操作系统原理

黄俊杰

2024年9月

第6章 虚拟存储器

Linux 内存管理视图

cs:ip

逻辑地址

4G

操作系统段

用户代码段

用户数据段

用户栈段

0

地址空间

物理地址

0x7008

0x00345008

线性地址

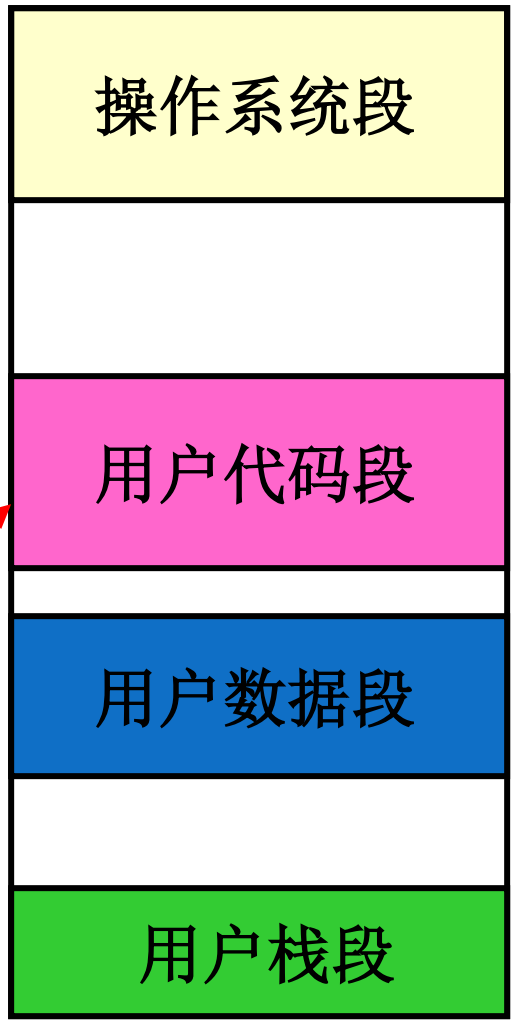
内存

对用户是透明的

用户眼里的内存!

cs:ip

逻辑地址



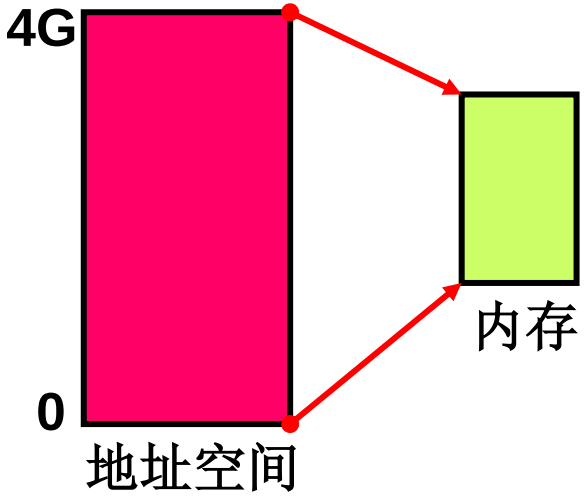
- 1个4GB(很大)的地址空间
- 用户可随意使用该地址空间，就象单独拥有4G内存
- 这个地址空间怎么映射到物理内存，用户全然不知

必须映射，否则不能用!

- 该地址空间就被称为“虚拟存储器”

虚拟存储器的优点

- 优点1: 地址空间>物理内存
  - 用户可以编写比内存大的程序
  - 4G空间可以使用，简化编程



- 优点2: 部分程序放入物理内存
  - 内存中可以放更多进程，并发度好，效率高
  - 将需要的部分放入内存，有些用不到的部分从来不放入内存，内存利用率高
  - 程序开始执行、响应时间等更快

如一些处理异常的代码!

虚拟内存思想既有利于系统，又有利于用户

虚拟存储器的引入

- 第5章介绍的存储管理方案要求作业**全部装入**内存才可运行。但这会出现两种情况：
  - 有的作业因太大，内存装不下而无法运行。
  - 系统中作业数太多，因系统容量有限只能让少数作业先运行。

加大物理内存

局部性原理 (Locality of Reference)

- 局部性原理（理论基础）1968年P.Denning 提出程序执行时，大多数情况下是顺序执行的。
  - 过程调用会使程序的执行轨迹从一部分内存区域转至另一部分区域，但过程调用的深度不会超过5。
  - 程序中有许多循环语句，这些语句会重复多次执行。
  - 程序中对数据结构的操作往往局限在很小的范围内。

局部性的表现

- 时间局部性
  - 程序中的的某条指令一旦执行，不久后会再次执行。
- 空间局部性
  - 程序一旦访问某存储单元，不久后会访问其附近的存储单元。

虚拟存储器的定义

- 所谓虚拟存储器是指具有请求调入功能和置换功能，能从逻辑上对内存容量进行扩充的一种存储器系统。
  - 离散性

    作业不装入连续的存储空间，内存分配采用离散分配方式
  - 多次性

    一个作业被分割，被多次调入内存。
  - 对换性

    作业在运行过程中换进、换出内存。
  - 虚拟性

    从逻辑上扩充了内存的容量。

虚拟存储器的功能

- 请求调入功能：
  - 将作业的部分页或段调入内存，其余的留在磁盘上，运行时，若所访问的页（段）不在内存（缺页或缺段）需要启用请求调入功能，将相关页（段）调入内存。
- 置换功能：
  - 若调入页（段）时，内存已满，即无法装入新页（段）需要启用页（段）置换功能将内存中暂时不用的页（段）调出至磁盘上腾出内存空间

如何实现虚拟内存!

