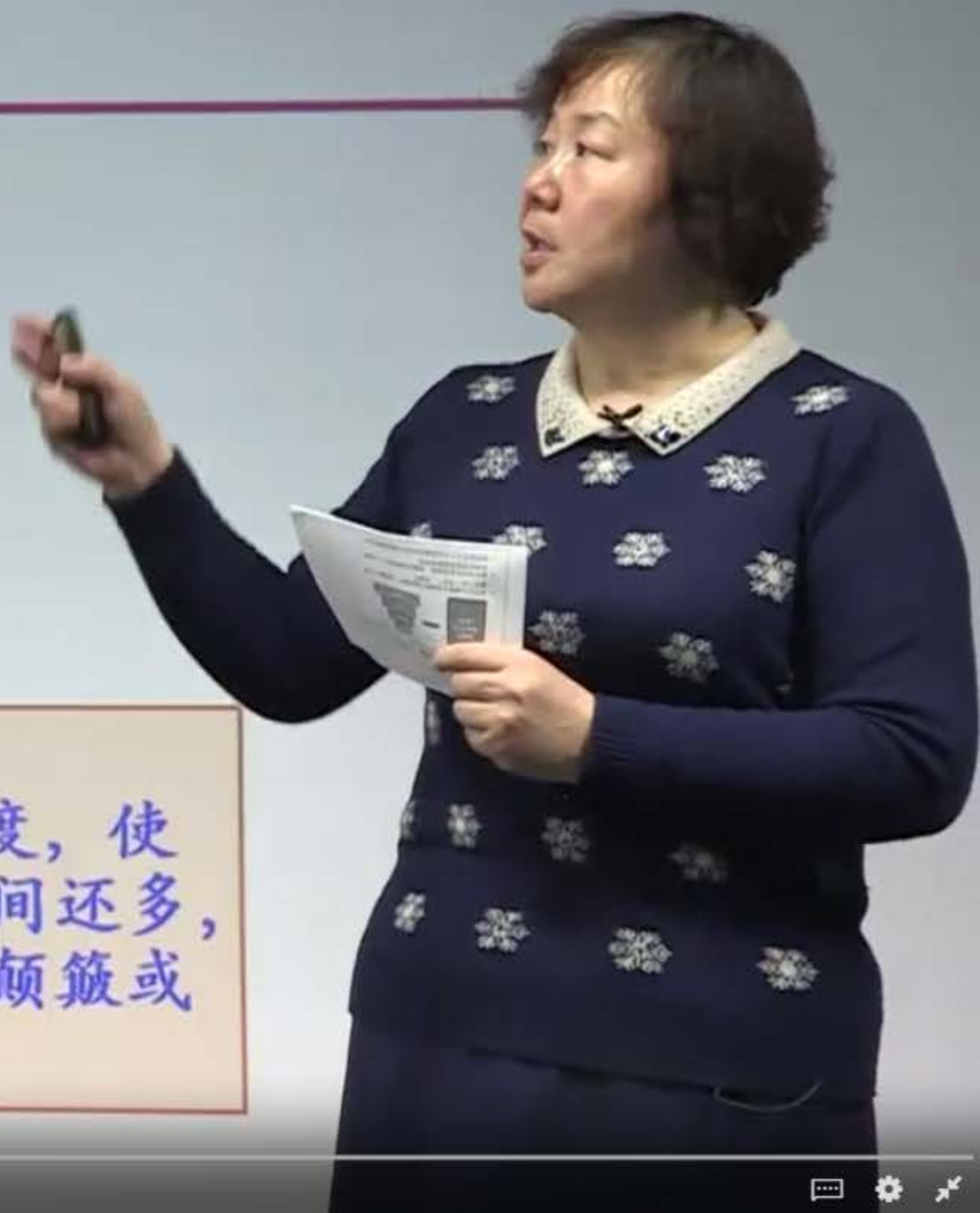
影响缺页次数的因素

- 页面置换算法
- 页面本身的大小√
- 程序的编制方法√
- 分配给进程的页框数量√

颠簸（Thrashing，抖动）

虚存中，页面在内存与磁盘之间频繁调度，使得调度页面所需的时间比进程实际运行的时间还多，这样导致系统效率急剧下降，这种现象称为颠簸或抖动



页面尺寸问题

- 确定页面大小对于分页的硬件设计非常重要
  而对于操作系统是个可选的参数

- 要考虑的因素：
  - 内部碎片
  - 页表长度
  - 辅存的物理特性

小页面?
大页面?

