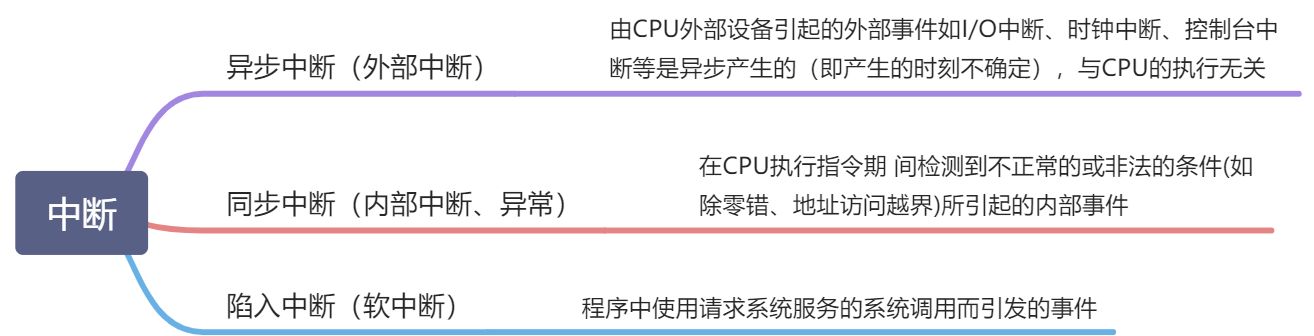
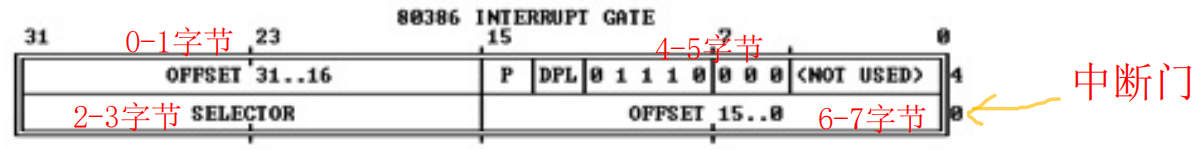
笔记内容：面向912考试，基于912真题和THU期末题以及之前做的重点笔记综合而成的更加凝练的笔记。

