进程运行时，若其访问的页面不在内存而需将其调入，但内存已无空闲空间时，就需要从内存中调出一页程序或数据，送入磁盘的对换区。  
  
选择调出页面的算法就称为页面置换算法。好的页面置换算法应有较低的页面更换频率，也就是说，应将以后不会再访问或者以后较长时间内不会再访问的页面先调出。

常见的置换算法有以下四种。

## 1. 最佳置换算法(OPT)

最佳(Optimal, OPT)置换算法所选择的被淘汰页面将是以后永不使用的，或者是在最长时间内不再被访问的页面,这样可以保证获得最低的缺页率。但由于人们目前无法预知进程在内存下的若千页面中哪个是未来最长时间内不再被访问的，因而该算法无法实现。  
  
最佳置换算法可以用来评价其他算法。假定系统为某进程分配了三个物理块，并考虑有以下页面号引用串：  
    7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1  
  
进程运行时，先将7, 0, 1三个页面依次装入内存。进程要访问页面2时，产生缺页中断，根据最佳置换算法，选择第18次访问才需调入的页面7予以淘汰。然后，访问页面0时，因为已在内存中所以不必产生缺页中断。访问页面3时又会根据最佳置换算法将页面1淘汰……依此类推，如图3-26所示。从图中可以看出釆用最佳置换算法时的情况。  
  
可以看到，发生缺页中断的次数为9，页面置换的次数为6。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 访问页面 | 7 | 0 | 1 | 2 | 0 | 3 | 0 | 4 | 2 | 3 | 0 | 3 | 2 | 1 | 2 | 0 | 1 | 7 | 0 | 1 |
| 物理块1 | 7 | 7 | 7 | 2 |  | 2 |  | 2 |  |  | 2 |  |  | 2 |  |  |  | 7 |  |  |
| 物理块2 |  | 0 | 0 | 0 |  | 0 |  | 4 |  |  | 0 |  |  | 0 |  |  |  | 0 |  |  |
| 物理块3 |  |  | 1 | 1 |  | 3 |  | 3 |  |  | 3 |  |  | 1 |  |  |  | 1 |  |  |
| 缺页否 | √ | √ | √ | √ |  | √ |  | √ |  |  | √ |  |  | √ |  |  |  | √ |  |  |

图3-26  利用最佳置换算法时的置换图

最佳置换算法进行了9次页面置换

2. 先进先出(FIFO)页面置换算法

优先淘汰最早进入内存的页面，亦即在内存中驻留时间最久的页面。该算法实现简单，只需把调入内存的页面根据先后次序链接成队列，设置一个指针总指向最早的页面。但该算法与进程实际运行时的规律不适应，因为在进程中，有的页面经常被访问。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 访问页面 | 7 | 0 | 1 | 2 | 0 | 3 | 0 | 4 | 2 | 3 | 0 | 3 | 2 | 1 | 2 | 0 | 1 | 7 | 0 | 1 |
| 物理块1 | 7 | 7 | 7 | 2 |  | 2 | 2 | 4 | 4 | 4 | 0 |  |  | 0 | 0 |  |  | 7 | 7 | 7 |
| 物理块2 |  | 0 | 0 | 0 |  | 3 | 3 | 3 | 2 | 2 | 2 |  |  | 1 | 1 |  |  | 1 | 0 | 0 |
| 物理块3 |  |  | 1 | 1 |  | 1 | 0 | 0 | 0 | 3 | 3 |  |  | 3 | 2 |  |  | 2 | 2 | 1 |
| 缺页否 | √ | √ | √ | √ |  | √ | √ | √ | √ | √ | √ |  |  | √ | √ |  |  | √ | √ | √ |

图3-27  利用FIFO置换算法时的置换图

这里仍用上面的实例，釆用FIFO算法进行页面置换。进程访问页面2时，把最早进入内存的页面7换出。然后访问页面3时，再把2, 0, 1中最先进入内存的页换出。由图 3-27可以看出，利用FIFO算法时进行了 15次页面置换，比最佳置换算法正好多一倍。  
  
FIFO算法还会产生当所分配的物理块数增大而页故障数不减反增的异常现象，这是由 Belady于1969年发现，故称为Belady异常，如图3-28所示。只有FIFO算法可能出现Belady 异常，而LRU和OPT算法永远不会出现Belady异常。



## 3. 最近最久未使用(LRU)置换算法

选择最近最长时间未访问过的页面予以淘汰，它认为过去一段时间内未访问过的页面，在最近的将来可能也不会被访问。该算法为每个页面设置一个访问字段，来记录页面自上次被访问以来所经历的时间，淘汰页面时选择现有页面中值最大的予以淘汰。  
  