最优页面大小
$P=\sqrt{2se}$

- Intel 80x86/Pentium： 4096 或 4M

- 多种页面尺寸：为有效使用TLB带来灵活性，但
  给操作系统带来复杂性

程序编制方法对缺页次数的影响

例子：分配了一个页框；页面大小为128个整数；
矩阵$A_{128X128}$按行存放

程序编制方法1：

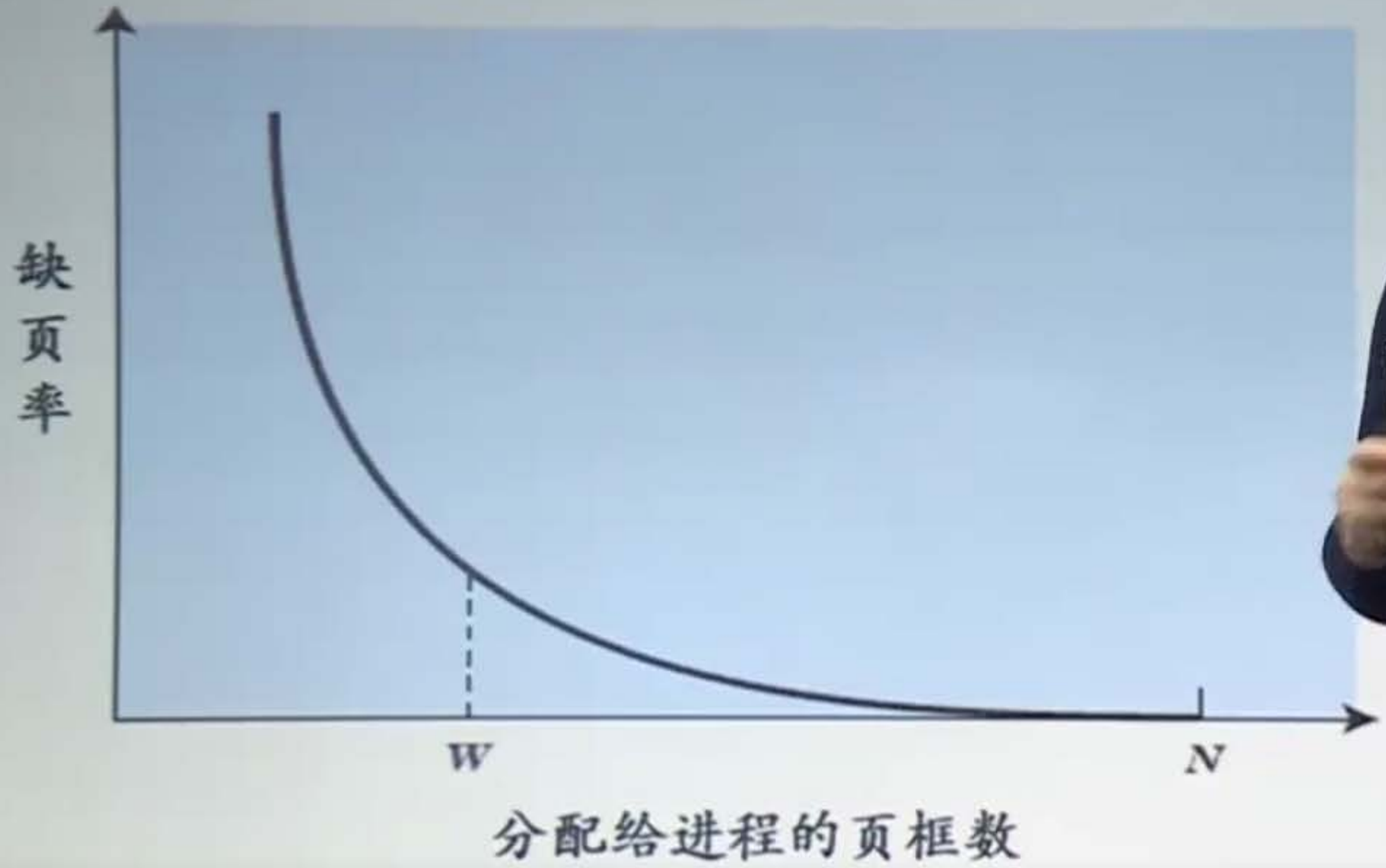
```
for j:=1 to 128
    for i:=1 to 128
        A[i,j]:=0;
```

