# 预备知识：

# 请求分页存储管理技术

请求分页也称为页式虚拟存储管理，它是建立在基本分页基础上，为了能支持虚拟存储器功能而增加了请求调页功能和页面置换功能。

它的基本思想是：在进程开始运行之前，不是装入全部页面，而是装入部分页面，之后根据进程运行的需要，动态装入其他页面。当内存空间已满，而又需要装入新的页面时，根据某种算法淘汰某个页面，以便装入新的页面。

# 一、时间片轮转法

算法描述：用于分时系统中的进程调度。每次调度时，总是选择就绪队列的队首进程，让其在CPU上运行一个系统预先设置好的时间片。一个时间片内没有完成运行的进程，返回到绪队列末尾重新排队，等待下一次调度。

# 二、内存分配策略和分配算法

---- 在为进程分配内存时，将涉及到3个问题：

1）**最小物理块数**的确定2）物理块的**分配策略**3）物理块的**分配算法**

**1、最小物理块数的确定**

-- 这里所说的最小物理块数，是指**能保证进程正常运行**所需的最小物理块数。

-- 当系统为进程分配的物理块数**小于此值**时，进程将无法运行。

-- 进程应获得的最少物理块数与计算机的**硬件结构**有关，取决于**指令的格式、功能和寻址方式**。

**2、物理块的分配策略**

-- 在请求分页系统中，可采取两种内存分配策略，即**固定**和**可变分配**策略。在进行置换时，也可采取两种策略，即**全局置换**和**局部置换**。

于是可组合出以下三种适用的策略。

**1）固定分配局部置换**（Fixed Allocation，Local Replacement）

---- 这是指基于**进程的类型**（交互型或批处理型等），或根据程序员、程序管理员的建议，为每个进程分配**一定数目**的物理块，在**整个运行期间**都不再改变。采用该策略时，如果进程在运行中发现缺页，则只能从**该进程在内存**的n个页面中选出一个页换出，然后再调入一页，以保证分配给该进程的**内存空间不变**（固定分配）。

---- 实现这种策略的困难在于：应为每个进程**分配多少个物理块**难以确定。若太少，会频繁地出现缺页中断，降低了系统的吞吐量；若太多，又必然使内存中驻留的进程数目减少，进而可能造成CPU空闲或其他资源空闲的情况，而且在实现进程对换时，会花费更多的时间。

**2）可变分配全局置换**（Variable Allocation，Global Replacement）

这可能是最易于实现的一种物理块分配和置换策略，已用于若干个OS中。在采用这种策略时，先为系统中的每个进程**分配一定数目的物理块**，而OS自身也保持一个**空闲物理块队列。**当某进程发现缺页时，由系统从空闲物理块队列中取出一个物理块分配给该进程，并将欲调入的（缺）页装入其中。这样，凡产生缺页（中断）的进程，都将获得新的物理块。仅当空闲物理块队列中的物理块用完时，OS才能从内存中选择一页调出，该页可能是系统中**任一进程的页**，这样，自然又会使那个进程的物理块减少，进而使其缺页率增加。

**3）可变分配局部置换**（Variable Allocation，Local Replacement）

这同样是基于进程的类型或根据程序员的要求，为每个进程分配一定数目的物理块，但当某进程发现缺页时，只允许从**该进程在内存的页面中**选出一页换出，这样就**不会影响其它进程的运行**。如果进程在运行中频繁地发生缺页中断，则系统需再为该进程分配若干个附加的物理块，直至该进程的缺页率减少到适当程度为止；反之，若一个进程在运行过程中的缺页率特别低，则此时可适当减少分配给该进程的物理块数，但不应引起其缺页率的明显增加。

