置换算法的功能和目标

- 功能
  - 当出现缺页异常，需调入新页面而内存已满时，置换算法选择被置换的物理页面，放到外存中.
- 设计目标
  - 尽可能减少页面的调入调出次数
  - 把未来不再访问或短期内不访问的页面调出(淘汰)
- 页面锁定(frame locking)：有些页面不可以放外存.
  - 描述必须常驻内存的逻辑页面
  - 操作系统的关键部分
  - 要求响应速度的代码和数据.外存太慢了.
  - 页表中的锁定标志位(lock bit)

置换算法的评价方法

- 记录进程访问内存的页面轨迹
  - 举例: 虚拟地址访问用(页号, 位移)表示
    (3,0), (1,9), (4,1), (2,1), (5,3), (2,0), (1,9), (2,4), (3,1), (4,8) 只关心页号
  - 对应的页面轨迹
    3, 1, 4, 2, 5, 2, 1, 2, 3, 4
    替换如 c, a, d, b, e, b, a, b, c, d
- 评价方法
  - 模拟页面置换行为，记录产生缺页的次数
  - 更少的缺页，更好的性能

页面置换算法分类

- 局部页面置换算法
  - 置换页面的选择范围仅限于当前进程占用的物理页面内
  - 最优算法、先进先出算法、最近最久未使用算法
  - 时钟算法、最不常用算法
- 全局页面置换算法
  - 置换页面的选择范围是所有可换出的物理页面
  - 工作集算法、缺页率算法

最优页面置换算法(OPT, optimal)

- 基本思路
  - 置换在未来最长时间不访问的页面.
- 算法实现
  - 缺页时,计算内存中每个逻辑页面的下一次访问时间
  - 选择未来最长时间不访问的页面
- 算法特征
  - 缺页最少,是理想情况
  - 实际系统中无法实现,因为无法预知每个页面在下次访问前的等待时间.
  - 作为置换算法的性能评价依据.

最优页面置换算法示例

缺页

缺页.

时间		0	1	2	3	4	5	6	7	8	9	10
访问请求			c	a	d	b	e	b	a	b	c	d
内存块号	0	a	a	a	a	a	a	a	a	a	a	
	1	b	b	b	b	b	b	b	b	b	b	
	2	c	c	c	c	c	c	c	c	c	c	
	3	d	d	d	d	d	e	e	e	e	e	
缺页状态							●					●
每页的下次访问时间							a=7 b=6 c=9 d=10	⇒未来访问时间.				a=? b=? c=? d=?

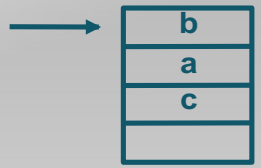
先进先出算法(First-In First-Out, FIFO)

- 思路
  - 选择在内存驻留时间最长的页面进行置换
- 实现
  - 维护一个记录所有位于内存中的逻辑页面链表
  - 链表元素按驻留内存的时间排序，链首最长，链尾最短
  - 出现缺页时，选择链首页面进行置换，新页面加到链尾
- 特征
  - 实现简单
  - 性能较差，调出的页面可能是经常访问的
  - 进程分配物理页面数增加时，缺页并不一定减少(Belady现象)
  - 很少单独使用

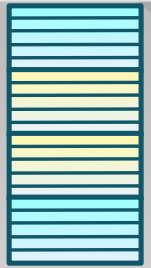
FIFO：完全不考虑过去的访问情况

在4个页帧中执行：

- 假定初始a->b->c->d顺序



访问页面链表



进程占用物理内存

a→b→c→d→e

b→c→d→e→a

c→d→a→a→b

a→e→a→b→c

e→a→b→c→d

时间		0	1	2	3	4	5	6	7	8	9	10
访问请求			c	a	d	b	e	b	a	b	c	d
驻留集	0	a	a	a	a	a	e	e	e	e	e	d
	1	b	b	b	b	b	b	b	a	a	a	a
	2	c	c	c	c	c	e	c	c	b	b	b
	3	d	d	d	d	d	d	d	d	d	c	c
缺页状态							●		●	●	●	●

最近最久未使用算法 (Least Recently Used, LRU)