程序编制方法2：

```
for i:=1 to 128
    for j:=1 to 128
        A[i,j]:=0;
```

缺页
几次？

分配给进程的页框数与缺页率的关系



工作集(WORKING SET)模型(1/3)

基本思想：

根据程序的局部性原理，一般情况下，进程在一段时间内总是集中访问一些页面，这些页面称为活跃页面，如果分配给一个进程的物理页面数太少了，使该进程所需的活跃页面不能全部装入内存，则进程在运行过程中将频繁发生中断

如果能为进程提供与活跃页面数相等的物理页面数，则可减少缺页中断次数

由Denning提出(1968)

工作集(WORKING SET)模型(2/3)

工作集：一个进程当前正在使用的页框集合

工作集$W(t, \Delta)$

= 该进程在过去的Δ个虚拟时间单位中访问到的页面的集合

内容取决于三个因素：

- 访页序列特性
- 时刻t
- 工作集窗口长度(Δ)

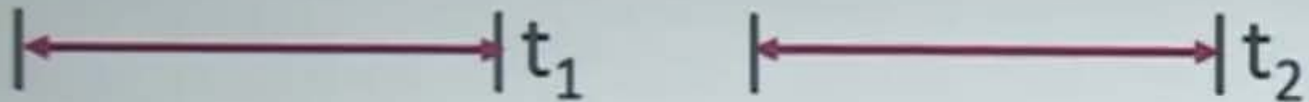
窗口越大，工作集就越大

那么工作集就越大，所以选择一个合适的工作窗口来计算当前的工作集

工作集(WORKING SET)模型(3/3)

例：

2615777516234123444343444 1327



$W(t_1,10)=\{1,2,5,6,7\}$

$W(t_2,10)=\{3,4\}$

所以活跃页面

随着时间的这个进展，活跃页面的数量呢是不一样的，所以工作集呢是需要随时调整的

工作集算法(1/2)

