页面置换(REPLACEMENT)算法

又称页面淘汰（替换）算法

最佳算法→先进先出→第二次机会→时钟算法→
最近未使用→最近最少使用→最不经常使用→老
化算法→工作集→工作集时钟

设计思想　算法实现　算法应用

那么有一些 页面置换算法呢希望大家能够知道它是如何来实现的

最佳页面置换算法（OPT）

- 设计思想：
  置换以后不再需要的或最远的将来才会用到的页面

- 实现？
- 作用？

作为一种标准来衡量其他算法的性能

最佳页面置换算法，那么这些算法就是好的算法 这就是它的主要的作用

先进先出算法（FIFO）

- 选择在内存中驻留时间最长的页并置换它

对照：超市撤换商品

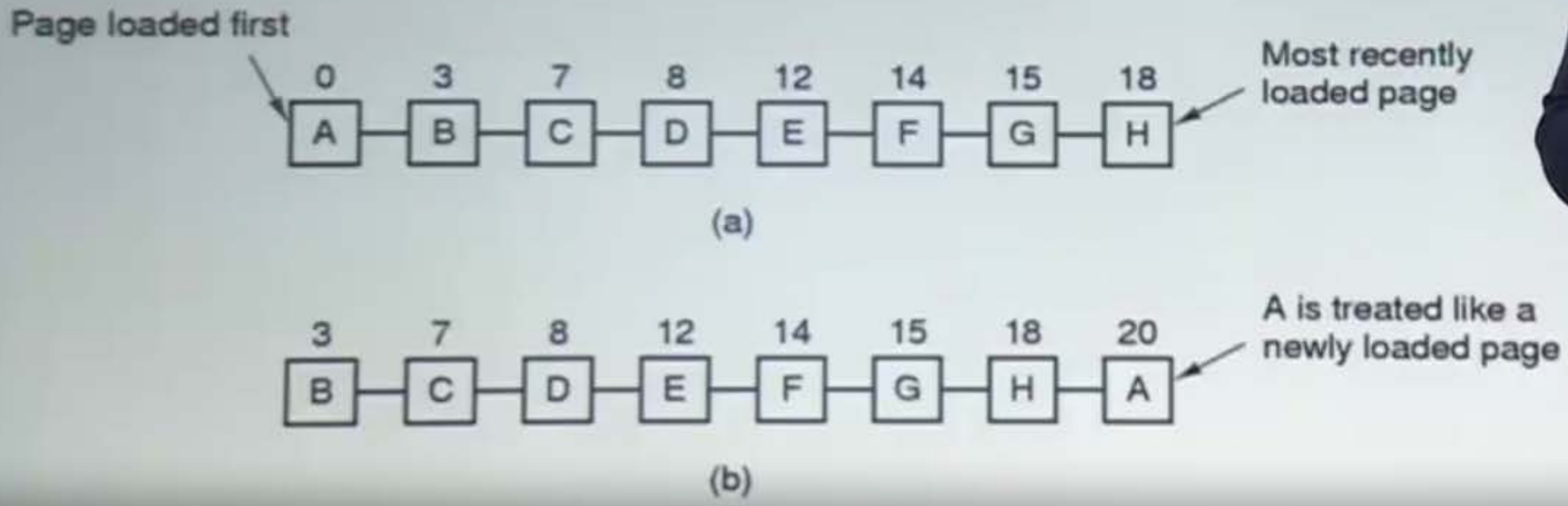
- 实现：页面链表法

这是先进先出页面置换算法。

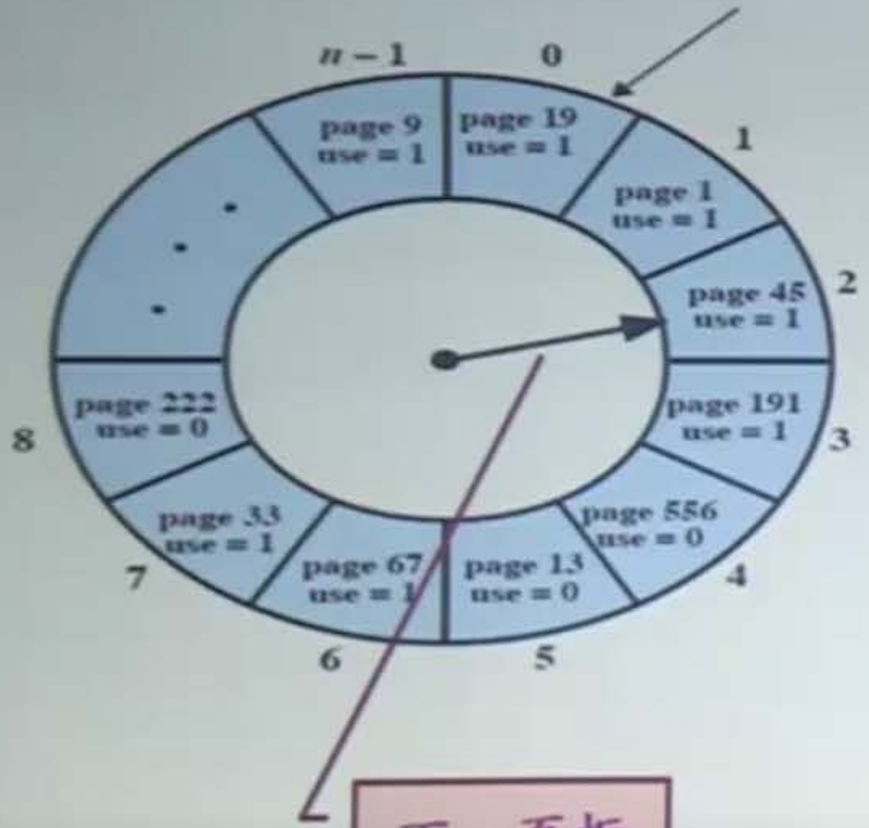
第二次机会算法(SCR)