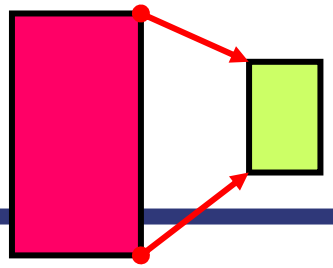
装入部分内存页面

- 你需要明确知道以下两个知识点：
  - 在程序运行期间，并不是所有的程序页面内容都会被访问到（比如处理异常的程序代码），因此，并不是所有的程序页面需要装到相应的物理内存页框。
  - 部分必要的程序页（或段）装入在物理内存后，程序是可以运行的；如果当所需要的指令或者数据所在的页（或段）没有装载入内存时，进程会暂停执行；但是，当相应的页面（或段）被载入内存后，进程又可以继续执行。

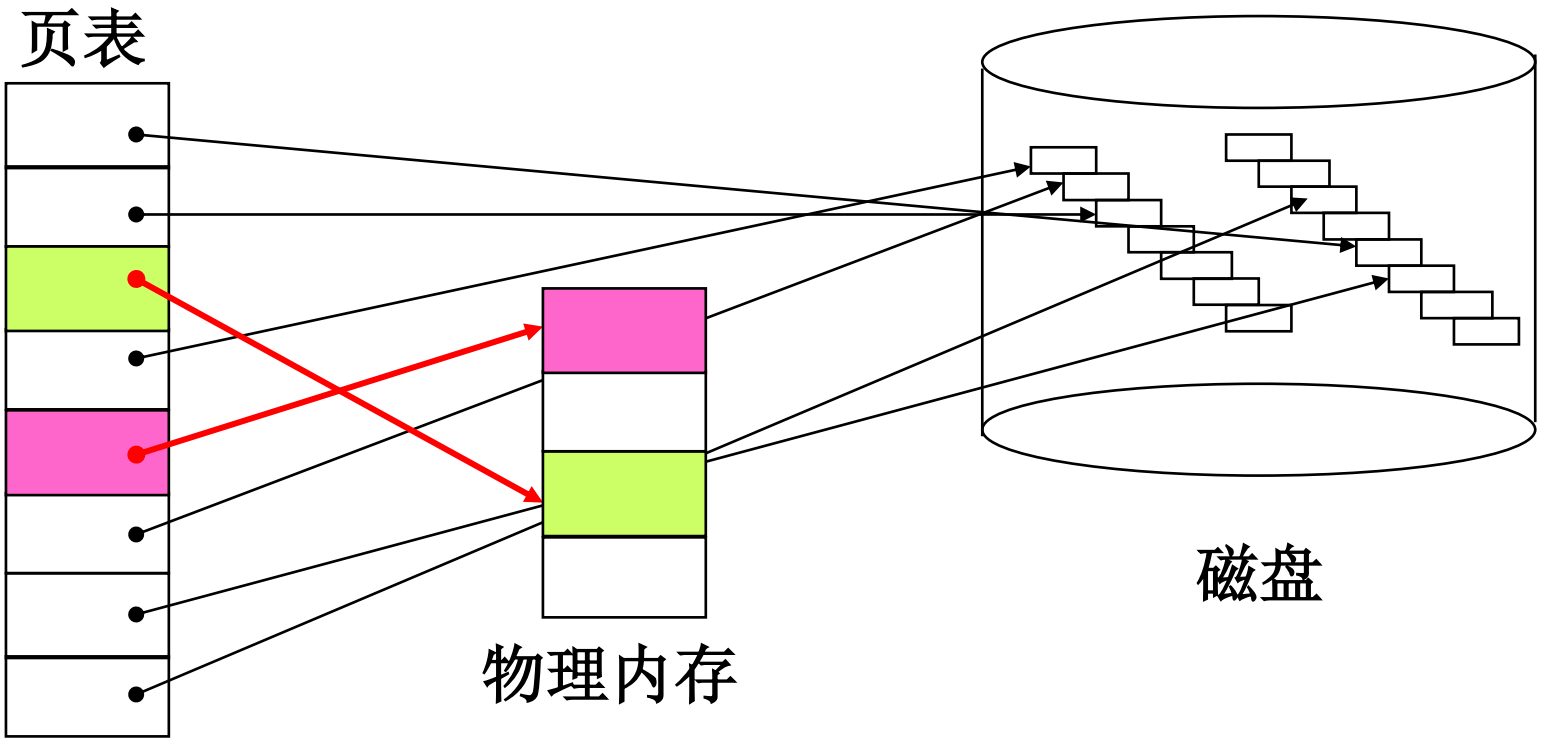
“请求调页”或者“请求调段”

请求调页！
(Demand Paging)