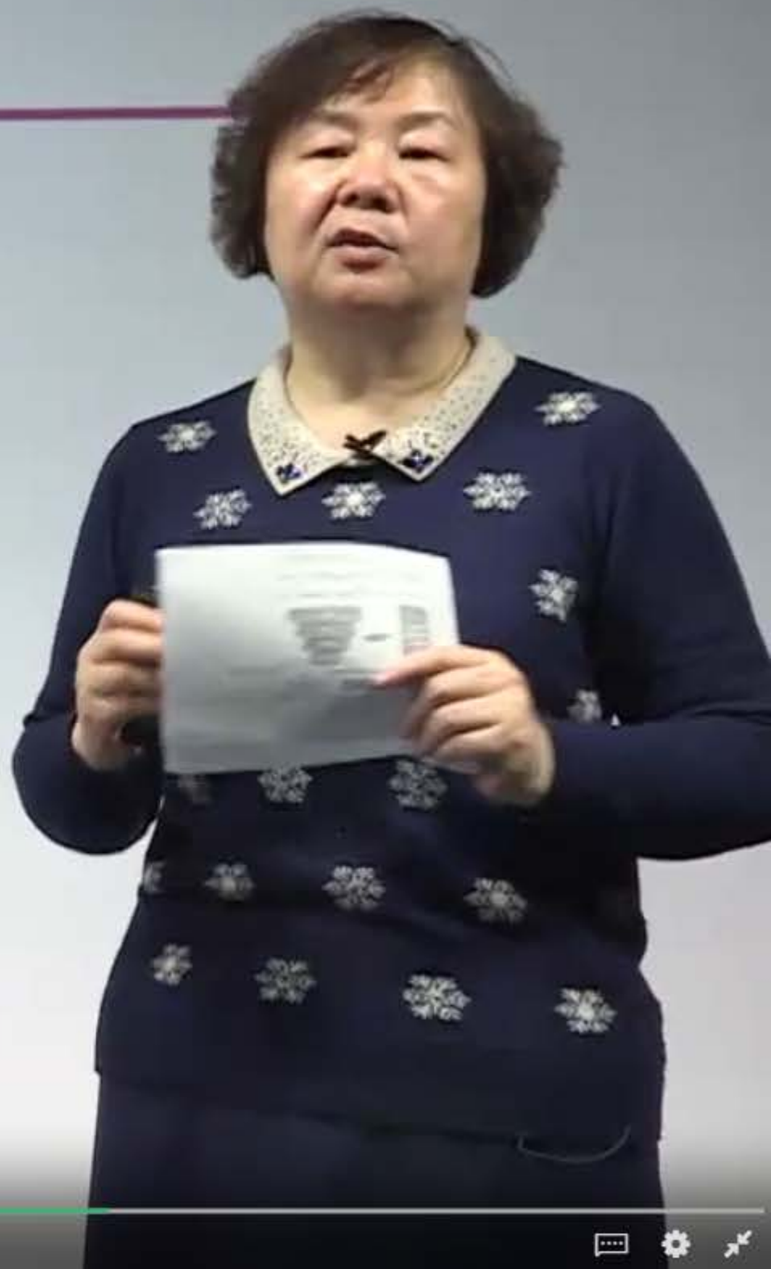
基本思路：

找出一个不在工作集中的页面并置换它

思路：

- 每个页表项中有一个字段：记录该页面最后一次被访问的时间
- 设置一个时间值T
- 判断：

根据一个页面的访问时间是否落在“当前时间-T”之前或之中决定其在工作集之外还是之内



工作集算法(2/2)

实现：

扫描所有页表项，执行操作

1. 如果一个页面的R位是1，则将该页面的最后一次访问时间设为当前时间，将R位清零
2. 如果一个页面的R位是0，则检查该页面的访问时间是否在“当前时间-T”之前

(1) 如果是，则该页面为被置换的页面；

(2) 如果不是，记录当前所有被扫描过页面的最后访问时间里面的最小值。扫描下一个页面并重复1、2

页面置换算法小结

算法	评价
OPT	不可实现，但可作为基准
NRU	LRU的很粗略的近似
FIFO	可能置换出重要的页面
Second Chance	比FIFO有很大的改善
Clock	实现的
LRU	很优秀，但很难实现
NFU	LRU的相对粗略的近似
Aging	非常近似LRU的有效算法
Working set	实现起来开销很大

