页面置换算法

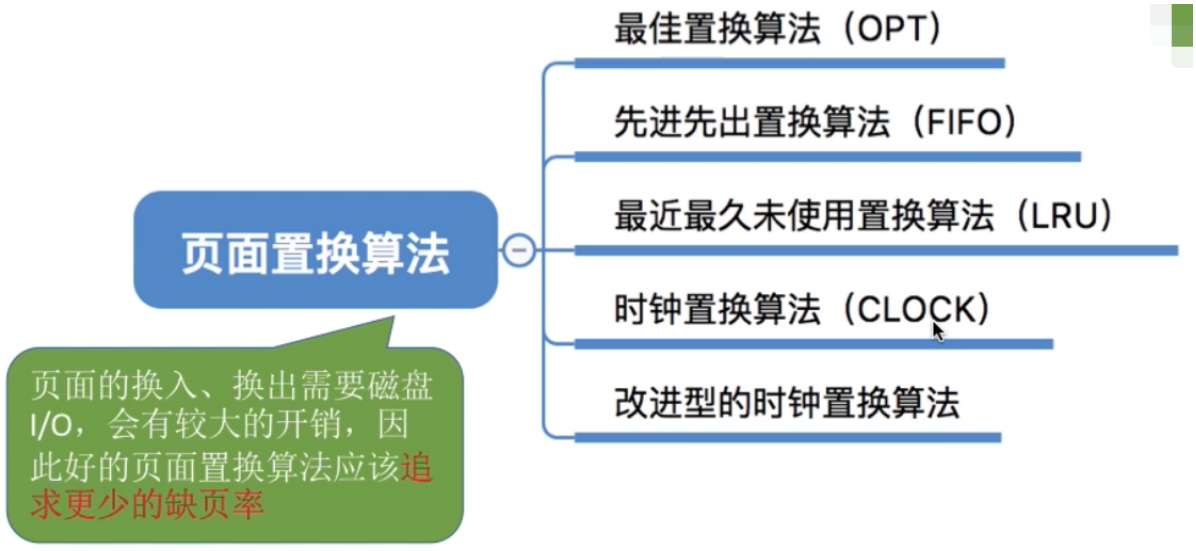
	算法规则	优缺点
OPT	优先淘汰最长时间内不会被访问的页面	缺页率最小，性能最好；但无法实现
FIFO	优先淘汰最先进入内存的页面	实现简单；但性能很差，可能出现Belady异常
LRU	优先淘汰最近最久没访问的页面	性能很好；但需要硬件支持，算法开销大
CLOCK（NRU）	循环扫描各页面 第一轮淘汰访问位=0的，并将扫描过的页面访问位改为1。若第一轮没选中，则进行第二轮扫描。	实现简单，算法开销小；但未考虑页面是否被修改过。
改进型CLOCK（改进型NRU）	若用（访问位,修改位）的形式表述，则 第一轮：淘汰（0,0） 第二轮：淘汰（0,1），并将扫描过的页面访问位都置为0 第三轮：淘汰（0,0） 第四轮：淘汰（0,1）	算法开销较小，性能也不错

请求分页存储管理与基本分页存储管理的主要区别：

在程序执行过程中，当所访问的信息不在内存时，由操作系统负责将所需信息从外存调入内存，然后继续执行程序。

若内存空间不够，由操作系统负责将内存中暂时用不到的信息换出到外存。

用页面置换算法决定应该换出哪个页面



最佳置换算法（OPT）

最佳置换算法（OPT, Optimal）：每次选择淘汰的页面将是以后永不使用，或者在最长时间内不再被访问的页面，这样可以保证最低的缺页率。

例：假设系统为某进程分配了三个内存块，并考虑到有以下页面号引用串（会依次访问这些页面）：

7，0，1，2，0，3，0，4，2，3，0，3，2，1，2，0，1，7，0，1

访问页面	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
内存块1	7	7	7	2		2		2			2			2				7		
内存块2		0	0	0		0		4			0			0				0		
内存块3			1	1		3		3			3			1				1		
是否缺页	√	√	√	√		√		√			√			√				√		