虚拟内存中的页面映射关系

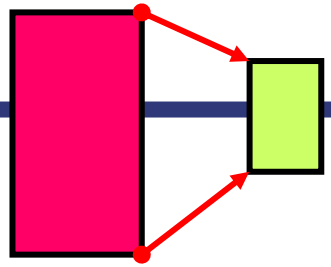


- 部分线性地址(逻辑页)对应物理页，那其它页呢？



页框号(物理页号)ppn	保留	0	L	D	A	PCD	PWT	U	W	P
31-12	11-9	8	7	6	5	4	3	2	1	0

请求调页过程



- 当访问没有映射（载入）的线性地址时…

显然是一个很好理解的过程

load [addr]

(1)

页表

(2)

页错误处理程序

(3)

磁盘

(4)

(5)

(6)

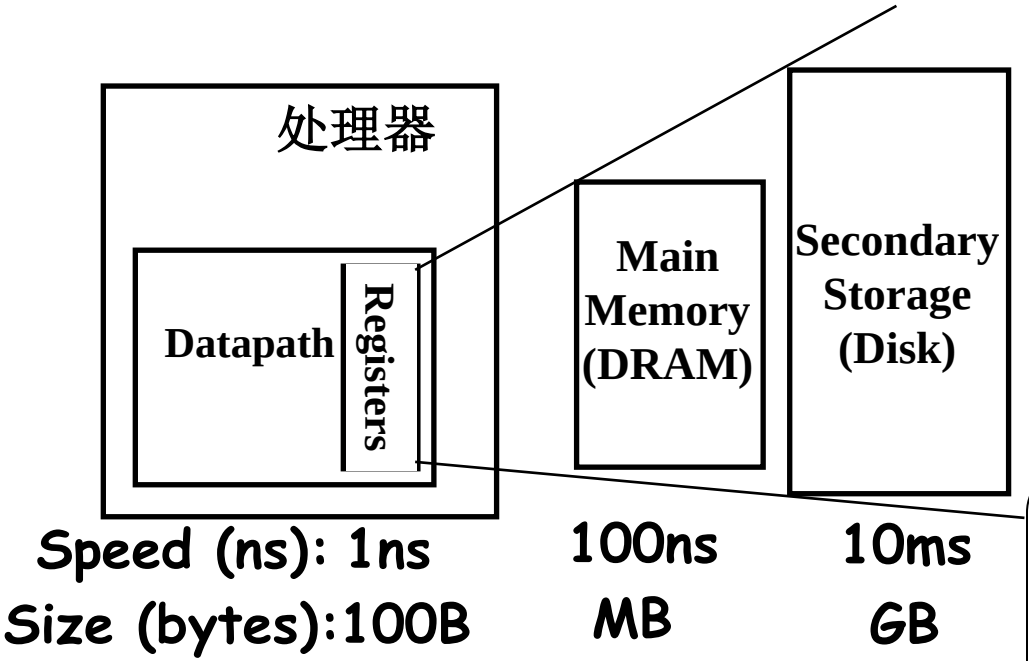
但完成这个过程很费时间（有时候一条指令会引起几次调页）！

物理内存

请求调页的性能分析

决定了请求调页是否可用

- 分析的背景: 又一个计算机基本特征!