那么工作集的算法呢实现起来开销很大，但是 呢，很多操作系统呢也采用了这种算法

0:00 那么下面我们来介绍一个新的页面置换算法，就是工作集算法。 我们首先来分析一下影响缺页次数的 因素。 第一个因素呢是页面置换算法。 刚才我们应用了三种页面置换算法，对同一个页面走向序列进行计算， 那么发现它的缺页次数都是不一样的。 因此我们知道页面置换算法的不同，那么 缺页的次数可能会不一样的。 第二个影响 缺页次数的这个因素呢就是页面本身的大小。 第三个呢是程序的编制方法。 而第四个呢是分配给进程的页框数量。 那么缺页越多，系统的性能就会越差， 那么我们就是用一个名词来形容它，就是颠簸和抖动。 所谓颠簸和抖动就指的在虚存当中，页面呢在内存与磁盘之间是频繁地调度， 使得这个进程的运行的实际时间都比 页面调度在磁盘和内存之间调度所花的时间都 少了，等于说磁盘调度花了大量的时间， 这样的话呢系统的性能、 效率肯定是下降， 那么出现的这种现象我们称之为颠簸和抖动。 因此缺页次数越多，那么就会出现这样一种现象。 那么页面尺寸肯定也会影响 缺页的次数。 为什么呢？因为 确定页面的大小对于分页系统的硬件设计是非常重要的， 那么对于操作系统来讲呢它是一个可选的参数。 我可以选择，你给我提供不同的尺寸 我可以根据你提供的尺寸来选择我的一个管理的尺寸，所以对操作系统而言也是个可选的参数。 那么在 Intel 体系当中呢，我们可以 看到它有两种尺寸的页面大小，一个是 4k，一个是 4 兆。 甚至有一些计算机系统呢会提供多种页面尺寸。 提供多种页面尺寸呢会为有效灵活地来使用 快表提供了一种灵活性，但是呢对于操作系统来讲呢带来了一定的复杂性。 那么页面尺寸到底多大比较合适呢？ 其实它和几个因素有关，比如说碎片问题。 我们知道页面太大，那么碎片就可能，内碎片就可能会增大。 那么页表长度的问题。 页面如果很小， 页表项就会很多，页表就会很大。 另外呢我们还有这个 磁盘的一些物理特性。 我们要综合考虑这个因素呢来 解决这个页面尺寸该多大。 那么如果我们的磁盘扇区大小或者是 簇的大小和页面大小成什么样的一个比例更合适，那么我们的 也要考虑这样的因素。 通常情况下呢， 大页面还是小页面的问题，有一些 比如操作系统它可能选择一些大页面，有一些数据结构它就换进换出 内存的话，会少一些。 那么小页面呢换进换出内存会多一些。 那么最优的页面大小呢往往是什么呢？有这么一个公式，经验的 一个公式或统计的一个结果。 那么 s 呢是一个进程的一个平均的 这个规模，那么这个 e 呢实际上是页表项的长度，所以我们可以算出一个最优的页面大小。 但是呢一般的计算机系统呢会采用一些 常用的结构尺寸，也就是 4k 啊、 4 兆这样一个尺寸。 那么下面我们再来探讨另外一个 因素，影响缺页次数的因素，就是程序的编制方法。

我们假设有一个例子，就给这个进程分配了一个页框， 这个大小呢是 128 个整数，那么有一个矩阵，这个矩阵是 128×128 的矩阵，这个矩阵的数值呢实际上是按照行来存放的，先放矩阵第一行， 再放矩阵第二行，其实一行正好是一页， 那么也就是说这个矩阵一共是 128 页。 那我们有两种编制程序的方法，大家都很熟悉，左边这个方法呢 是按列赋值，初始化，给这个矩阵 A 初始化，是按列 赋值。 先给第一列赋完值之后再给第二列赋值。 而右边这种编制方法呢是按行赋值，按行赋值。 那我们可以算一下这两个不同的 程序编制方法在运行过程中会产生的缺页次数。 很显然，右边这种编制方法 每次给一行所有的这个元素赋值， 那么我们正好一行正好是一页，所以把这一页读入内存，把它全赋完值这一行 就都赋完了。 然后再读入下一行，因为我们只有这一个页框， 那再把下一个页框读入内存，把前面的就给它覆盖掉，或者把前面那个换出去， 所以呢就给第二行赋值。 那么一共呢 需要呢 128 行，那么初始的时候呢， 初始的时候没有说，那初始时候假设是 为空的话，那么就需要产生 128 次的缺页异常。 那么左边这个大家会看到每次呢按列赋值，也就是 假设按这个来讲，当把第一页就因为 这个矩阵是按行存放，正好一页正好是这个 128 整数，所以 当给这个矩阵第一个元素赋值的时候呢，实际上是把这个矩阵的第一行读进内存，也就相当于- 把第一页读进内存。 但是读进内存，它只给第一个元素赋值，后面那个没有赋值。 那么给第一行第一个元素赋完值之后呢，就要给 第二行第一个元素赋值，那这个时候呢就要把第二页调进内存。 然后给第三行第一个元素赋值，当把第一列 128 个元素都赋完值之后呢就要给第一行的第二个元素赋值，可是这个时候呢又要把 这个矩阵的这一行读进内存，也就是把这矩阵的这一页读进内存。 因此我们知道左边的这个程序编制方法 导致的缺页次数呢就会达到 128 的平方，所以相差 很多，相差很多。 所以程序的编制方法实际上对缺页次数是有影响的。 那么下面我们看看第三个因素就是 分配给进程的页框数与缺页率的关系。 前面其实我们 在讲的过程中其实已经看到这样一个，我分配给进程的页框数越多， 它的缺页率就会越下降，越低；如果你所需要的页框我都给你了，那就没有缺页了。 这是这张图给出的这样一个示例。 但是呢 我们前面也讲过了，我们不可能给一个进程，它要多少给多少，我们要给它一个驻留集的概念- ，就是说 分配它一定数量的页框。 那么分配多少比较合适呢？ 那我们要考虑到在里头图的这一点，那么这一点是一个平衡点， 那么给它更多的页框它的缺页率降低得不是很多，

