哈工大操作系统-L25内存换出

- 选择哪一页换出去？是个问题...
- get free page 找不到空闲页了，就需要换出了

1.换出算法概述

get_free_page? 还是...

```
page=get_free_page();
bread_page(page, current->executable->i_dev, nr);
```

- 并不能总是获得新的页，内存是有限的
  - 需要选择一页淘汰，换出到磁盘，选择哪一页?
  - FIFO，最容易想到，但如果刚换入的页马上又要换出怎么办?
  - 有没有最优的淘汰方法? MIN
  - 最优淘汰方法能不能实现，是否需要近似? LRU

- FIFO
- MIN
- LRU

2.FIFO，淘汰最早被换入的页面=

FIFO页面置换

- 一实例: 分配了3个页框(frame)，页面引用序列为

  A B C A B D A D B C B

  引用序列即需要访问哪些页

D换A不太合适!

Ref: Page:	A	B	C	A	B	D	A	D	B	C	B
1	A					D				C	
2		B					A				
3			C						B		

- 评价准则: 缺页次数；本实例，FIFO导致7次缺页

  问题：换谁最合适?

缺点：缺页次数多

3.MIN，淘汰最远将使用的页

MIN页面置换

- MIN算法: 选最远将使用的页淘汰，是最优方案
  - 继续上面的实例: (3frame)A B C A B D A D B C B

Ref: Page:	A	B	C	A	B	D	A	D	B	C	B
1	A									C	
2		B									
3			C			D					

  - 本实例，MIN导致5次缺页
  - 可惜，MIN需要知道将来发生的事... 怎么办?

- MIN是最优的方案
- 但是需要预知未来的页面引用序列

4.LRU，淘汰最近最长一段时间没有使用的页

LRU页面置换

- 用过去的历史预测将来。LRU算法: 选最近最长一段时间没有使用的页淘汰(最近最少使用)。
  - 继续上面的实例: (3frame)A B C A B D A D B C B

Ref: / Page:	A	B	C	A	B	D	A	D	B	C	B
1	A									C	
2		B									
3			C			D					

  - 本实例，LRU也导致5次缺页
  - LRU是公认的很好的页置换算法，怎么实现?

和MIN完全一样!

- 利用了程序的局部性的特点

准确实现--时间戳实现

LRU的准确实现，用时间戳

- 每页维护一个时间戳(time stamp)
  - 继续上面的实例: (3frame)A B C A B D A D B C B

	A	B	C	A	B	D	A	D	B	C	B
A	1	1	1	4	4	4	7	7	7	7	7
B	0	2	2	2	5	5	5	5	9	9	11
C	0	0	3	3	3	3	3	3	3	10	10
D	0	0	0	0	0	6	6	8	8	8	8

time stamp

选具有最小时间戳的页!

选A淘汰!

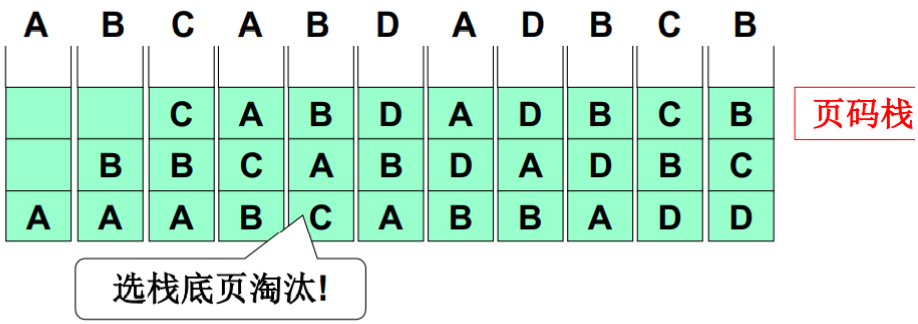
  - 每次地址访问都需要修改时间戳，需维护一个全局时钟，需找到最小值 … 实现代价较大

每次需要换出时，选择时间戳最小的(非0)页换出

准确实现--页码栈实现

LRU准确实现，用页码栈

- 维护一个页码栈
  - 继续上面的实例: (3frame)A B C A B D A D B C B



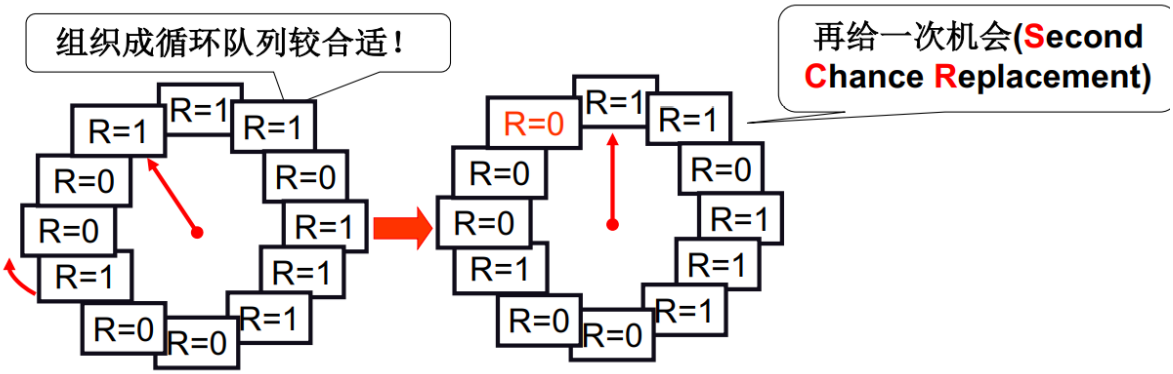
  - 每次地址访问都需要修改栈(修改10次左右栈指针) … 实现代价仍然较大 ⇒ LRU准确实现用的少