存储器层次

100(1-p)+10000000p=
100+9999900p ≤ 110
➔p<0.000001

- 请求调页时的有效访问时间

有效访问时间=(1-p)×MA+p×调页时间

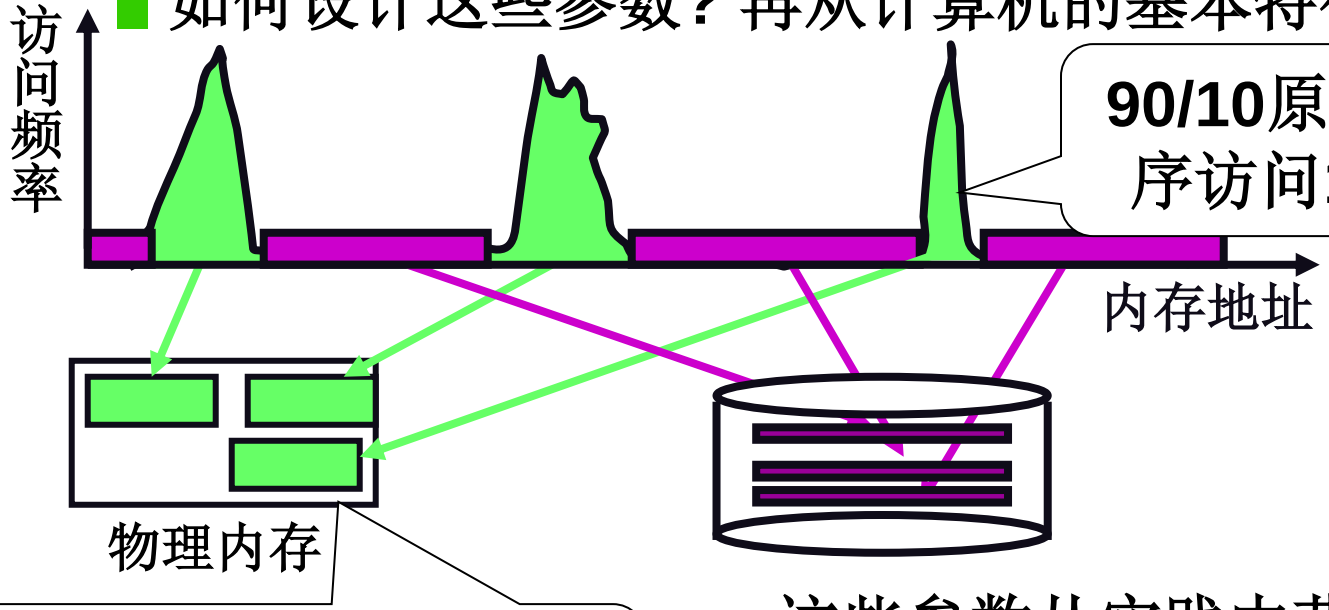
- 缺页率(页错误率)应该很小: $1/10^5$

如何降低缺页率？

- 分配给一个进程的物理页框数应该足够多!
- 页面应该足够大!

分别违背了请求调页和分页的原则，又需要折衷!

- 如何设计这些参数? 再从计算机的基本特征开始!

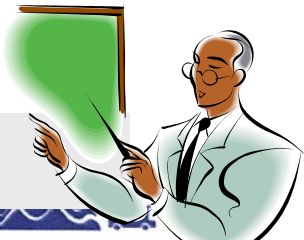


90/10原则! 90%的程序访问10%的地址!

页面4K，调入一页后许多指令不出现页错误

- 这些参数从实践中获得!

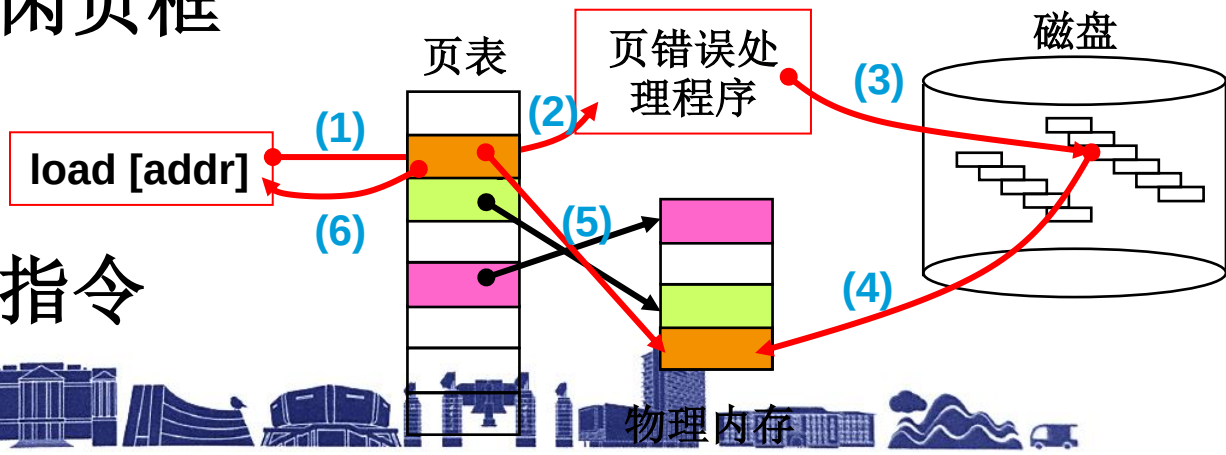
实践性很强的学科怎么学?



请求调页的具体实现细节

- (1): load [addr]，而addr没有映射到物理内存
  - 根据addr查页表，页表项的P位为0（页不在内存），引起缺页中断(page fault)
- (2): 设置“缺页中断”即可
- (3): “缺页中断处理程序”需要读磁盘
- (4): 选一个空闲页框
- (5): 修改页表
- (6): 重新开始指令

学过磁盘处理后
自然就明白了!



请求页式存储管理的调入策略

- 何时调入页面
  - 预调
  - 请调
- 从何处调入
  - 进程的所有页面都放在对换区。
  - 只将修改过的页面放在对换区，未改的放在文件区。
  - UNIX系统方式，首次从文件区调入，换出时放在对换区，以后从对换区调入。

如何选一个空闲页框？页面淘汰(置换)

- 没有空闲页框怎么办？分配的页框数是有限的
  - 需要选择一页淘汰
  - 有多种淘汰选择。如果某页刚淘汰出去马上又要用…
  - FIFO，最容易想到，怎么评价？
  - 有没有最优的淘汰方法，OPT
  - 最优淘汰方法能不能实现，能否借鉴思想，LRU
  - 再来学习几种经典方法，它可以用在许多需要淘汰(置换)的场合…

FIFO页面置换

- 淘汰算法: FIFO
  - 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
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

OPT页面置换

- **OPT算法: 选最远将使用的页淘汰。是一种最优的方案，可以证明缺页数最小!**
  - 继续: 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，OPT导致5次缺页
  - 可惜，OPT需要知道将来发生的事… 怎么办?

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OPT完全一样!