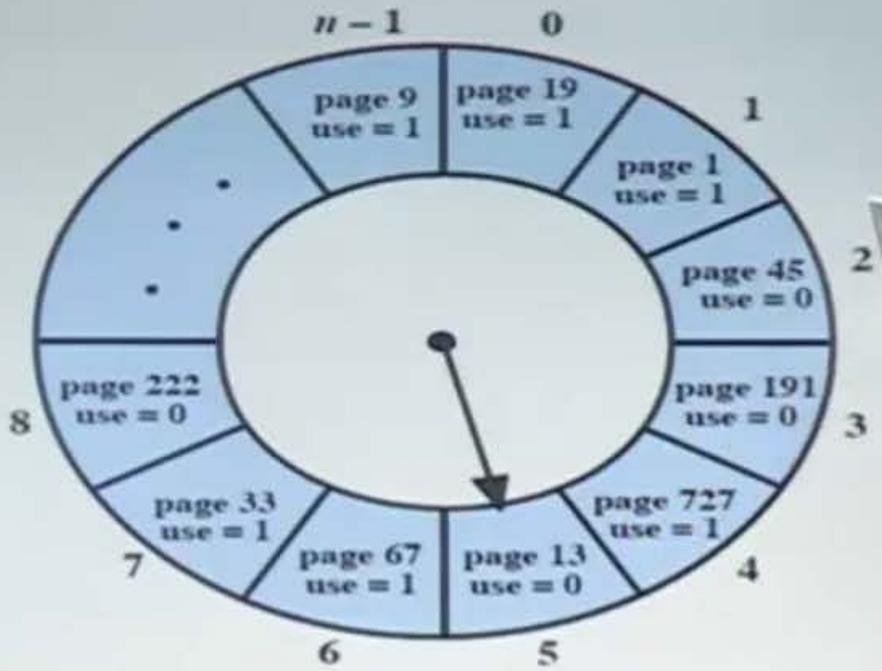
SCR-Second Chance

按照先进先出算法选择某一页面，检查其访问位R，如果为0，则置换该页；如果为1，则给第二次机会，并将访问位置0



时钟算法(CLOCK)


n − 1
0
1
2
3
4
5
6
7
8
page 9 use = 1
page 19 use = 1
page 1 use = 1
page 45 use = 1
page 191 use = 1
page 556 use = 0
page 13 use = 0
page 67 use = 1
page 33 use = 1
page 222 use = 0
下一页框
指针


n − 1
0
1
2
3
4
5
6
7
8
page 9 use = 1
page 19 use = 1
page 1 use = 1
page 45 use = 0
page 191 use = 0
page 727 use = 1
page 13 use = 0
page 67 use = 1
page 33 use = 1
page 222 use = 0

最近未使用算法(NRU)(1/2)

Not Recently Used

选择在最近一段时间内未使用过的一页并置换

实现：设置页表表项的两位

访问位（R），修改位（M）

如果硬件没有这些位，则可用软件模拟（做标记）

启动一个进程时，R、M位置0

R位被定期清零（复位）

这样的话呢，如果很长时间内它没有被使用 过，它实际上就清完零，那以后就不是 1

最近未使用算法(NRU)(2/2)

发生缺页中断时，操作系统检查R，M：

第1类：无访问，无修改

第2类：无访问，有修改

第3类：有访问，无修改

第4类：有访问，有修改

算法思想：

随机从编号最小的非空类中选择一页置换

随机选择一个置换出去，好，这是最近未使用的这样一个页面置换算法

时钟算法实现

1. 从指针的当前位置开始，扫描页框缓冲区，选择遇到的第一个页框（*r*=0；*m*=0）用于置换(本扫描过程中，对使用位不做任何修改)
2. 如果第1步失败，则重新扫描，选择第一个（*r*=0，*m*=1）的页框(本次扫描过程中，对每个跳过的页框，将其使用位设置成0)
3. 如果第2步失败，指针将回到它的最初位置，并且集合中所有页框的使用位均为0。重复第1步，并且，如果有必要，重复第2步。这样将可以找到供置换的页框

当然我们 就是想说，诶，这个算法可以用时钟算法来实现，啊实现

最近最少使用算法(LRU)

Least Recently Used

选择最后一次访问时间距离当前时间最长的一页并置换

即置换未使用时间最长的一页

- 性能接近OPT
- 实现：时间戳 或 维护一个访问页的栈

→ 开销大

栈，栈，也都可以，但是呢总体来讲，开销比较大

LRU算法的一种硬件实现

- 页面访问顺序0, 1, 2, 3, 2, 1, 0, 3, 2, 3

Page

	0	1	2	3
0	0	1	1	1
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0

(a)

Page

0	1	2	3
0	0	1	1
1	0	1	1
0	0	0	0
0	0	0	0

(b)

