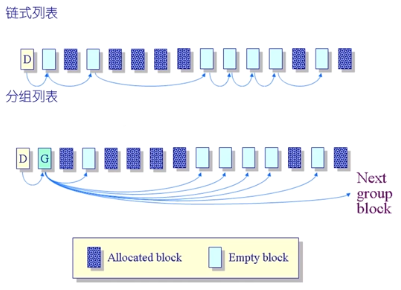
索引分配



空闲空间列表







多磁盘管理 – RAID

磁盘调度