LRU的准确实现

- 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

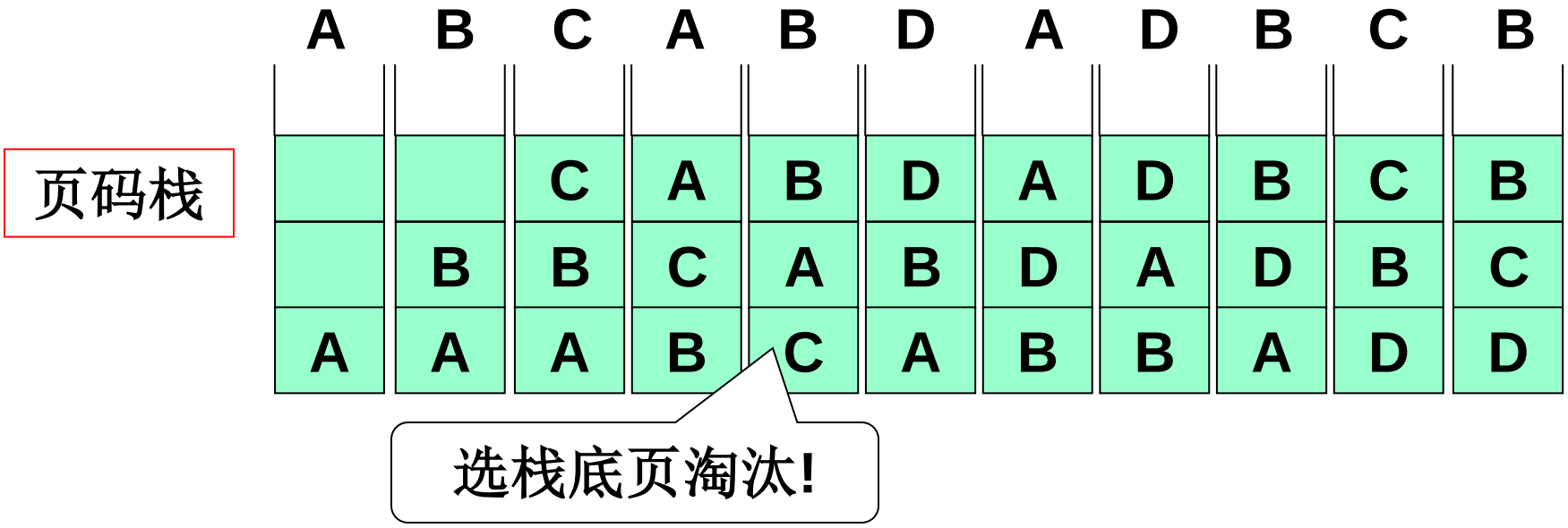
选具有最小时间戳的页!

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(该时钟溢出怎么办?)，需要找到最小值 … 这样的实现代价较大 ⇒ 几乎没人用

LRU准确实现之页码栈

- 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

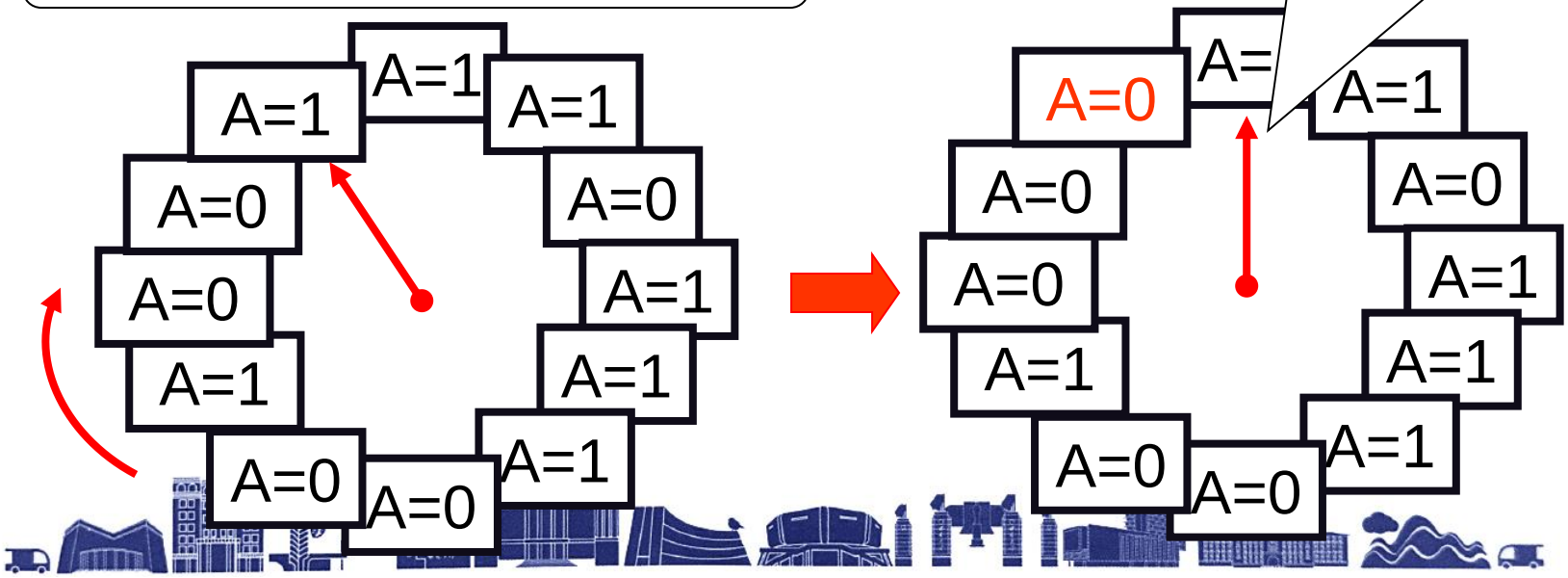
LRU近似实现 – 将时间计数变为是和否

- 每个页加一个访问位(access 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

再给一次机会 (Second Chance Replacement)