近似实现--CLOCK算法/SCR二次机会算法，使用循环队列

LRU近似实现 – 将时间计数变为是和否

- 每个页加一个引用位(reference bit)
  - 每次访问一页时，硬件自动设置该位
  - 选择淘汰页：扫描该位，是1时清0，并继续扫描；是0时淘汰该页

SCR这一实现方法称为 Clock Algorithm

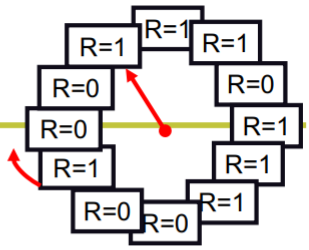


- 最近访问过，则有一次不被淘汰的机会
- 最近没被使用，则可以被替换

改动简单，只需要修改一个数，可以用MMU修改，把这个数保存在表项目中。

近似实现--双指针CLOCK算法

Clock算法的分析与改造



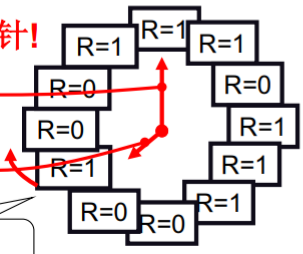
- 如果缺页很少，会？ 所有的R=1 因为只有缺页时，指针扫描时才会把1变为0
  - hand scan一圈后淘汰当前页，将调入页插入hand位置，hand前移一位
  - 原因: 记录了太长的历史信息… 怎么办?
  - 定时清除R位… 再来一个扫描指针!

退化为FIFO!

用来清除R位，移动速度要快!

用来选择淘汰页，移动速度慢!

更像Clock吧!



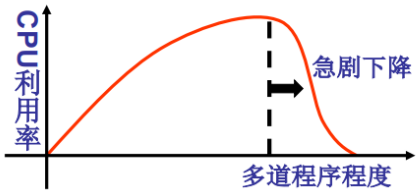
5. 每个进程分配多少页框?

置换策略有了，还需要解决一个问题

- 给进程分配多少页框(帧frame)
  - 分配的多，请求调页导致的内存高效利用就没用了!
  - 那分配的太少呢?

解释: 系统内进程增多 ⇒ 每个进程的缺页率增大 ⇒ 缺页率增大到一定程度，进程总等待调页完成 ⇒ CPU利用率降低 ⇒ 进程进一步增多，缺页率更大 …

看下面的现象: 横轴是进程个数



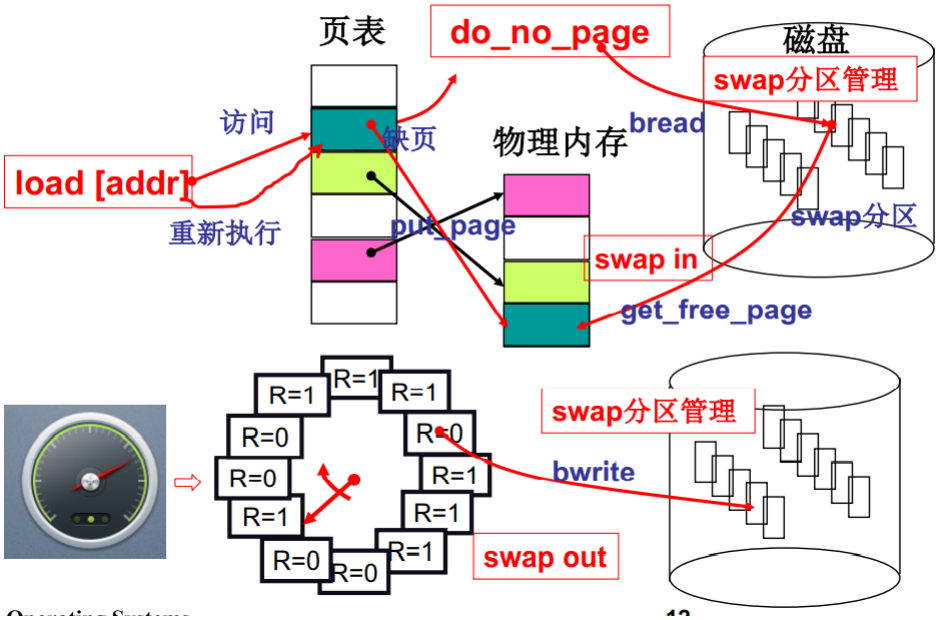
- 称这一现象为颠簸(thrashing)

帧，分配给进程的页框的个数

- 分配太多，内存无法高效率利用，并发的进程数也少
- 分配太少，进程个数多，缺页率增大，每许多进程都在等待调页，发生颠簸
- 有一个叫工作集的算法，计算分配多少个页框

6.总结

swap in　swap out　swap分区管理



进程带动的内存管理：

进程的执行需要把程序载入内存-->载入内存需要高效的管理内存-->高效管理内存需要段、页结合-->段页结合机制需要虚拟内存为桥梁-->虚拟内存的实现需要换入换出为支撑