Page

0	1	2	3
0	0	0	1
1	0	0	1
1	1	0	1
0	0	0	0

(c)

Page

0	1	2	3
0	0	0	0
1	0	0	0
1	1	0	0
1	1	1	0

(d)

Page

0	1	2	3
0	0	0	0
1	0	0	0
1	1	0	1
1	1	0	0

(e)

0	0	0	0
1	0	1	1
1	0	0	1
1	0	0	0

(f)

0	1	1	1
0	0	1	1
0	0	0	1
0	0	0	0

(g)

0	1	1	0
0	0	1	0
0	0	0	0
1	1	1	0

(h)

0	1	0	0
0	0	0	0
1	1	0	1
1	1	0	0

(i)

0	1	0	0
0	0	0	0
1	1	0	0
1	1	1	0

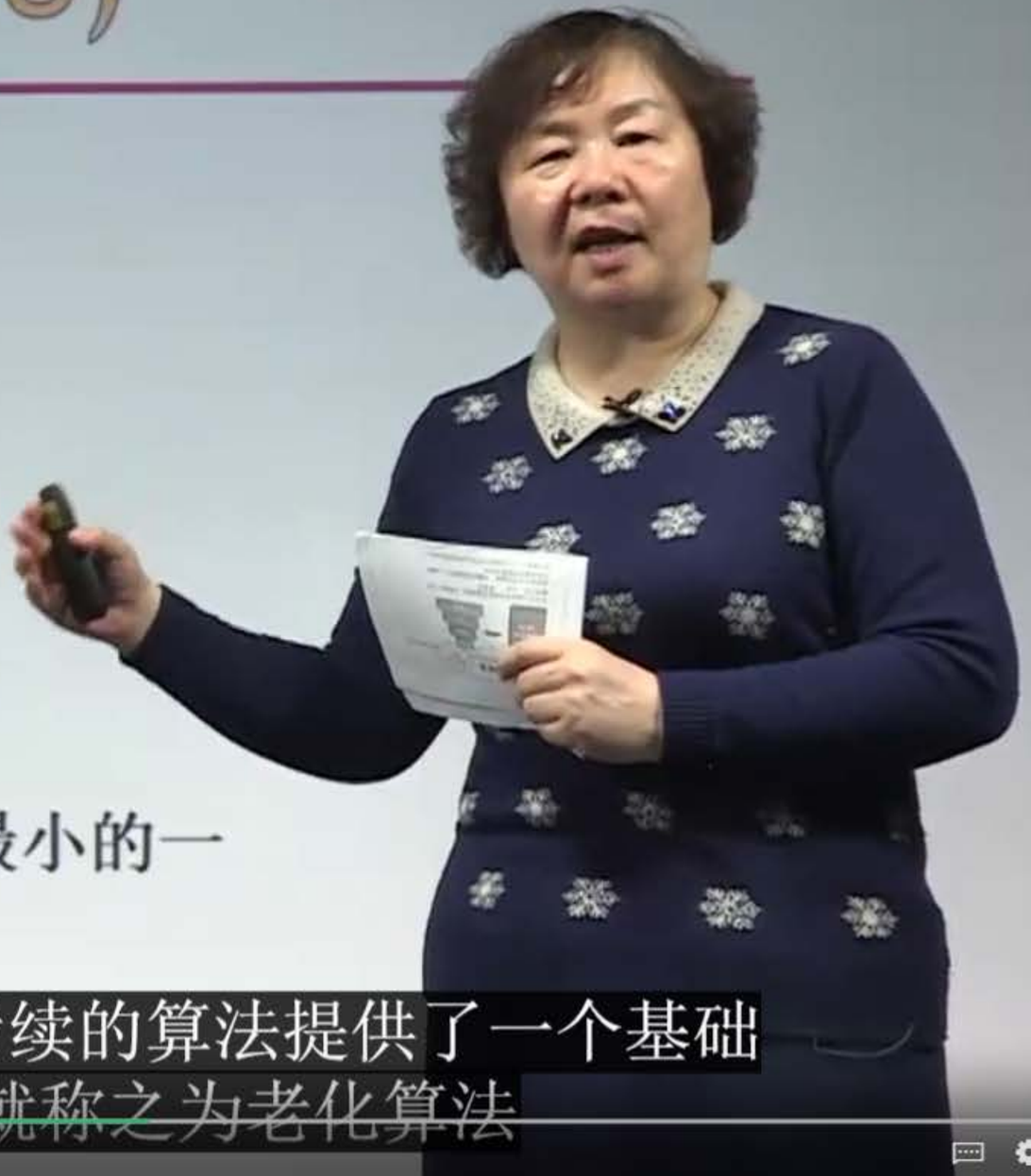
(j)

最不经常使用算法(NFU)

Not Frequently Used

选择访问次数最少的页面置换

- LRU的一种软件解决方案
- 实现：
  - 软件计数器，一页一个，初值为0
  - 每次时钟中断时，计数器加R
  - 发生缺页中断时，选择计数器值最小的一页置换



的还比较远，但是呢，给我们后续的算法提供了一个基础
而我们后续的算法呢，就称之为老化算法

老化算法(AGING)