适合于最后冲刺阶段经常翻看。

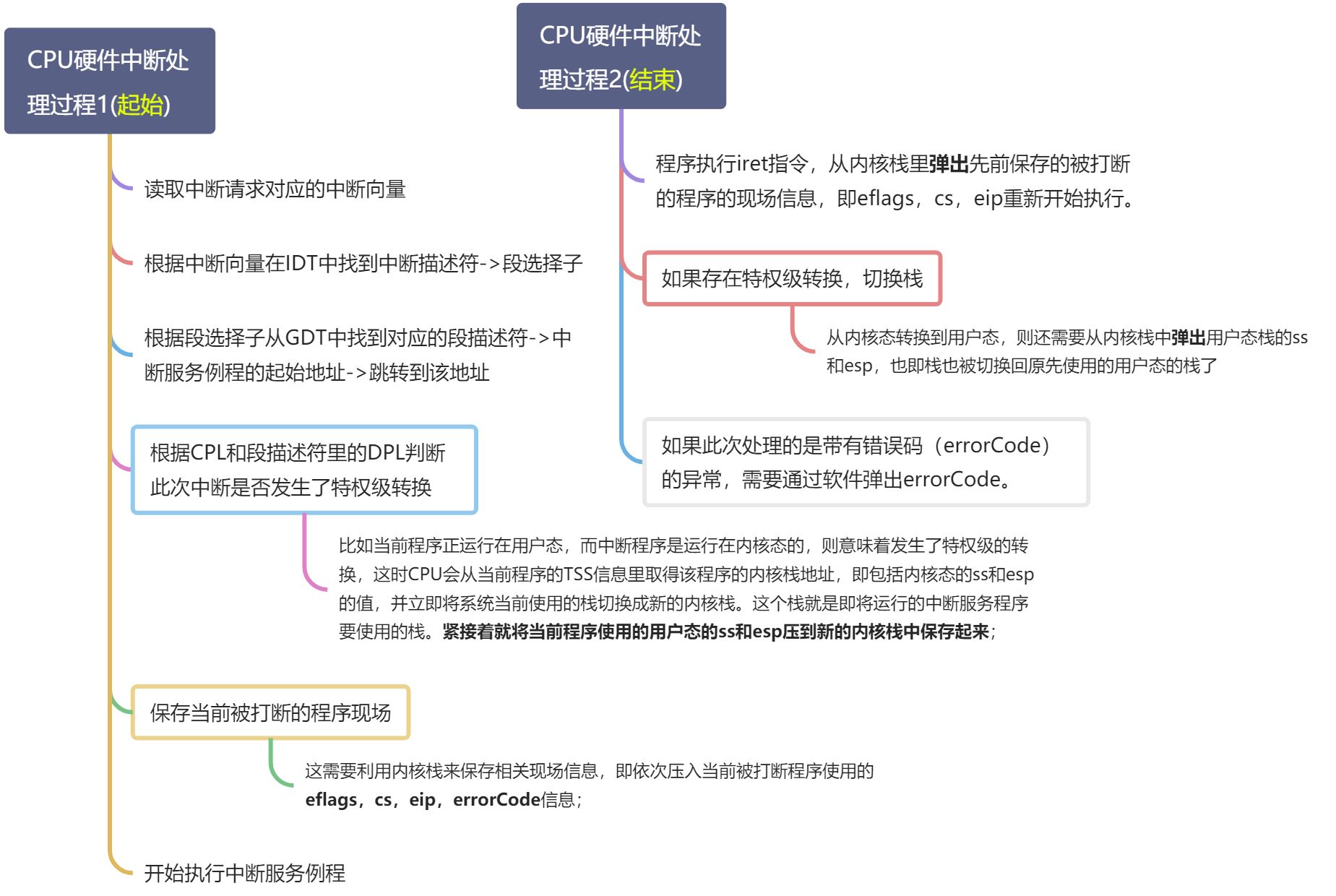
**系统启动、中断、异常和系统调用**



中断向量表一个表项占用8字节，其中2-3字节是段选择子，0-1字节和6-7字节拼成位移，两者联合便是中断处理程序的入口地址。



x86CPU中，CR3寄存器用于存储页目录表起始物理地址。



**物理内存管理**

**连续内存分配**

**动态分区分配策略**

* 最先匹配First-fit — 有外部碎片 ，分配大块时较慢
* 最佳匹配Best-fit — 有外部碎片 ，释放分区较慢 ，容易产生很多无用的小碎片
* 最差匹配Worst-fit — 有外部碎片 ，释放分区较慢 ，能避免出现太多的小碎片 ，但容易破坏大的空闲分区

**伙伴系统(Buddy System)**

既会产生外部碎片也会产生内部碎片

**合并条件**

* 大小相同 2^i
* 地址相邻
* 低地址空闲块起始地址为2(i＋1)的位数    ⭐

**非连续内存分配**

|  |  |
| --- | --- |
| **连续分配的缺点** | **非连续分配的设计目标** |
| * 分配给程序的物理内存必须连续 * 存在外碎片和内碎片 * 内存分配的动态修改困难 * 内存利用率较低 | * 提高内存利用效率和管理灵活性 * 允许一个程序的使用非连续的物理地址空间 * 允许共享代码与数据 * 支持动态加载和动态链接 |