组织成循环队列较合适！

称为Clock Algorithm



Clock置换算法(NRU)

- 为每页设置一个访问位，内存中的所有页面都通过链接指针链成一个循环队列。当页被访问时，该位置为1。页面置换时，首先淘汰访问位为0的页；若访问位为1，则重新置为0，暂不换出。当队列中的最后一个页面仍为1，则返回到队首重新检查。

块号	页号	访问位	指针
0			
1			
2	4	0	
3			
4	2	1	
5			
6	5	0	
7	1	1	

替换指针

块号	页号	访问位	指针
0			
1			
2	4	0	
3			
4	2	0	
5			
6	5	0	
7	1	0	

替换指针

Clock算法分析的改造

- 在改进的Clock增加了一个M位， M=0 表示该页未被修改过。这样我们选择页面换出时，既要最近未访问过的页面，又要未被修改过的页面。其执行过程分一下三步：
  - 1. 从开始位置循环扫描队列，寻找A=0、M=0的第一类面，找到立即置换。另外，第一次扫描期间不改变访问位A。
  - 2. 如果第一步失败，则开始第二轮扫描，寻找A=0且M=1的第二类页面，找到后立即置换，并将所有扫描过的A都置0。
  - 3. 如果第二步也失败，则返回指针开始位置，然后重复第一步，必要时再重复第二步，此时必能找到淘汰页。

改进的Clock置换算法

页面	访问位	修改位
0	1	0
1	0	1
2	1	1
3	1	1

初始状态

页面	访问位	修改位
0	1	0
1	0	1
2	1	1
3	1	1

第一次扫描
无(0,0)页面

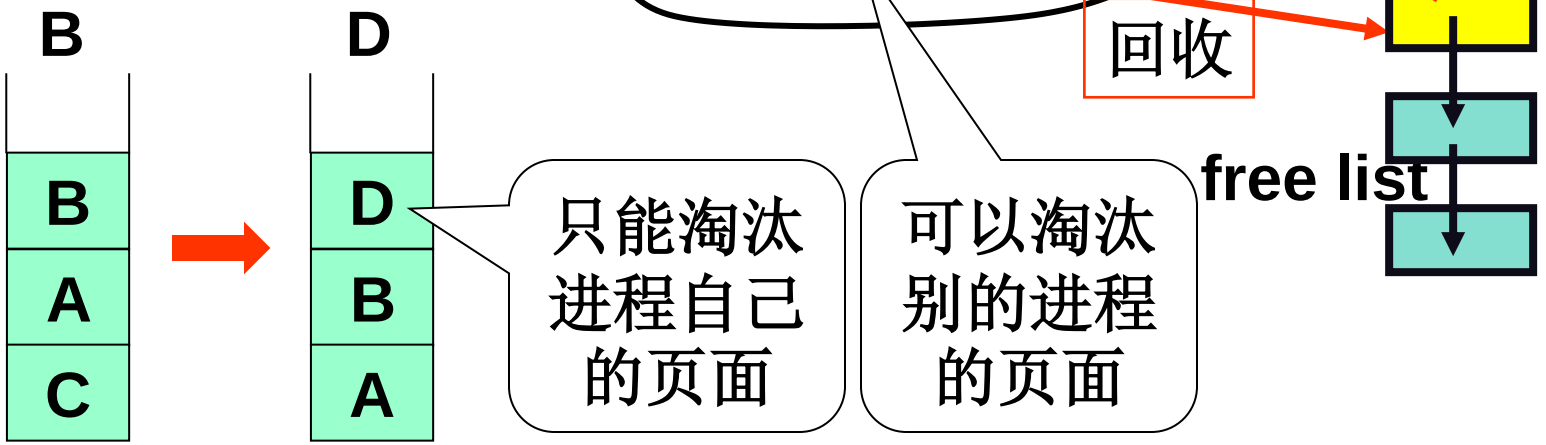
页面	访问位	修改位
0	0	0
1	0	1
2	1	1
3	1	1

第二次扫描
找到(0,1)页面

它会被选择换出

两种置换策略: 全局置换和局部置换

- 复习一下：
  - 想一想前面的例子!