思考：与LRU的区别

改进（模拟LRU）：计数器在加R前先右移一位

R位加到计数器的最左端

Page	R bits for pages 0-5, clock tick 0	R bits for pages 0-5, clock tick 1	R bits for pages 0-5, clock tick 2	R bits for pages 0-5, clock tick 3	R bits for pages 0-5, clock tick 4
	1 0 1 0 1 1	1 1 0 0 1 0	1 1 0 1 0 1	1 0 0 0 1 0	0 1 1 0 0 0
0	10000000	11000000	11100000	11110000	01111000
1	00000000	10000000	11000000	01100000	10110000
2	10000000	01000000	00100000	00100000	10010000
3	00000000	00000000	10000000	01000000	00100000
4	10000000	11000000	01100000	10110000	01011000
5	10000000	01000000	10100000	01010000	00101000
	(a)	(b)	(c)	(d)	(e)

页面置换算法的应用

例子：

- 系统给某进程分配3个页框(固定分配策略)，初始为空
- 进程执行时，页面访问顺序为：

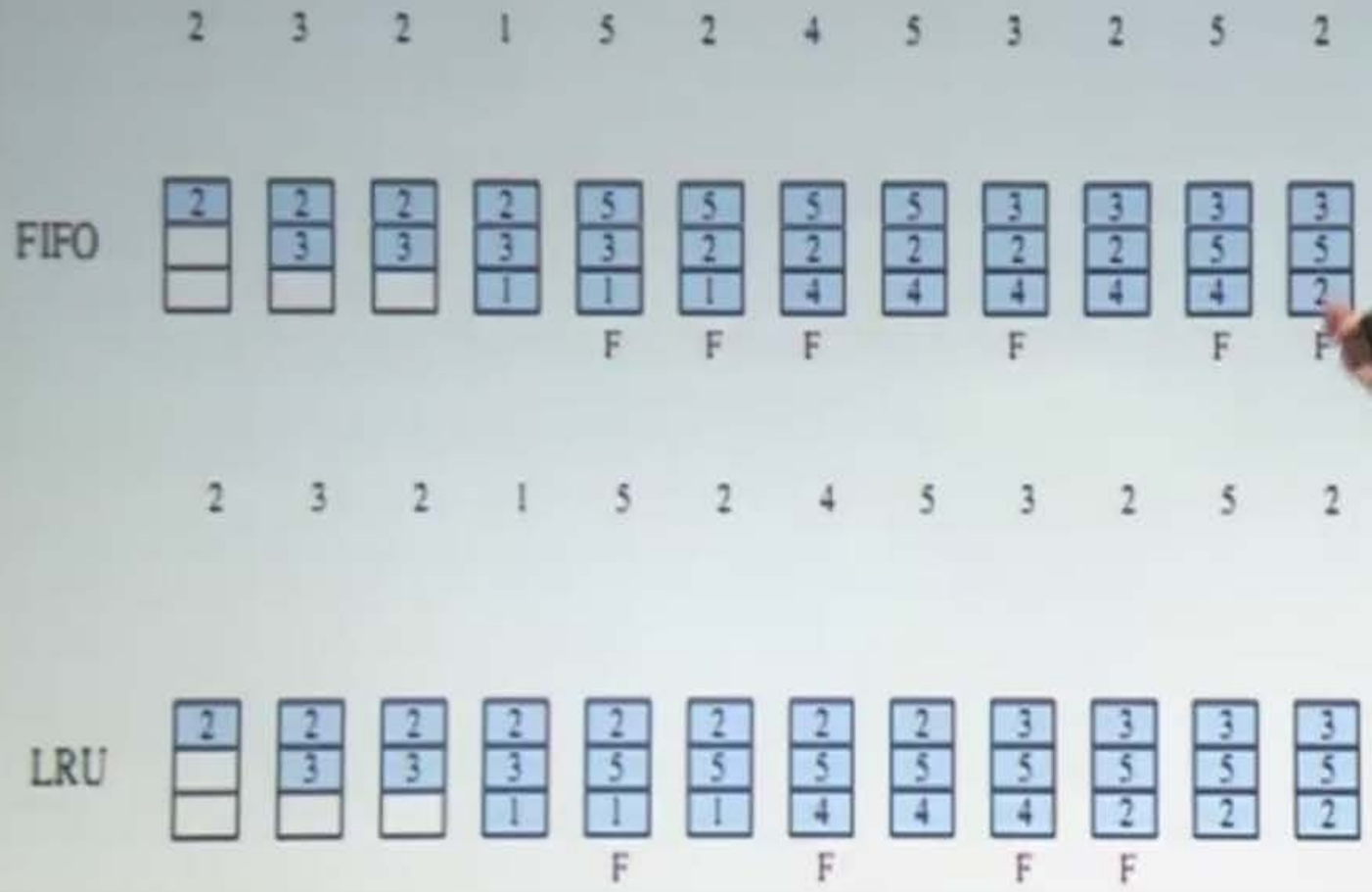
  2 3 2 1 5 2 4 5 3 2 5 2

要求：

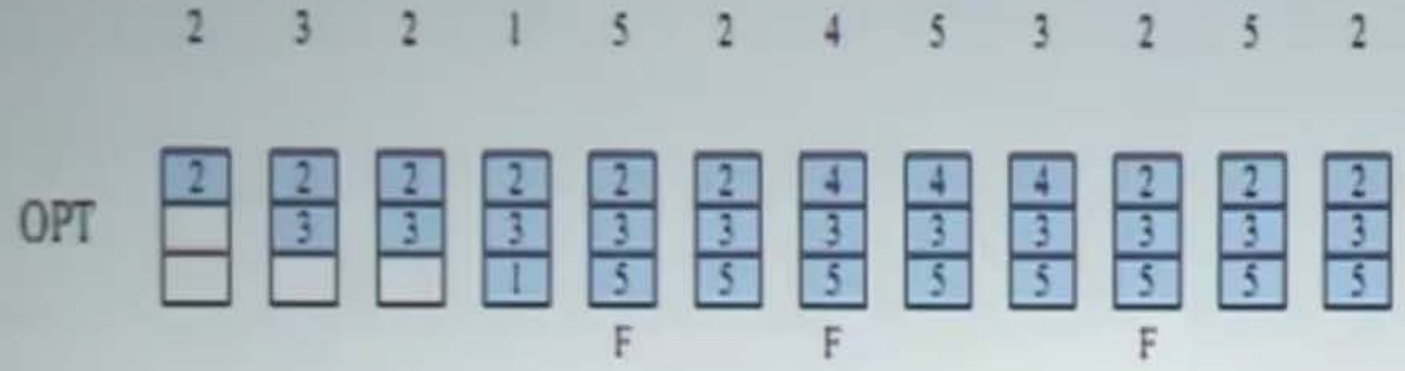
计算应用FIFO、LRU、OPT算法时的缺页次数

页面访问序列，它的缺页次数是多少？下面我们来看一下计算过程

应用FIFO、LRU页面置换算法



应用OPT页面置换算法



所以我们可以看到 LRU 是最接近最佳页面置换算法的。

BELADY现象

例子：系统给某进程分配 m个页框，初始为空

页面访问顺序为

1 2 3 4 1 2 5 1 2 3 4 5

采用FIFO算法，计算当 m=3 和 m=4 时的缺页中断次数

m=3时，缺页中断9次；m=4时，缺页中断10次

注：FIFO页面置换算法会产生异常现象（Belady现象），即：当分配给进程的物理页面数增加时，缺页次数反而增加