**x86 特权级**

**访问门时** CPL <= DPL[门]  & CPL >= DPL[段] ⭐

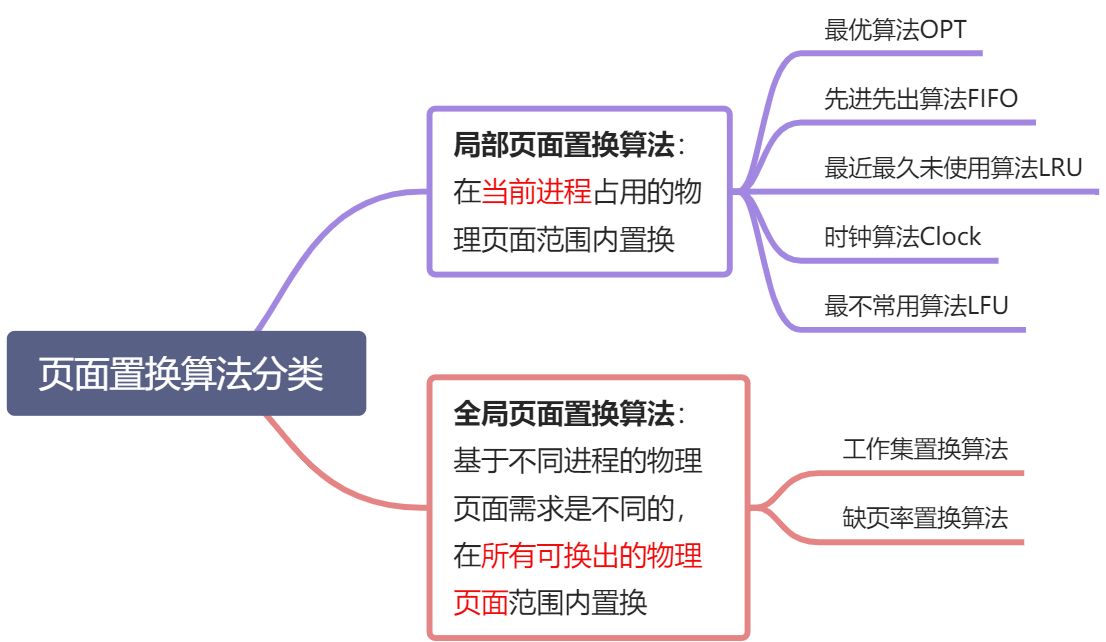
**访问段时** MAX(CPL, RPL) <= DPL[段]

**虚拟存储概念**

三种应对内存不够用的方法：覆盖、交换、虚拟存储

局部性原理包括：时间局部性、空间局部性、分支局部性

**页面置换算法**



**时钟置换算法（Clock）**

**注意几点：**

* 刚开始指针指向最先调入的页面
* 发生换页后，指针停留的位置在换页后的下一个位置
* 如果题目直接给的是访问序列，内存中没有任何已有页，那么这些访问的页第一次访问都算是缺页
* 还需注意对于不缺页即在内存中的页，访问过后要标记为1

**Belady现象**

LRU和OPT属于一种栈算法 ，不产生belady异常

**产生belady的算法：FIFO，CLOCK，改进CLOCK，不恢复计数的LFU。** 重要！

**不产生belady算法：OPT，LRU，恢复计数的LFU。**

**FIFO产生belady异常的例子**

序列：123412512345

物理页面数为3时缺页9次，物理页面数为4时缺页10次，可以看到页面增加，缺页次数反而增多，故属于belady异常。

**CLOCK产生belady异常的例子**

因如果所有页的访问位都为1时，clock算法将退化为FIFO，可以同样构造序列：123412512345

物理页面数为4时缺页10次，而物理页面数为3时缺页9次，页面增加，缺页次数反而增多，故属于belady异常。

**不恢复计数的LFU产生belady的例子**

序列：00111220022313131........后面是3和1的循环

物理页面数为2时，之后3和1循环不会出现任何缺页；而物理页数为3时，之后的3和1的循环总是缺页。

可以看到，物理页面增加了，缺页次数反而增加，故属于belady异常。

**证明OPT和LRU不会产生belady异常**

设有n个物理页页面数，记内存中的页面集合为S。若页面数增加1，则内存中的页面集合为S’