- 思路
  - 选择最长时间没有被引用的页面进行置换
  - 如某些页面长时间未被访问，则它们在将来还可能会长时间不会访问
- 实现
  - 缺页时，计算内存中每个逻辑页面的上一次访问时间
  - 选择上一次使用到当前时间最长的页面
- 特征
  - 最优置换算法的一种近似

精细地考虑每一次访问情况

最近最未被使用算法(LRU)

置换的页面是最长时间没有被引用的

时间		0	1	2	3	4	5	6	7	8	9	10
访问请求			c	a	d	b	e	b	a	b	c	d
内存块	0	a	a	a	a	a	a	a	a	a	a	a
	1	b	b	b	b	b	b	b	b	b	b	b
	2	c	c	c	c	c	e	e	e	e	e	d
	3	d	d	d	d	d	d	d	d	d	c	c
缺页状态							●				●	●
每页的下次访问时间							a=2 b=4 c=1 d=3				a=7 b=8 e=5 d=3	a=7 b=8 e=5 c=9

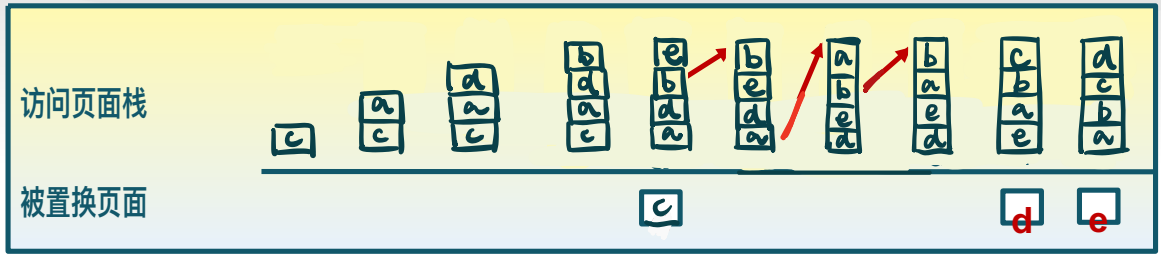
LRU算法的可能实现方法

- 页面链表
  - 系统维护一个按最近一次访问时间排序的页面链表
    - 链表首节点是最近刚刚使用过的页面
    - 链表尾节点是最久未使用的页面
  - 访问内存时，找到相应页面，并把它移到链表之首
  - 缺页时，置换链表尾节点的页面
- 活动页面栈
  - 访问页面时，将此页号压入栈顶，并栈内相同的页号抽出
  - 缺页时，置换栈底的页面
- 特征
  - 开销比较大

用栈实现LRU算法

保持一个最近使用页面的“栈”

时间		0	1	2	3	4	5	6	7	8	9	10
访问请求			c	a	d	b	e	b	a	b	c	d
栈内容	0	a	a	a	a	a	a	a	a	a	a	a
	1	b	b	b	b	b	b	b	b	b	b	b
	2	c	c	c	c	c	e	e	e	e	e	b
	3	d	d	d	d	d	d	d	d	d	c	c
缺页状态							●				●	●