0:00 下面呢我们来介绍置换算法 置换算法呢又称之为页面淘汰算法或者是替换算法 我们主要有这样一些页面置换算法 针对每一个页面置换算法 我们要求大家能够掌握这些页面置换算法的设计思路 和它的一个算法的应用。 那么有一些 页面置换算法呢希望大家能够知道它是如何来实现的 第一个要介绍的是最佳页面置换算法 顾名思义，这个算法应该是最优的 它的设计思想是置换以后不再需要的页面 或者是以后 需要，但是是在最远的将来才会用到的页面 当然这是一个很好的算法，但是它能实现吗？ 我们怎么知道哪些页面以后会用到，什么时候用到呢？ 因此呢，这个算法的实现是要建立在已经知道页面的 走向序列的基础之上，我们才能够实施这个算法 当然，一个程序运行过去之后，如果 我们把它的访问的页面都记录下来，我们可以针对这个记录的结果 运用这样一个最佳的页面置换算法来给出哪些页面要换出内存 那么这个算法的 更大的意义是它是作为一种标准来衡量其他的 置换算法它的性能是好，还是不好 哪些页面置换算法最接近 最佳页面置换算法，那么这些算法就是好的算法 这就是它的主要的作用 那么下面我们介绍 先进先出的这样一个页面置换算法 基本思想就是选择在内存当中驻留时间最长的页面置换出去，也就是先进 内存的，那么先被置换出去 在实现的时候呢，我们可以用一个页面的链表 每加载到内存一个页面，就把它挂入到链表里头去 那么置换的时候呢，是从链表的头开始选择 选择链首的一个页面把它置换出去 那么这个算法呢如果我们对照一个日常生活中的 场景，就是说超市或者便利店里头如何撤换商品这样一个 现象，我们来做一个解释 那么有一个便利店，那么它卖各种各样的商品 那么出了一种新的商品，期望能够进入这个便利店 那么便利店呢可能会通过腾出，把原来的某一个 商品把它换出去，腾出地方来，来放新的商品 到底选择哪些商品把它淘汰了，然后换新的商品呢？ 如果我们按照先进先出的这个算法，那我们一定会把那些最早进入这个便利店来 销售的这个商品换掉，而我们大家都知道，那个最早 卖的这些东西都是大家日常中最需要买的东西 经常买的东西，柴米油盐酱醋茶，这些东西都是常用的 如果你把某一项淘汰掉了，因为它最早进入，那么实际上呢 对于这个便利店来讲，那么就不便利了，因为很多人要买的东西你没有 所以呢，先进先出这个页面置换算法其实有这样一种问题 特别是一些常用的页面都是很早就进入的 那么你把它淘汰掉，它还得再 通过缺页的异常处理，把它再调进来，实际上又增加了新的开销 这是先进先出页面置换算法。 那么我们就 在先进先出页面置换算法基础之上呢，我们又进行了一些改进，叫做第二次机会算法 所谓第二次机会算法呢，就是 首先我先按先进先出算法先选择一个页面，选择