再对上面的实例釆用LRU算法进行页面置换，如图3-29所示。进程第一次对页面2访问时，将最近最久未被访问的页面7置换出去。然后访问页面3时，将最近最久未使用的页面1换出。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 访问页面 | 7 | 0 | 1 | 2 | 0 | 3 | 0 | 4 | 2 | 3 | 0 | 3 | 2 | 1 | 2 | 0 | 1 | 7 | 0 | 1 |
| 物理块1 | 7 | 7 | 7 | 2 |  | 2 |  | 4 | 4 | 4 | 0 |  |  | 1 |  | 1 |  | 1 |  |  |
| 物理块2 |  | 0 | 0 | 0 |  | 0 |  | 0 | 0 | 3 | 3 |  |  | 3 |  | 0 |  | 0 |  |  |
| 物理块3 |  |  | 1 | 1 |  | 3 |  | 3 | 2 | 2 | 2 |  |  | 2 |  | 2 |  | 7 |  |  |
| 缺页否 | √ | √ | √ | √ |  | √ |  | √ | √ | √ | √ |  |  | √ |  | √ |  | √ |  |  |

图3-29  LRU页面置换算法时的置换图

在图3-29中，前5次置换的情况与最佳置换算法相同，但两种算法并无必然联系。实际上，LRU算法根据各页以前的情况，是“向前看”的，而最佳置换算法则根据各页以后的使用情况，是“向后看”的。  
  
LRU性能较好，但需要寄存器和栈的硬件支持。LRU是堆栈类的算法。理论上可以证明，堆栈类算法不可能出现Belady异常。FIFO算法基于队列实现，不是堆栈类算法。

## 4. 时钟(CLOCK)置换算法

## **4. 时钟(CLOCK)置换算法**

LRU算法的性能接近于OPT,但是实现起来比较困难，且开销大；FIFO算法实现简单，但性能差。所以操作系统的设计者尝试了很多算法，试图用比较小的开销接近LRU的性能，这类算法都是CLOCK算法的变体。  
  
简单的CLOCK算法是给每一帧关联一个附加位，称为使用位。当某一页首次装入主存时，该帧的使用位设置为1;当该页随后再被访问到时，它的使用位也被置为1。对于页替换算法，用于替换的候选帧集合看做一个循环缓冲区，并且有一个指针与之相关联。当某一页被替换时，该指针被设置成指向缓冲区中的下一帧。当需要替换一页时，操作系统扫描缓冲区，以查找使用位被置为0的一帧。每当遇到一个使用位为1的帧时，操作系统就将该位重新置为0；如果在这个过程开始时，缓冲区中所有帧的使用位均为0，则选择遇到的第一个帧替换；如果所有帧的使用位均为1,则指针在缓冲区中完整地循环一周，把所有使用位都置为0，并且停留在最初的位置上，替换该帧中的页。由于该算法循环地检查各页面的情况，故称为CLOCK算法，又称为最近未用(Not Recently Used, NRU)算法。  
  
CLOCK算法的性能比较接近LRU，而通过增加使用的位数目，可以使得CLOCK算法更加高效。在使用位的基础上再增加一个修改位，则得到改进型的CLOCK置换算法。这样，每一帧都处于以下四种情况之一：

1. 最近未被访问，也未被修改(u=0, m=0)。
2. 最近被访问，但未被修改(u=1, m=0)。
3. 最近未被访问，但被修改(u=0, m=1)。
4. 最近被访问，被修改(u=1, m=1)。

算法执行如下操作步骤：

1. 从指针的当前位置开始，扫描帧缓冲区。在这次扫描过程中，对使用位不做任何修改。选择遇到的第一个帧(u=0, m=0)用于替换。
2. 如果第1)步失败，则重新扫描，查找(u=0, m=1)的帧。选择遇到的第一个这样的帧用于替换。在这个扫描过程中，对每个跳过的帧，把它的使用位设置成0。
3. 如果第2)步失败，指针将回到它的最初位置，并且集合中所有帧的使用位均为0。重复第1步，并且如果有必要，重复第2步。这样将可以找到供替换的帧。

改进型的CLOCK算法优于简单CLOCK算法之处在于替换时首选没有变化的页。由于修改过的页在被替换之前必须写回，因而这样做会节省时间。

例题：

\*在5个页框上使用LRU页面替换算法，当页框初始为空时，引用序列为0、1、7、8、6、2、3、7、2、9、8、1、0、2，系统将发生（C）次缺页

    A、13            B、12           C、11          D、8

解析：内存中驻留5个页框：



 LRU是堆栈类的算法，最后访问的页面放在栈顶，可以得到答案为C。