* 对于OPT，S是最近n个**之后即将要访问**的页面，S’是最近n+1个**之后**即将要访问的页面，集合S’包含集合S
* 对于LRU，S是最近n个**之前已经访问**的页面，S’是最近n+1个**之前**已经访问的页面，集合S’包含集合S

所以S中命中的页，在S’中也一定命中，因而缺页率不会增加，因而不会产生belady异常。

**工作集置换算法和缺页率置换算法比较**

工作集算法淘汰页面是在每一次访问页面的时候做这件事情，开销很大

而缺页率置换算法的页面置换是在缺页中断的时候完成，开销更低。

缺页率置换算法的物理页面数是在整个过程中动态调节的，更逼近工作集变化曲线。

**进程和线程及进程控制**

**Copy on wirte机制**

Copy on wirte机制下fork()不会复制进程的页目录表，是多个进程共享。只有当某个进程对某页进行写操作时，才会引发Page fault单独赋值并分配该页给操作进程。

如没有页机制的虚存管理，则无法实现采用Copy on Write机制的fork系统调用。⭐

**僵尸进程**：一个子进程在父进程还没有调用wait()方法或者waitpid()方法的情况下退出，这个子进程就是僵尸进程；

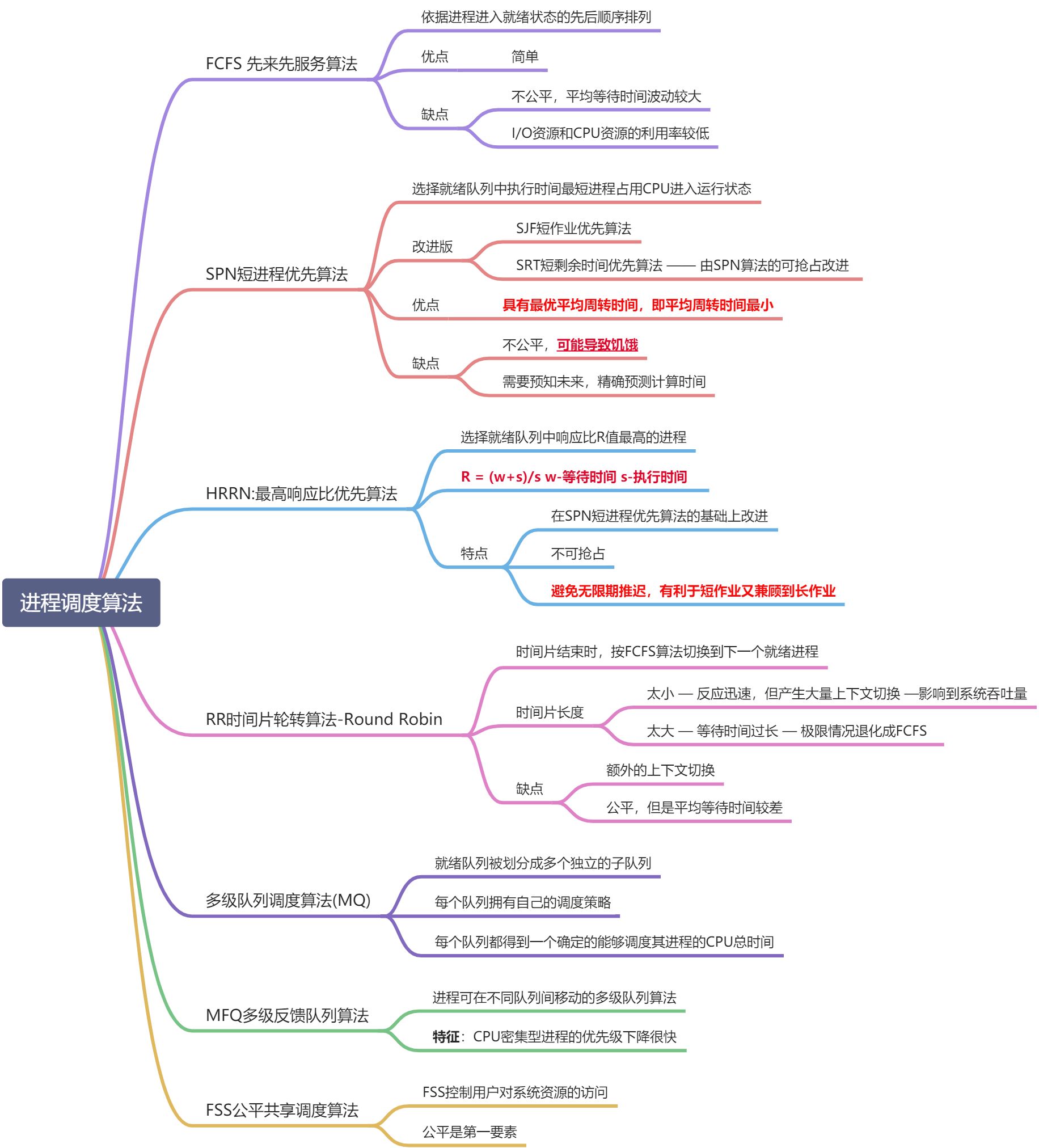
**孤儿进程**：一个父进程退出，它的一个或多个子进程还在运行，子进程将成为孤儿进程，孤儿进程将被init进程所收养；