一个页框 那么这个时候呢我去检查它的访问位，R 位，访问位是 R 位 前面我们有 A 位，这里头是 R 位，Reference，那么检查这个访问位 R 位，如果为 0 就说明它有一段时间没有被访问过了，那么就把它置换出去 这就是要置换的页面。 如果为 1，就给它第二次机会 这个时候呢，把这个访问位呢置成 0，所以访问位 R 位，那么它会 清零的，所以下一次再转过来呢，就可以把这个页面，如果它还是 0 就把它淘汰掉，淘汰掉，或者叫置换出去 那么这里头呢我们来看这样一个例子。 那么 A 呢 现在是到头了，如果检查 A 的这个访问位是 1 是 1 的话，那就把送到了后面 然后呢把它的那个访问位呢置成 0 就可以了，这就是第二次机会算法 下面我们介绍时钟算法 我们先来看第二次机会算法 在实现的时候的一个问题，当我选中了一个 页框以后，那么如果这个页框的访问位是 1 ，我就给它第二次机会 那么把这个页框从链首摘下来，把它的访问位置成 0，然后把它挂到链尾 那么摘链、挂链都需要花一些开销 那么怎么能加快这样一个速度呢？我们就通过的是时钟算法 时钟算法呢是把所有的页框组织成一个环形 然后有一个指针，你通过移动指针来选择下一个要淘汰的页框 好，那么假设现在指针指向这个页框，它的访问位是 1 那么这个根据规则，那么我们就要把它保留，那么把它的 访问位设置成 0，然后呢指针 下移，指针移完之后又访问到这个，又查找到这样一个页框 它的访问位也是 1 ，所以呢也把这个页框保留 那么再指向下一个页框，下一个页框大家会发现 访问位是 0，所以就把这个页框淘汰了，淘汰了以后，就置换出 置换进新的页面。 它置换进新的 页面置换呢，访问位是 1，然后这个指针就指向下一个 要置换的这样一个页框 这就是时钟算法，我们可以看到它是通过移动这个 指针来选择下一个要置换的页框的 这样就要比摘链、挂链要快。 那么下面我们来介绍 最近未使用的一个页面置换算法 那么这个算法它的思想是选择在 最近一段时间内没有使用过的一页把它置换出去 它的具体做法呢，是根据页表表项的 这个两位，一位呢是访问位 R 位，一位呢是修改位 M 位 就根据这两位来决定哪一个页面要被置换 那么我们知道硬件会给出这两位的设置，那么如果 硬件没有这些位的话呢，其实我们也可以用软件来模拟，来模拟这样子，做个标记就可以了 具体的当启动一个进程的时候，那么 R 位、 M 位都设置成 0 那么如果有修改，那么 M 肯定是 1 了，那么 R 位呢？因为每次访问，R 位都被 设置成 1，那这样 R 位就没有任何的信息量可参考，所以 R 位我们是对它定期清零的 定期清零。 这样的话呢，如果很长时间内它没有被使用 过，它实际上就

清完零，那以后就不是 1 了，有这种可能性了 那么这个算法呢就是当 产生缺页异常的时候呢，那么操作系统就会去检查 R 位和 M 位 就分成了这么几类页框。 第一类页框是 R 位是0，M 位是0，就是无访问，没有修改 第二类呢就是没有访问，有修改，因为我们的 R 位是定期清零，所以第二种这种是会出现的 第三类集合里头的页框呢，是有访问，没有修改 第四类是有访问，有修改，而最近未使用这样一个页面置换算法呢就是从这样 一个编号，这个不同的集合当中，从编号最小的这样一个 非空类的集合当中选择，任意选择一页把它置换出去 随机选择一个置换出去，好，这是最近未使用的这样一个页面置换算法 那我们也同样可以对这个算法进行一个时钟的实现，时钟算法的实现 就是从当前的，指针的当前位置开始扫描 整个的页框的缓冲区。 选择到了第一个遇到的页框 r = 0 ; m = 0，那么就用于置换 在扫描过程当中呢，对于使用我也不做任何修改 那么如果第一步失败了，第一步失败了那就重新扫描 那么选择 r = 0 ; m = 1 的这样的页框 那么如果，在扫描过程中如果你选中了它 那么，要跳过的页框的话，就把它的使用位设置成 0 如果第二步也失败了没有选中，那么指针就回到了最初的位置，那么 这时候集合当中的所有页框的使用位呢，就都是 0 了，就重复第一步，重复第一步 如果有必要的话，再重复第二步。 当然我们就是想说，诶，这个算法可以用时钟算法来实现，啊实现 最近最少使用页面置换算法，这是一个 最常用的，就是很多的系统中常用的这么一个算法 那么最近最少使用算法的基本思想是 选择最后一次访问时间，距离当前时间最长的一页置换掉。 也就是 它的访问时间距离当前时间间隔最大的，我们把它置换掉 那么也就是置换出未使用时间最长的一页 为什么这个算法用得比较多呢？是因为这个算法的性能是最接近 我们的最佳页面置换算法的。 但是在实现这个算法的时候呢 会遇到一个非常大的开销，因为我们要给每一个页面页框设置一个时间戳 记录下来它使用的时间。 那然后 我们就会，当页面置换的时候我们就在所有的页框当中去 比较，看看哪个时间最久，哪个时间最久 最小，那么当然，这个开销就非常大，系统中有太多的页面了 当然也可以维护一个，页面访问的一个所谓的页面的一个 栈，栈，也都可以，但是呢总体来讲，开销比较大 那么这里头，也就是我们的教材当中呢，举了一个例子 就是说用硬件的办法来实现 LRU ，当然啦，这首先需要硬件支持 那么，这里头我们来看一下。 它的思想是说，给每一个页框呢，设置了这么 一个矩阵，当然这是硬件，设置了矩阵。 当这个页框被访问的时候 就把这个页框所对应