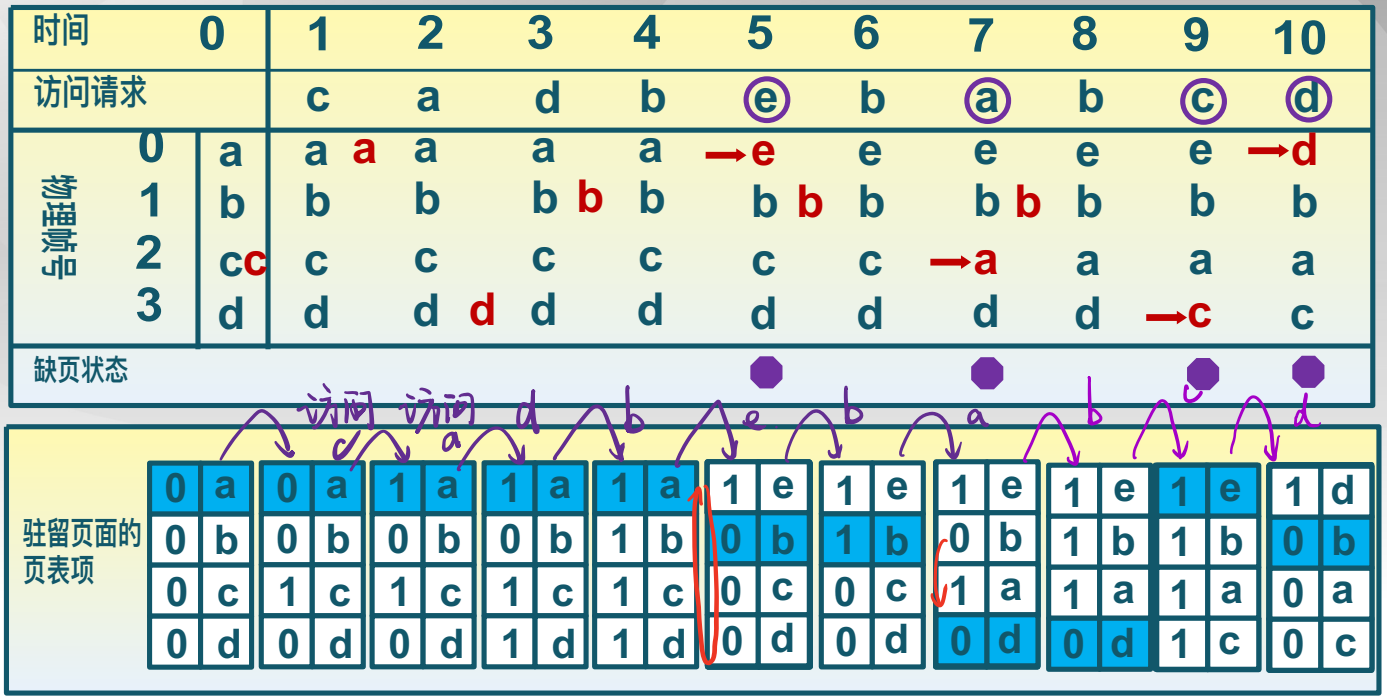
时钟置换算法（Clock）

- 思路
  - 仅对页面的访问情况进行大致统计 → LRU 统计太细.
- 数据结构
  - 在页表项中增加访问位，描述页面在过去一段时间的内访问情况
  - 各页面组织成环形链表
  - 指针指向最先调入的页面
- 算法
  - 访问页面时，在页表项记录页面访问情况
  - 缺页时，从指针处开始顺序查找未被访问的页面进行置换
- 特征
  - 时钟算法是LRU和FIFO的折中

时钟置换算法的实现

- 页面装入内存时，访问位初始化为0
- 访问页面（读/写)时，访问位置1
- 缺页时，从指针当前位置顺序检查环形链表
  - 访问位为0，则置换该页
  - 访问位为1，则访问位置0
    ，并指针移动到下一个页面，直到找到可置换的页面

时钟页面置换示例


时间 0 1 2 3 4 5 6 7 8 9 10
访问请求 c a d b e b a b c d
驻留序号 0 1 2 3
a a a a a e e e e e d
b b b b b b b b b b b
c c c c c c a a a a a
d d d d d d d d c c
缺页状态
访问 访问 d b e b a b c d
驻留页面的页表项
0 a 0 a 1 a 1 a 1 a 1 e 1 e 1 e 1 e 1 e 1 d
0 b 0 b 0 b 0 b 1 b 0 b 1 b 0 b 1 b 1 b 0 b
0 c 1 c 1 c 1 c 1 c 0 c 0 c 1 a 1 a 1 a 0 a
0 d 0 d 0 d 1 d 1 d 0 d 0 d 0 d 0 d 1 c 0 c

最不常用算法 (Least Frequently Used, LFU)

- 思路：缺页时，置换访问次数最少的页面。
- 实现
  - 每个页面设置一个访问计数
  - 访问页面时，访问计数+1
  - 缺页时，置换计数最小的页面
- 特征
  - 开销大
  - 开始时频繁使用，但后期不使用的页面很难置换
    - ⇒ 解决方法：计数器定期右移（除以2）
- LRU 和 LFU 区别
  - LRU 关注多久未访问，时间越短越好。
  - LFU 关注访问次数，次数越多越好。