如果少于这个点，那么缺页率 直线地上升，所以我们要找到这样一个平衡点，这也就是我们下一个页面置换算法 它的一个基本的出发点。 我们下面来介绍工作集模型。 那么工作集模型呢 它的基本思想也是根据程序的一个局部性的原理。 由于程序在执行过程中总是能够集中在一段 时间内集中访问一些页面，那么我们就把这些页面呢称之为活跃页面。 如果我们分配给它的物理页面数或者页框少于这个 活跃页面所需要的页框数，以致于这些活跃页面不能全部装入内存， 那就会导致一个现象，就是进程在运行 过程中呢会频繁地产生中断，产生缺页。 如果我能够预测出来这种活跃页面数，而 为进程提供与这个活跃页面数相匹配的 这样一个页框，页框数量，那么就可以减少缺页中断的产生。 那么这就是这个工作集思想的一个基本的思路。 那么由丹宁在这个 1968 年提出了这样一个工作集模型。 那么所谓工作集呢我们现在来看，给工作集下个 定义，其实就是一个进程当前正在使用的页框的集合。 其实也就是驻留集的这样一个基本思路，但是呢 工作集是需要随时调整的，驻留集就分为它了 那么工作集是需要随时调整的。 所以我们要计算当时的一个 当前的一个工作集是多少，那么我们来看一下，那么 我如果给了一个时间 t，给了一个时间间隔 Δ 那么我们就可以计算出来这个进程在过去的这么 若干个时间单位之内访问到页面的集合，我们就把它作为时间 t 的进程的工作集 因此工作集的 内容取决于你的页面访问的特性 某一个时刻和工作集的窗口长度，所以我这个窗口越长 那么工作集就越大，所以选择一个合适的工作窗口来计算当前的工作集 我们举一个例子，这有一串页面访问序列，我们假设有 两个时刻 t1 和 t2，工作集的窗口呢是 10 个，10 个页面的窗口 当然是很小了，实际上不会这么小，我们当时为了也画不下去，画不下来了这个 屏幕很小，所以我们就取了 10 个这样一个窗口，长度是 10 我们来看看 t1 时刻，落入这个工作集的有哪些页面呢？ 1,2,5,6,7，要 5 个页面 而在 t2 时刻，其实呢只需要 2 个页面。 所以活跃页面 随着时间的这个进展，活跃页面的数量呢是不一样的，所以工作集呢是需要随时调整的 下面我们介绍基于上述的工作集模型 设计的一个工作集算法。 工作集算法的基本思想是 找出一个不在工作集的页面，把它置换掉，因为一个进程 它的驻留集里头有一些 页面是不在工作集里头的，所以我们要找出一个不在工作集的页面把它置换出去 具体的思路是这样的，那么每个页表项呢我增加一个字段，记录了这个 页面最后一次被访问的时间。 然后我设置一个时间值 T 这个时间值 T 就相当于我们的工作集窗口的作用，然后我就去