的这个，这一行设置为 1 然后，把这个页框所对应的这一列设置为 0 那么，如果我们来看一下啊。 页面访问的顺序是 0， 1， 2，3，3，2，1 0， 3，2，3 的话，那么访问 0 的话，我们就 把 0 这一，这一，对应的这一行设置为 1 把这一列设置为 0，如果访问的是 这个页框，就把这个页框这一行设置为 1，把对应的这一列设置为 0，那么经过了 每次访问都有这么一个这个设置，当然这是硬件 做，要不然就太慢啦。 到最后，比如说到了这样一个情况，这个时候要 置换某页框了，那么怎么做呢？就在这个矩阵当中找到 这个值最小的那一个行走，值最小的那一个把它置换出去 这就是 LRU 算法的一种硬件的实现。 当然啦我们都看到，开销是非常大的，这是 4 个页框，4 × 4 的这么一个，要如果是 100 个页框呢， 100 × 100 的这么一个硬件的这种矩阵，那个开销非常大 那么，下面我们来看看有没有别的办法来实现 LRU，就是近似的来实现 LRU 那么，这个过程当中呢，首先先出现了一个叫，最不经常使用的页面置换算法 它的思想是说，选择访问次数最少的页面置换掉 原来是说，时间间隔最远的，我把它置换掉 它就把它转换成了频率了，访问次数最少的。 当然啦，这个思想是不吻合的 那么，最初提出这个算法是意图，它说它是一个 LRU 的一个软件解决方案 当然这个，我们很明显看到它这个和 LRU 的思想相差还是甚远的 但是它做法是这样的，软件计数器到，给每个页框有个软件计数器，就不能用硬件了 那么，一个页框一个，那么初值是 0。 然后每次时钟中断的时候呢，就给这个 计数器加一个值，这个值呢就是 R，R 如果是 0 呢，就是加 0，R 是 1 呢，就加 1 那么一旦发生了缺页中断，那么就选择这个计数器值最小的那个页面，把它置换出去啊 这是最不经常使用这样一个算法 那么这个算法虽然和 LRU，相比是相差的还比较远，但是呢，给我们后续的算法提供了一个基础 而我们后续的算法呢，就称之为老化算法 老化算法实际上是一种，通过对刚才的这个算法的改进 让它来模拟 LRU，更进一步地模拟 LRU 那么，基本思想是什么呢？就是这个计数器还是计数器，只是计数器 在加 R 之前，首先先要右移一位，先右移一位那么我们知道右移一位就是相当于衰减，衰减，就除 2 了，啊除 2 了 好，那么，我们的 R 呢，也不是加在这个计数器的最 右端，最低位部分。 而是把这个 R 呢，加到了计数器的 最左端高位部分。 那我们知道，如果 R 是 1，那么 这个值就变得很大，如果 R 是 0，就相当于没有任何的影响 那么，这种算法的基本思想是什么呢？ 就是如果一个页面被访问过 而被访问过的页面呢，那么它就是，按照我这算法它就

是要在 这个计数器的最左端啊，高位部分加 1 就 R 嘛，被访问过，我们是 1 ，然后加 1 但是如果很久很久了，这个页面，就是很久很久就没有被访问过了 那么这个 R ，这个原来这个 1 呢，实际上就慢慢慢慢的作用就越来越小 这样就衰减掉了。 因为它每次右移每次右移，那么这个 1 的位置就会越来越往右 那么，我们知道，如果很久没被访问过了，前面都填充的是 0 ，那么这个数值就会越来越小 实际上就是利用了这样一个思想，来近似地来模拟 LRU 我们来看一个例子，那么这个例子的话呢，比如说在 某一时刻，这几个页面的访问，是它被访问过了 它没被访问，访问过，没被访问，访问，访问。 因此我们来看到这个计数器 最高位部分是 1 ， 0 ， 1 ，0 ， 1 ， 1 ，对吧？就是符合这样一个访问的 那么过了一个时间，过了一个时间。 比如说第二个时间 tick 2 ， tick 1 ，过一个时间，那么 这些页面的这个 R 位呢，变成了 1 ， 1 ， 0 ， 0 ， 1 ， 0 ，那么这个时候呢，我们 就要首先先把所有的计数器都是右移，然后再把这个 R 位加上去 一个时间一个时间过去，那么到了这个时刻，那么我们得到了一个结果 而这个时候如果要置换的话，我们就可以从这些计数器 当中选择这个值最小的一个，可以把它置换出去 那么，如果有相同的呢，我们就随机选一个就可以了。 这就是所谓 老化算法。 通过了这样一个改变 对计数器的这样一个改变，那么我们实际上发现，就是如果 R 位是访问过的，那么它的这个访问值加在最左端 使这个数值很大。 如果它很早以前访问过了，经过了一段时间就衰减掉了 所以，它的作用就起不到了。 这就近似地模拟了我们的 LRU，淘汰 淘汰间隔时间最远的那个页框 当然大家可以思考一下，它所以，它与 LRU 有什么样的区别？ 那么下面我们给出页面置换算法的应用 例子。 我们假设啊，系统给进程分配了 3 个页框 那么，我们假设这是固定的分配策略，为了计算方便，我们给出的是固定分配策略 那么初始的时候，这页框都是空的，还没有页面进入内存。 那么 进程执行时候的页面访问的顺序呢，是这样一个顺序，比如说：2 3 2 1 5 2 4 啊 ， 5 3 2 5 2 啊，有这么一个页面访问次序。 那么下面我们要求大家能够计算 应用先进先出算法 LRU 算法和最佳页面置换算法的时候，针对这样一个 页面访问序列，它的缺页次数是多少？下面我们来看一下计算过程 上面这张图呢，是运用了 先进先出的页面置换算法。 我们来看一下 在这里，初始的时候三个页框都是空的，那么，要访问 2 号页面 那么就进来，啊，这是缺页一次，啊，把二号页面读进内存。 然后，要访问三号页面，好，缺页一次，把三号页面读进内存。 然后又要访