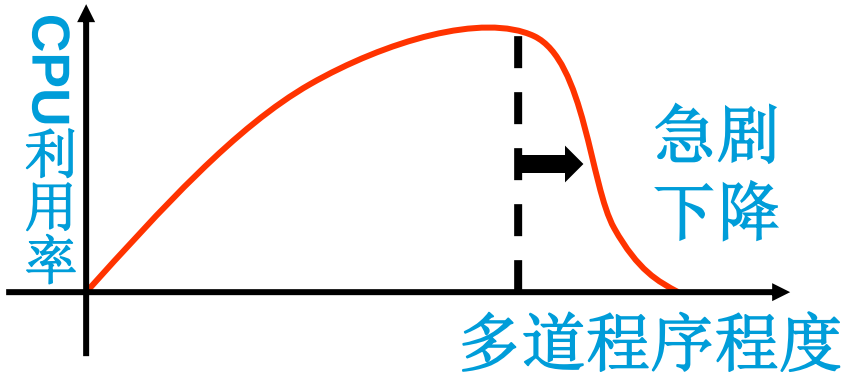


  - 全局置换: 实现简单
  - 但全局置换不能实现公平、保护

局部置换需要考虑的关键问题

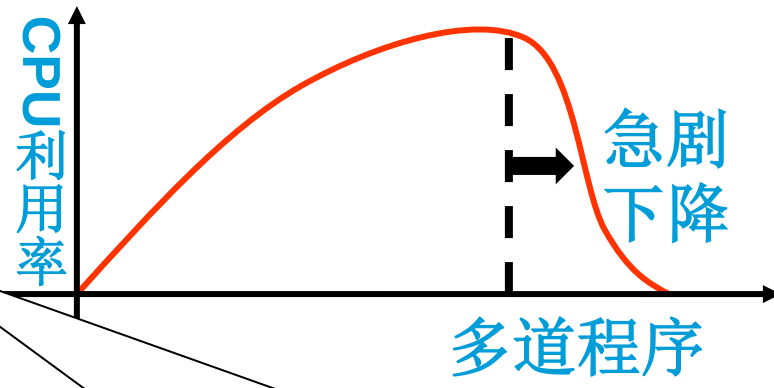
- 给进程分配多少页框(帧frame)
  - 分配的多，请求调页的意义就没了！一定要少?
  - 至少是多少? 至少可执行任意一条指令
  - 是不是就选该下界值?
  - 来看一个实例: 操作系统监视CPU使用率，发现CPU使用率太低时，向系统载入新进程。

为什么会发生程序数目越多，CPU利用率反而更低了?



CPU利用率急剧下降的原因

- 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 …
- 称这一现象为抖动(thrashing)



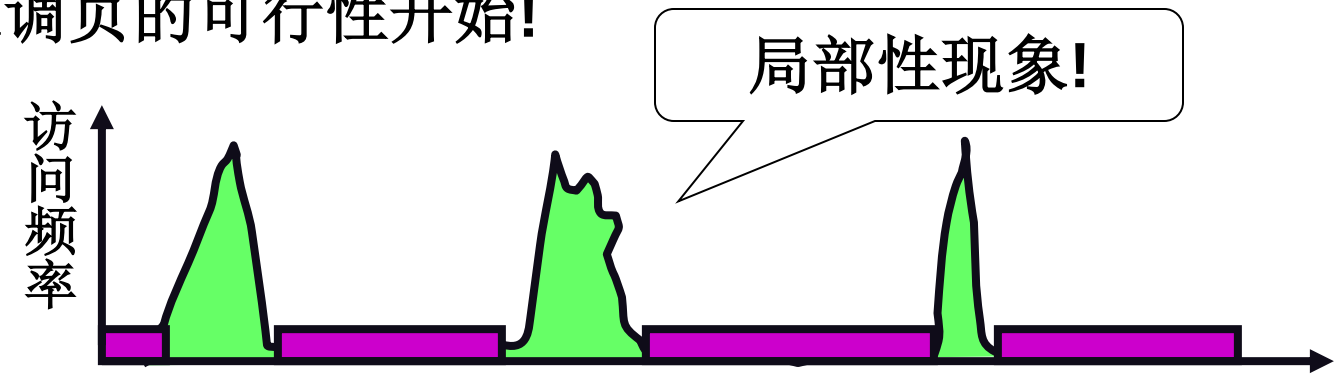
此时: 进程调入一页，需将一页淘汰出去，刚淘汰出去的页马上要需要调入，就这样……

- 显然，防止的根本手段给进程分配足够多的帧

问题时怎么确定进程需要多少帧才能不抖动?

工作集模型（The model of work set）

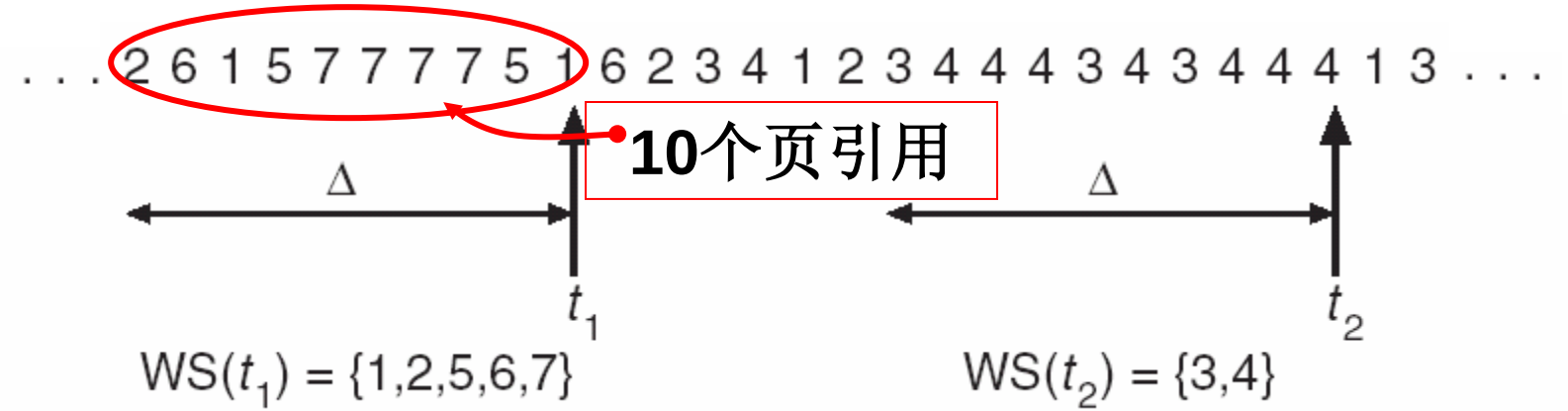
- 任何计算都需要一个模型! 要确定进程所需的帧数该依靠什么信息呢?
  - 从请求调页的可行性开始!