判断一下 根据一个页面的访问时间，判断一下它这个时间是落在了当前时间 减去刚才我们设定的这个时间值 T，是落在了这个时间的范围 之内，还是之外？是之前，是之中？ 如果落在了之中，那么我们就把它认为是在工作集里头的 如果是之前，也就是前于 当前时间减去 T，小于它，那么就认为是落在了这个工作集之外，一个是工作集之内，一个是工作集之外 具体的有一样这样一个实现 扫描所有的页表项，执行如下操作。 那么当然这里头 还把其他的一些思想揉进来。 如果一个页面的 R 位是 1 的话 那么则将该页面的最后一次访问时间设置为当前时间，然后 R 位清零 如果要把选择的一个页面，它的 R 位是 0 那么就要去检查它是不是在刚才说的落入工作 集之内，还是落在了工作集之外，要检查刚才这个 内容。 如果是落在了它的 之前，那么就这个页面就要被置换出去了，这就是落在工作集之外了 如果是落在之中，那就说明它不是要淘汰的 所以它是一个落在工作集里头的，是属于工作集的，那就要记录 当前所有被扫描过页面当中的最后 访问时间当中的那个最小的那个值，就把那最小值选取出来 然后接着去扫描，这就是一个工作集算法的一个基本思想 那么 Windows 操作系统呢实际上呢就是采用了这种工作集的思想 但是它的实现呢和这个呢都有一些区别。 它的做法是这样的，就是 当进程创建的时候呢，它会给每一个进程一个 页框的一个范围，也就是工作集的范围 最少要多少页框，最多多少页框 它这个进程的运行过程中不断地填充这些页框 当它超过了，达到了它的最大的页框数的时候呢 ，当然这个进程需要这个 我们说这个进程需要这个新的页框的时候呢，那么 如果系统有足够的空闲页框，那么 Windows 还是会分配给这进程 更多的页框的，是不受到最大值的一个限制 但是如果系统中的空闲页框数少了 降低到一定的程度，也就是说我设定了一个 阈值，那么系统中空闲页框的数量低于这个阈值了 那这个时候，在 Windows 里头有一个平衡级管理器，那么它是每秒钟这个 工作一次，其实定期出来工作。 前面我们在讲这个 这个一些算法的时候，我们在讲 Windows 算法的时候呢，实际上我们在讲 这个如果有饥饿现象发生，那么平衡级管理器会去临时 提升这些饥饿线程的优先级，其实也是这个平衡级管理器的工作 那么我们现在也是它的工作。 所以这个平衡级管理器就 1 秒钟 比如出来工作一次，它就去检查一下说，我现在 系统的这个空闲页框数少了，那么这个时候就开始启动 清除策略，就启动页面置换策略来清除 把那些超过了最大值的这样的这个 页框数的这样一些进程，它们的这些页框可以回收回来，回收回来 这就是前面我们所说的

清除策略的这样一个工作过程，所以在 Windows 当中，用的是工作集的思想，那么它的清除过程呢是用我们前面所介绍的那种 页缓冲的技术。 下面我们总结一下页面置换算法 OPT 算法呢虽然不可以实现，但可以作为衡量其他 页面置换算法的一个基准。 FIFO 算法呢可能置换出一些 重要的页面，那么第二次机会算法呢对它进行了相应的改进 时钟算法其实是从实现角度考虑的，为了加快 速度，提高性能。 LRU 算法是优秀的算法，但它的实现 是很难的，或者是开销非常大的，因此通常都是采用一些近似的 实现 LRU 算法。 那么 老化算法就是一种 LRU 算法的一个近似的实现 那么工作集的算法呢实现起来开销很大，但是 呢，很多操作系统呢也采用了这种算法