问二号页面，我们知道二号页面已经在内存了，所以这时候没有缺页，没有变- 化内存。然后要访问一号页面，那么一号页面呢，啊，进入内存，这个时候内存已经满了，所以这个时候 已经有三次缺页了，三次缺页了，从现在开始，这个时候我们来看，要 访问五号页面，那么五号页面我们知道 内存已经满了，所以我要淘汰，啊，一个页框，淘汰哪个页框呢？我们来看谁 先进来的，因为先进来的是二号页面，啊，就是二号页面所在的页框，所以我就把这个页框， 把二号页面淘汰掉，然后把五号页面送到这个位置。 然后就是二号页面刚刚被置换出去，所以又要进内存，所以 又要进内存呢，所以就把三号页面，啊，置换出去，因为三号页面是其次进来的。 那么就变成了这样一个情况，这也缺页，然后访问四号页面，四号页面要进来，就把一号页面- 置换出去， 然后又要访问五号页面，这个时候没有缺页。 再访问三号页面，又缺页一次，然后 经一次计算，我们可以看到一共缺页呢，是 这有六次，前面有三次，所以一共是九次缺页。 下面我们来看一下 LRU 的算法的这个应用。 前面我们都一样，啊，到了这开始啊， 当三个页面已经把内存填满了以后，这时候要访问 五号页面，那么五号页面呢，要置换出一个页面来，啊，那么置换哪一个呢？ 就要看它没被访问的这个时间， 就是间隔最久的，那么二号页面虽然先进来，但是二号页面又被访问过了，所以我们可以 看到是三号页面，啊，未被访问的时间是最久的，所以把三号页面置换掉，所以五呢， 把三号页面置换掉，然后又要访问二号页面，这就 没有出现缺页了，所以就跟 FIFO 比起来，它又好了。 然后这时候要访问四号页面，四号页面要把谁置换掉呢？我们来看一下 那么这里头有二号、 五号和一号，而一号呢，是最早被访问的，所以要把一号页面置换掉， 缺页一次，置换出一号页面，然后五号页面， 再访问，这时候我们发现，又命中了，没有，不需要那个 啊，出现缺页，再访问三号页面的时候呢，就把二号页面置换掉，等等等等。 大家自己回去慢慢看一下，那么我们这里头会发现，它的缺页次数呢就是 四次再加三次，一共是七次，所以刚才是九次，这是七次，说明 LRU 还比 FIFO 要好了。 我们来看一下，最佳页面置换算法，缺页几次呢？ 大家会看到，在五号页面要进来， 淘汰谁呢，我们来看一下，一号页面以后不再需要了。 所以，最好的就是把一号页面淘汰掉，所以我们知道最佳 页面置换算法肯定效果是最理想的，所以把一号页面置换掉。 而这时候缺页呢，是在 2、 3、 5 里头选， 2、 3、 5 里头我们可以看到，二是最远的将来才访问到的，所以把二号页面置换

掉。 啊。 在这里缺页的话呢，是在 4、 3、 5 里头选，那么我们在这里头 可以说把二号也可以去掉，然后这个 在这里头我们可以看一下，那么如果这次缺页，二号要进来，在 4、 3、 5 里头选， 那么四号和三号其实都符合要求，我们就随机选一个就可以了。 所以这是最佳页面置换算法，所以总共、 一共是六次缺页。 所以我们可以看到 LRU 是最接近最佳页面置换算法的。 下面我们来介绍一个现象，叫 BELADY 现象，那么这个现象呢，我们给一个场景： 就是系统呢，会给进程分配了 m 个页框， 然后，初始的时候，这个 m 页框都是空，那我们给出了 一个页面访问的顺序是，1 2 3 4 1 2 5 1 2 3 4 5 要求采用 FIFO 这种页面置换算法。 计算出来当 m 等于 3 的时候、 m 等于 4 的时候，它的缺页的次数。 那么经过计算呢， 其实我们发现了这样一个结果，m 等于 3 的时候，缺页次数是 9 次； m 等于 4 的时候，是缺页中断是 10 次。 那么我们 前面看的那张图，分配给你的页框越多， 那么缺页次数就应该越少，而这儿却出现了一个 异常的现象，那么这就是一个反常的事情发生了，我们 这种反常呢，是由 BELADY 发现的，所以它这个现象呢，我们称之为 BELADY 现象。 那么 FIFO 算法，也就是先进先出的这个页面 置换算法呢，它就会产生这个现象，那么这个现象的思想是： 分配给这个进程的物理页面数增加，它的缺页数反而增加。