选择从0，1，7中淘汰一页。按最佳置换的规则，往后寻找，最后一个出现的页号就是要淘汰的页面

整个过程缺页中断发生了9次，页面置换发生了6次。

注意：缺页时未必发生页面置换。若还有可用的空闲内存块，就不用进行页面置换。

$$缺页率 = 9/20 = 45\%$$

最佳置换算法可以保证最低的缺页率，但实际上，只有在进程执行的过程中才能知道接下来会访问到的是哪个页面。操作系统无法提前预判页面访问序列。因此，最佳置换算法是无法实现的。

先进先出置换算法（FIFO）

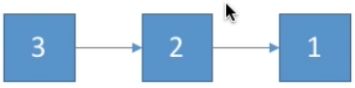
先进先出置换算法（FIFO）：每次选择淘汰的页面是最早进入内存的页面

实现方法：把调入内存的页面根据调入的先后顺序排成一个队列，需要换出页面时选择队头页面即可。队列的最大长度取决于系统为进程分配了多少个内存块。

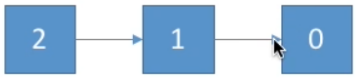
例：假设系统为某进程分配了三个内存块，并考虑到有以下页面号引用串：

3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4

访问页面	3	2	1	0	3	2	4	3	2	1	0	4
内存块1	3	3	3	0	0	0	4			4	4	
内存块2		2	2	2	3	3	3			1	1	
内存块3			1	1	1	2	2			2	0	
是否缺页	√	√	√	√	√	√	√			√	√	



访问页面	3	2	1	0	3	2	4	3	2	1	0	4
内存块1	3	3	3	0	0	0	4			4	4	
内存块2		2	2	2	3	3	3			1	1	
内存块3			1	1	1	2	2			2	0	
是否缺页	√	√	√	√	√	√	√			√	√	



分配三个内存块时，缺页次数：9次

例：假设系统为某进程分配了四个内存块，并考虑到有以下页面号引用串：
3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4

访问页面	3	2	1	0	3	2	4	3	2	1	0	4
内存块1	3	3	3	3			4	4	4	4	0	0
内存块2		2	2	2			2	3	3	3	3	4
内存块3			1	1			1	1	2	2	2	2
内存块4				0			0	0	0	1	1	1
是否缺页	√	√	√	√			√	√	√	√	√	√

分配四个内存块时，缺页次数：10次
分配三个内存块时，缺页次数：9次

Belady异常——当为进程分配的物理块数增大时，缺页次数不减反增的异常现象。

只有FIFO算法会产生Belady异常。另外，FIFO算法虽然实现简单，但是该算法与进程实际运行时的规律不适应，因为先进入的页面也有可能最经常被访问呢。因此，算法性能差

最近最久未使用置换算法（LRU）

最近最久未使用置换算法（LRU, least recently used）：每次淘汰的页面是最近最久未使用的页面

实现方法：赋予每个页面对应的页表项中，用访问字段记录该该页面自上次被访问以来所经历的时间t。

当需要淘汰一个页面时，选择现有页面中t值最大的，即最近最久未使用的页面。

例：假设系统为某进程分配了四个内存块，并考虑到有以下页面号引用串：
1, 8, 1, 7, 8, 2, 7, 2, 1, 8, 3, 8, 2, 1, 3, 1, 7, 1, 3, 7

访问页面	1	8	1	7	8	2	7	2	1	8	3	8	2	1	3	1	7	1	3	7
内存块1	1	1		1		1					1						1			
内存块2		8		8		8					8						7			
内存块3				7		7					3						3			
内存块4						2					2						2			
缺页否	√	√		√		√					√						√			