僵尸进程将会导致资源浪费，而孤儿进程则不会。

exec()系统调用时，进程ID和PID都不会变，但代码，堆栈，堆，数据都会完全被重写（相同的进程，不同的程序）。

**处理机调度**

**进程调度算法**



**对称多处理器(SMP调度**

* 每个处理器运行自己的调度程序
* 调度程序对共享资源的访问需要进行同步
* 进程分配
  + 静态进程分配 —— 进程从开始到结束都被分配到一个固定的处理机上执行，每个处理机有自己的就绪队列。
    - 优点：调度开销小；缺点：各处理机可能忙闲不均
  + 动态进程分配 —— 进程在执行中可分配到任意空闲处理机执行，所有处理机共享一个公共的就绪队列。
    - 优点：调度开销大；缺点：各处理机的负载是均衡的

**优先级反置**是指：低优先级进程抢占了高优先级进程的资源，优先级继承时低优先级进程能动态改变优先级。

**Stride调度算法**

**特点**：

* 简单易于实现以外
* 可控性：如我们之前所希望的，可以证明 Stride Scheduling对进程的调度次数**正比于**其优先级。
* 确定性：在不考虑计时器事件的情况下，整个调度机制都是可预知和重现的。

**Stride调度算法的基本思想**：

1. 为每个runnable的进程设置一个当前状态stride，表示该进程当前的调度权。另外定义其对应的pass值，表示对应进程在调度后，stride 需要进行的累加值。

2. 每次需要调度时，从当前 runnable 态的进程中选择stride最小的进程调度。

3. 对于获得调度的进程P，将对应的stride加上其对应的步长pass，即pass = BIG\_STRIDE / P->priority，P->stride += pass。

4. 在一段固定的时间之后，回到步骤2，重新调度当前stride最小的进程。

可以证明，如果令 P.pass =BigStride / P.priority，则该调度方案**为每个进程分配的时间将与其优先级成正比**。

（ 其中 P.priority 表示进程的优先权 （大于 1），而 BigStride 表示一个预先定义的大常数）

**Stride溢出问题**

因为 P->stride = P->stride + pass，经过不断的累加之后，这个值有可能溢出。在 stride溢出以后，基于stride 的比较可能会出现错误。

**那么如何解决溢出的问题？**

令PASS\_MAX为当前所有进程里最大的步进值，有这样的结论结论：对每次Stride调度器的调度步骤中，有其最大的步进值STRIDE\_MAX和最小的步进值 STRIDE\_MIN之差： STRIDE\_MAX – STRIDE\_MIN <= PASS\_MAX。