---- 全局是指置换页面时，换出的页面可能是内存中的任一进程的页面，局部只能是本（缺页）进程的页面。

---- 固定分配指为一个进程分配的物理块是固定的，可变分配指可根据缺页率调整所分配的物理块数。

**3、物理块分配算法**

-- 在采用固定分配策略时，如何将系统中可供分配的所有物理块分配给各个进程，可采用下述几种算法。

**1）平均分配算法**

-- 这是将系统中所有可供分配的物理块平均分配给各个进程。例如，当系统中有100个物理块，有5个进程在运行时，每个进程可分得20个物理块。

这种方法看似公平，实际不然，因为它**未考虑到各进程本身的大小**。

假如有一个进程大小为200页，只分配给它20个块，它必然有很高的缺页率；而另一进程只有10页，却有10个物理块闲置未用。

**2）按比例分配算法**

-- 这是**根据进程的大小按比例分配**物理块的算法。如果系统中共有n个进程，每个进程的页面数为Si，则系统中各进程页面数的总和为：

    S=S1+S2+...+Si+...+Sn-1+Sn (i=1～n)     又假定系统中可用的物理块总数为m，则每个进程所能分到的物理块数为bi，则有：

    bi=(Si\*m)/S   其中，b应该取整，它必须大于最小物理块数。

**3）考虑优先权的分配算法**

-- 在实际应用中，为了照顾到**重要的、紧迫的**作业能尽快地完成，应为它分配较多的内存空间。通常采取的方法是把内存中可供分配的所有物理块

分成两部分：一部分按比例地分配给各进程；另一部分则根据各进程的优先权，适当地增加其相应份额后，分配给各进程。

在有的系统中，如**重要的实时控制系统**，则可能是**完全按优先权**来为各进程分配其物理块的。

# 三、调页策略

为确定系统将运行时所缺的页面调入内存的时机，可采用以下两种调页策略：   
1）**预调页策略**。根据局部性原理，一次调入若干个相邻的页可能回避一次调入一页更高效。但如果调入的一批页面中大多数都未被访问，则又是低效的。所以就需要采用以预测为基础的预调页策略，将预计在不久之后便会访问的页面预先调入内存。但预调页的成功率仅约50%。故这种策略主要用于进程的首次调入时，由程序员指出应该先调入哪些页。   
2)**请求调页策略**   
进程在运行中需要访问的页面不在内存而提出请求，由系统将所需页面调入内存。由这种策略调入的页一定会被访问，且这种策略比较易于实现，故在目前的虚拟存储器中大多采用此策略。它的缺点在于一次只调入一页，调入调出页面数多时会花费过多的I/O开销。

# 四、LRU定义：

LRU是Least Recently Used的缩写，即最近最少使用页面置换[**算法**](http://lib.csdn.net/base/datastructure)，是为虚拟页式存储管理服务的，是根据页面调入内存后的使用情况进行决策了。由于无法预测各页面将来的使用情况，只能利用“最近的过去”作为“最近的将来”的近似，因此，LRU算法就是将最近最久未使用的页面予以淘汰。

实现方法：

1.用一个数组来存储数据，给每一个数据项标记一个访问时间戳，每次插入新数据项的时候，先把数组中存在的数据项的时间戳自增，并将新数据项的时间戳置为0并插入到数组中。每次访问数组中的数据项的时候，将被访问的数据项的时间戳置为0。当数组空间已满时，将时间戳最大的数据项淘汰。

2.利用一个链表来实现，每次新插入数据的时候将新数据插到链表的头部；每次缓存命中（即数据被访问），则将数据移到链表头部；那么当链表满的时候，就将链表尾部的数据丢弃。

可以用一个特殊的栈来保存当前正在使用的各个页面的页面号。当一个新的进程访问某页面时，便将该页面号压入栈顶，其他的页面号往栈底移，如果内存不够，则将栈底的页面号移除。这样，栈顶始终是最新被访问的页面的编号，而栈底则是最近最久未访问的页面的页面号。

**算例**

| **P:** | **2** | **3** | **2** | **1** | **5** | **2** | **4** | **5** | **3** | **2** | **5** | **2** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| M=3 | 2 | 3 | 2 | 1 | 5 | 2 | 4 | 5 | 3 | 2 | 5 | 2 |
|  |  | 2 | 3 | 2 | 1 | 5 | 2 | 4 | 5 | 3 | 2 | 5 |
|  |  |  |  | 3 | 2 | 1 | 5 | 2 | 4 | 5 | 3 | 3 |
|  |  |  |  |  | F |  | F |  | F | F |  |  |

### 缺页中断率

假定作业 p 共计n页，而系统分配给它的主存块只有 m 块（m,n均为正整数，且１≤m≤n），即最多主存中只能容纳该作业的m页。如果作业p在运行中成功的访问次数为s（即所访问的页在主存中）， 不成功的访问次数为F（即缺页中断次数），则总的访问次数Ａ为：   
A = S + F   
又定义：   
f = F / A   
则称 f 为缺页中断率。影响缺页中断率 f 的因素有：   
1. 主存页框数。作业分得的主存块数多，则缺页中断率就低，反之，缺页中断率就高。   
2. 页面大小。如果划分的页面大，则缺页中断率就低，否则缺页中断率就高   
3. 页面替换算法。   
4. 程序特性。程序编制的方法不同，对缺页中断的次数有很大影响,程序的局部性要好