LFU算法示例

执行在4个页帧中：

- 假定最初的访问次数a->8 b->5 c->6 d->2

访问c7次

时间		0	1	2	3	4	5	6	7	8	9	10
访问请求			c^{7}	a^{1}	d^{14}	b^{5}	e^{18}	b^{1}	a^{19}	b^{20}	c^{20}	d^{17}
页帧号	0	a^{8}	a^{8}	a^{9}	a^{9}	a^{9}	e^{18}	e^{18}	e^{18}	e^{18}	e^{18}	d^{17}
	1	b^{5}	b^{5}	b^{5}	b^{5}	b^{10}	b^{10}	b^{11}	a^{19}	a^{19}	a^{19}	a^{19}
	2	c^{6}	c^{13}	c^{13}	c^{13}	c^{13}	c^{13}	c^{13}	c^{13}	b^{20}	b^{20}	b^{20}
	3	d^{2}	d^{2}	d^{2}	d^{16}	d^{16}	d^{16}	d^{16}	d^{16}	d^{16}	c^{20}	c^{20}
缺页状态							●		●	●	●	●

每次都置换掉计数最小的那个

Belady现象

- 现象
  - 采用FIFO
    等算法时，可能出现分配的物理页面数增加，缺页次数反而升高的异常现象
- 原因
  - FIFO算法的置换特征与进程访问内存的动态特征矛盾
  - 被它置换出去的页面并不一定是进程近期不会访问的
- 思考
  - 哪些置换算法没有Belady现象?

FIFO算法有Belady现象

访问顺序：1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

物理页面数: 3　　缺页次数: 9

FIFO	1	2	3	4	1	2	5	1	2	3	4	5
尾	1	2	3	4	1	2	5	5	5	3	4	4
		1	2	3	4	1	2	2	2	5	3	3
头			1	2	3	4	1	1	1	2	5	5
缺页状态	●	●	●	●	●	●	●			●	●	

置换掉1.

FIFO算法有Belady现象

访问顺序：1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

物理页面数: 4　　缺页次数: 10

正好每次都是刚拿出去的那个.

FIFO	1	2	3	4	1	2	5	1	2	3	4	5
尾	1	2	3	4	4	4	5	1	2	3	4	5
头		1	2	3	3	3	4	5	1	2	3	4
头			1	2	2	2	3	4	5	1	2	3
头				1	1	1	2	3	4	5	1	2
缺页状态	●	●	●	●			●	●	●	●	●	●

LRU算法没有Belady 现象

物理页面数: 3　　缺页次数: 10

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

栈的方法.

物理页面数: 4　　缺页次数: 8

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

时钟/改进的时钟页面置换是否有Belady现象?
为什么LRU页面置换算法没有Belady现象?

LRU、FIFO和Clock的比较

- LRU算法和FIFO本质上都是先进先出的思路
  - LRU依据页面的最近访问时间排序
  - LRU需要动态地调整顺序
  - FIFO依据页面进入内存的时间排序
  - FIFO的页面进入时间是固定不变的
- LRU可退化成FIFO
  - 如页面进入内存后没有被访问，最近访问时间与进入内存的时间相同

    不用改变栈顺序.
  - 例如：给进程分配3个物理页面，逻辑页面的访问顺序为1、2、3、4、5、6、1、2、3…

LRU、FIFO和Clock的比较

- LRU算法性能较好，但系统开销较大，会全部改变栈顺序
- FIFO算法系统开销较小，会发生Belady现象
- Clock算法是它们的折衷
  - 页面访问时，不动态调整页面在链表中的顺序，仅做标记
  - 缺页时，再把它移动到链表末尾
- 对于未被访问的页面，Clock和LRU算法的表现一样好（clock=LRU=FIFO）
- 对于被访问过的页面，Clock算法不能记录准确访问顺序，而LRU算法可以

LRU会全部改变.

访问过的直接跳过去.

只记录了访问与否.