有了该结论，在加上之前对优先级有 Priority > 1 限制，我们有 STRIDE\_MAX – STRIDE\_MIN <= PASS\_MAX <= BIG\_STRIDE,于是我们只要将BigStride取在某个范围之内，即可保证对于任意两个 Stride 之差都会在机器整数表示的范围之内。

stride, pass 是无符号整数 ，用**有符号整数**表示（Proc.A.stride –Proc.B.stride）

**【例】**（20180525期末）

设lab6中使用Stride调度算法，取BIGSTRIDE=100，假定各个进程的stride初始化为0。请回答下列问题：

A) 如果不考虑进程stride的值的溢出，那么对于任意两个进程A、B的stride值SA和SB，应当恒有abs(SA-SB) ≤ (\_\_1\_\_) ，为什么？

B) 考虑到abs(SA-SB)这一性质，假设stride值存在溢出，可将stride值的更新变为：stride = (stride + BIGSTRIDE/priority) mod n

那么，只要 n > (\_\_2\_\_) ，那么stride算法就可以正常运行，为什么？

答：

A） abs(SA-SB) ≤ BIGSTRIDE = 100

反证法：

假设：SA-SB > BIGSTRID 成立

stride算法要求调度当前stride值最小的进程。假设某个时刻首次出现了A、B两个进程并且SB>SA+BIGSTRIDE的情况，也即SB>SA，本应该调度A而不是B，但是由于 SA-SB >= BIGSTRID，也即SB≤SA+BIGSTRIDE，那么意味着在上一轮调度中，调度了B而不是A，这就与stride算法矛盾。

故假设不成立，原结论成立。

B）n > 200

原因：若真实值有SB>SA，但SB溢出了SA没溢出，那么需要它们的差值SA-SB>BIGSTRIDE用以判断出SB已经溢出，实际上真实的stride值应当是SB>SA。

即需要满足：

* a. SB-SA ≤ BIGSTRIDE
* b. SA-(SB-n) > BIGSTRIDE

若需要(b)式恒成立，需要 n>2\*BIGSTRIDE

**【例】**(2019年912真题)

stride调度算法中，用八位无符号数来表示进程A的运行时间有可能会溢出，但是当步长s   **<=** 127时，只要做小小的技巧就仍可以正确判断时间，然后又引入进程B，a为A的stride，b为B的stride，当sign(a-b) **<=0** 时说明A正在运行，此时a+s **<** 256，当sign(a+s-b) **>** 0时，轮到B运行。

**同步互斥**

临界区实现方法：

* 禁用中断
* 软件方法
* 更高级的抽象方法 - 锁、信号量、硬件原语

**自旋锁（spinlock）**

* 是指当一个线程在获取锁的时候，如果锁已经被其它线程获取，那么该线程将循环等待，然后不断的判断锁是否能够被成功获取，直到获取到锁才会退出循环。
* 获取锁的线程一直处于活跃状态，但是并没有执行任何有效的任务，使用这种锁会造成[busy-waiting](https://en.wikipedia.org/wiki/Busy_waiting" \t "_blank)。
* 它是为实现保护共享资源而提出一种锁机制。

**自旋锁与互斥锁区别**：

对于互斥锁，如果资源已经被占用，资源申请者只能进入睡眠状态。但是自旋锁不会引起调用者睡眠，如果自旋锁已经被别的执行单元保持，调用者就一直循环在那里看是否该自旋锁的保持者已经释放了锁，”自旋”一词就是因此而得名。

**自旋锁与非自旋锁区别**：

* 自旋锁不会使线程状态发生切换，一直处于用户态，即线程一直都是active的；不会使线程进入阻塞状态，减少了不必要的上下文切换，执行速度快。
* 非自旋锁在获取不到锁的时候会进入阻塞状态，从而进入内核态，当获取到锁的时候需要从内核态恢复，需要线程上下文切换。