在手动做题时，若需要淘汰页面，可以逆向检查此时在内存中的几个页面号。在逆向扫描过程中最后一个出现的页号就是要淘汰的页面。

该算法的实现需要专门的硬件支持，虽然算法性能好，但是实现困难，开销大

时钟置换算法（CLOCK）

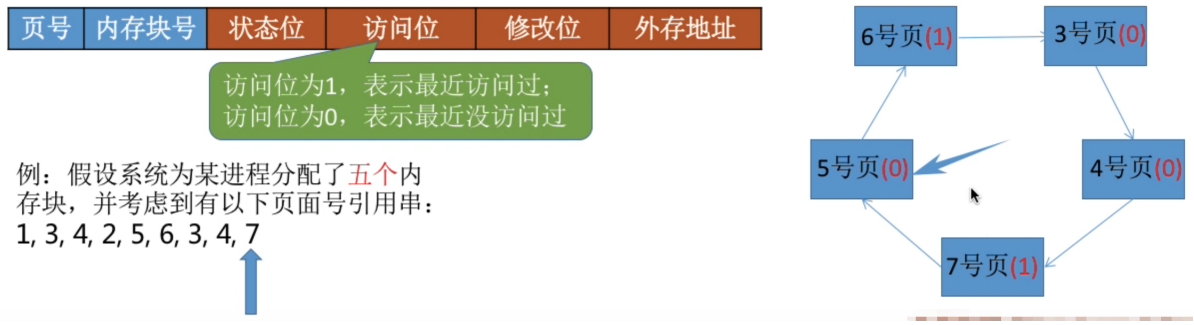
最佳置换算法性能最好，但无法实现；

先进先出置换算法实现简单，但性能最差；

最近最久未使用置换算法性能好，是最接近OPT算法性能的，但是实现起来需要专门的硬件支持，算法开销大。

时钟置换算法是一种性能和开销较均衡的算法，又称CLOCK算法，或最近未用算法（NRU, Not Recently Used）

简单的CLOCK算法实现方法：为每个页面设置一个访问位，再将内存中的页面都通过链接指针链接成一个循环队列。当某页被访问时，其访问位置为1。当需要淘汰一个页面时，只需检查页的访问位。如果是0，就选择该页换出；如果是1，则将它置为0，暂不换出，继续检查下一个页面，若第一轮扫描中所有页面都是1，则将这些页面的访问位依次置为0后，再进行第二轮扫描（第二轮扫描中一定会有访问未为0的页面，因此简单的CLOCK算法选择一个淘汰页面最多会经过两轮扫描）



改进型的时钟置换算法

简单的时钟置换算法仅考虑到一个页面最近是否被访问过。事实上，如果被淘汰的页面没有被修改过，就不需要执行I/O操作写回外存。只有被淘汰的页面被修改过时，才需要写回外存。

因此，除了考虑一个页面最近有没有被访问过之外，操作系统还应考虑页面有没有被修改过。在其他条件都相同时，应优先淘汰没有修改过的页面，避免I/O操作。这就是改进型的时钟置换算法的思想。

为方便讨论，用(访问位, 修改位)的形式表示各页面状态。如(1,1)表示一个页面近期被访问过，且被修改过。

算法规则：将所有可能被置换的页面排成一个循环队列

第一轮：从当前位置开始扫描到第一个(0,0)的帧用于替换。本轮扫描不修改任何标志位

第二轮：若第一轮扫描失败，则重新扫描，查找第一个(0,1)的帧用于替换。本轮将所有扫描过的帧访问位设为0

第三轮：若第二轮扫描失败，则重新扫描，查找第一个(0,0)的帧用于替换。本轮扫描不修改任何标志位

第四轮：若第三轮扫描失败，则重新扫描，查找第一个(0,1)的帧用于替换。

由于第二轮已将所有帧的访问位设为0，因此经过第三轮、第四轮扫描一定会有一个帧被选中，因此改进型CLOCK置换算法选择一个淘汰页面最多会进行四轮扫描

第一优先级：最近没访问，且没修改的页面

第二优先级：最近没访问，但修改过的页面

第三优先级：最近访问过，但没修改过的页面

第四优先级：最近访问过，且修改过的页面