**自旋锁**具有随机性，不能保证进程按先来后到的顺序使用cpu资源，顺序可能乱。

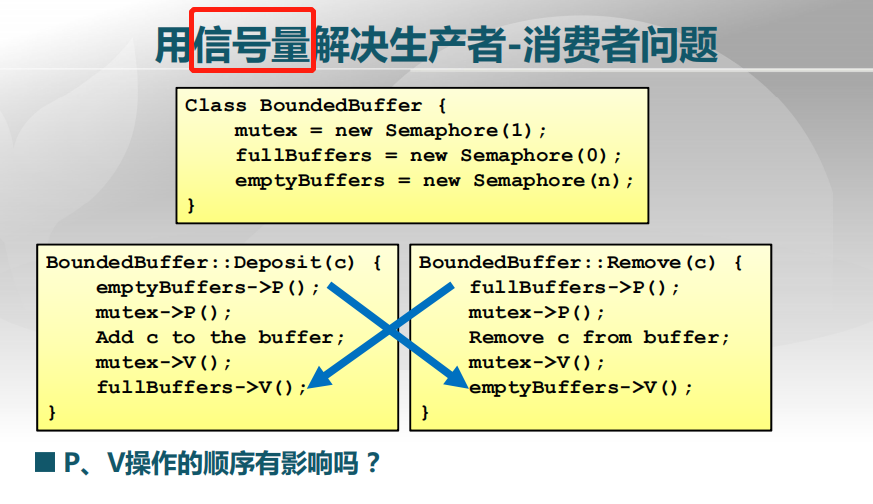
**信号量和管程**



管程和信号量在功能上等价，都是实现进程间同步互斥的方法。

管程将资源抽象成条件变量，通过变量值的增减来控制进程的访问。

**管程**支持暂时放弃互斥资源访问，等待信号。



**【例题】基于信号量的同步互斥**（20201028期末）

假设某仓库可存放 A、B 两种物品，其容量无限大；任何时刻只允许一个物品进行仓库进行入库操作；要求仓库中 A 物品数量 X 和 B 物品数量 Y 满足不等式： -M ≤ ( X - Y ) ≤ N ，其中 M 和 N 为正整数。请用信号量机制实现 A、B 两种物品入库的处理过程，要求能做到正确且高效的同步与互斥。

请说明所定义的信号量的含义和初始值，描述需要进行互斥处理的各种行为，描述需要进行同步处理的各种行为；要求用类 C 语言伪代码实现，并给出必要的简明代码注释。

X-Y <= N   即A的物品数量可以比B多，但是不能超过N个；如果没有B，A最多只能有N个

Y-X <= M  即B的物品数量可以比A多，但是不能超过M个；如果没有A，B最多只能有M个

**死锁和进程通信**

**死锁的四个必要条件**：互斥访问、占有并等待、非剥夺、循环等待。

**进程间四种通信机制：**信号、管道、消息队列和共享队列。

**信号和管道**

**消息队列和共享队列**

**文件系统**

**文件系统和文件**

文件描述符是打开文件的标识

文件系统中的基本操作单位是数据块

目录是一类特殊的文件

* 目录的内容是文件索引表<文件名, 指向文件的指针>

**文件分配分配方式**

* **连续分配**
  + 分配策略：最先匹配, 最佳匹配,
  + 优点：文件读取表现好，高效的顺序和随机访问
  + 缺点：碎片，文件增长问题
* **链式分配**
  + 实现方式：文件以数据块链表方式存储；文件头包含了到第一块和最后一块的指针
  + 优点：创建、增大、缩小很容易 ；没有碎片
  + 缺点：无法实现真正的随机访问；可靠性差，破坏一个链，后面的数据块就丢了
* **索引分配**
  + 实现方式：为每个文件创建一个索引数据块，指向文件数据块的指针列表；文件头包含了索引数据块指针
  + 优点：创建、增大、缩小很容易；没有碎片；支持直接访问
  + 缺点：当文件很小时，存储索引的开销；如何处理大文件? —— 链式索引块 (IB+IB+…)、多级索引块(IB\*IB \*…)

**引用计数计算**

F1是某文件的文件名，F2、F3都是该文件的别名，他们都对应唯一个物理空间，系统中还维护了一个数据结构（UNIX为inode），其中有一数据项为count记录共享该文件的目录数。

[**软链接**](https://www.yuque.com/dekeshile/iiy3kz/bg5pi3)

符号链接（软链接）是一类特殊的[文件](https://baike.baidu.com/item/%E6%96%87%E4%BB%B6)， 其包含有一条以绝对路径或者相对路径的形式指向其它文件或者目录的引用。

路径可以是任意文件或目录，可以链接不同文件系统的文件。（链接文件可以链接不存在的文件，这就产生一般称之为”断链”的现象），链接文件甚至可以循环链接自己（类似于编程中的递归）。

* [新建一个inode的快捷方式；](https://www.yuque.com/dekeshile/iiy3kz/bg5pi3)
* [并不会影响原有文件的count使用计数；](https://www.yuque.com/dekeshile/iiy3kz/bg5pi3)
* [软链接的count总是为1，或者说它的count没有意义；](https://www.yuque.com/dekeshile/iiy3kz/bg5pi3)

[**硬链接**](https://www.yuque.com/dekeshile/iiy3kz/bg5pi3)

* [不会新建一个inode，而是指向同一个文件；](https://www.yuque.com/dekeshile/iiy3kz/bg5pi3)
* [硬，指的是删除硬链接或者源文件，都不会真实删除，而是count-1，直到count=0的时候才会真正删除；](https://www.yuque.com/dekeshile/iiy3kz/bg5pi3)

【例】：设文件F1的当前引用计数值为1，先建立文件F1的符号链接（软链接）文件F2，在建立文件F1的硬链接F3，然后删除文件F1。此时，文件F2和文件F3的引用计数分别是（）。  (2020年)

解答：

当建立F1的软链接F2时，F1和F2的引用计数值都为1。

当再建立F1的硬链接F3时，F1和F3的引用计数值就都变成了2。

当后来删除F1时，F3的引用计数值为2-1=1。F2的引用计数值仍然保持不变。

**RAID-冗余磁盘阵列**

【例】（20180525期末）

现有一个RAID磁盘阵列，包含6个磁盘，每个磁盘大小都是2TB，最大写入速度 200 MB/s，

最大读取速度 250 MB/s 的硬盘。用它们分别组成RAID级分别为0、1和5。假设在理想情况下（无中断、异常、预先缓存等外在干扰因素等），请回答下列问题。

A) 用它们组成的RAID0阵列的总可用空间为 (\_\_24.1\_\_)，最大写入速度为 (\_\_24.2\_\_)，最大读取速度为 (\_\_24.3\_\_)；

B) 用它们组成的RAID1阵列的总可用空间为 (\_\_24.4\_\_)，最大写入速度为 (\_\_24.5\_\_)，最大读取速度为 (\_\_24.6\_\_)；

C) 用它们组成的RAID5阵列的总可用空间为 (\_\_24.7\_\_)，最大写入速度为 (\_\_24.8\_\_)，最大读取速度为 (\_\_24.9\_\_)。

答：

A) 用它们组成的RAID0阵列的总可用空间为 ( 12TB )，最大写入速度为 ( 1200MB/s )，最大读取速度为 ( 1500MB/s )；

B) 用它们组成的RAID1阵列的总可用空间为 ( 6TB )，最大写入速度为 ( 600MB/s )，最大读取速度为 ( 1500MB/s )；

C) 用它们组成的RAID5阵列的总可用空间为 ( 10TB )，最大写入速度为 ( 1000MB/s )，最大读取速度为 ( 1250MB/s )。

**IO子系统**

**磁盘调度算法**

* 先进先出(FIFO)算法
* 最短服务时间优先(SSTF) ,也称为电梯算法(elevator algorithm)
* 循环扫描算法(C-SCAN)
* C-LOOK算法
* 双队列扫描(FSCAN)算法

访问频率置换算法(Frequency-based Replacement)综合采用了LRU和LFU算法用于磁盘缓存置换。

在短周期中使用LRU算法，而在长周期中使用 LFU算法。