  - 只要分配的帧空间能覆盖整个局部就不会出现太多的缺页!
  - 工作集模型就用来计算一个局部的宽度(帧数)

工作集定义

- 某段时间间隔△内，进程实际要访问的页面的集合。具体地说，某进程在时间 t 的工作集记为W(t, △)，把变量△称为工作集“窗口尺寸”。W表示此期间所访问的不同页面的集合。
  - 一个例子: Δ定义为10个页引用数!



  - WS_i的用法: (1)计算D= Σ|WS_i|; (2)如果D>m，则选择一个进程换出; (3)如果D<m，可以选一个进程换入。

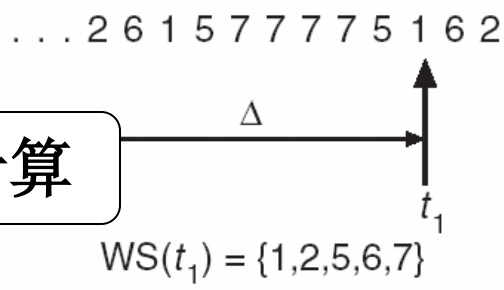
选择哪个进程换入、换出，中程调度

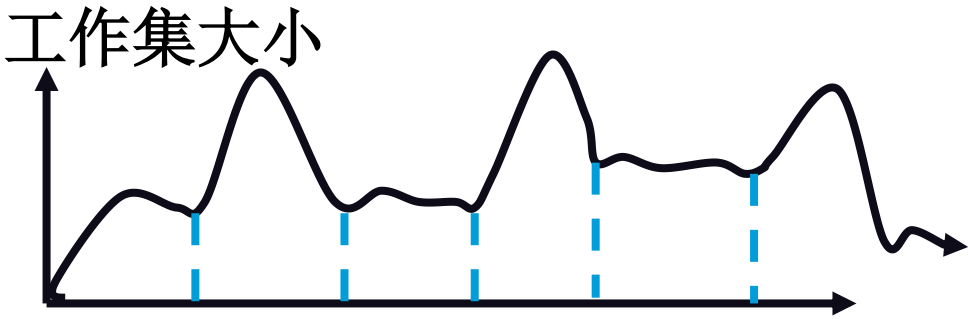
工作集的计算

- 根据定义，每次引用都重新计算WS，会很低效
  - 定期扫描+定期计算(在定时中断中)

UNIX, 扫描周期: 几秒.计算周期: 几分钟.

是近似计算



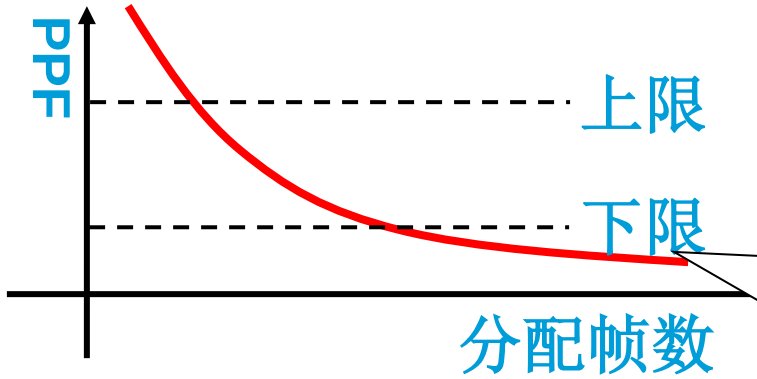


  - Δ该定为多少? 太小盖不住一个局部，太大会包含多个局部。 试试看? 但系统有时并不敏感
  - 提出了基于页错误率的帧分配方案

基于页错误率的帧分配

- 页错误率(PFF) = 页错误/指令执行条数
  - 如果PFF>上限，增加分配帧数

如果没有空闲帧，则换出进程



有趣的是，帧数越多，PPF并不一定下降

  - 此种方法简单直接，在处理颠簸时常用。
  - 往往是PFF和WS互相配合
  - 但现代OS并不十分重视颠簸现象，因为CPU更快了，进程很快exit；内存更大了，局部的变化不大

Belady异常

- 来看一个例子!
  - 引用序列1，2，3，4，1，2，5，1，2，3，4，5

FIFO页置换

3frame

9 faults

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

4frame

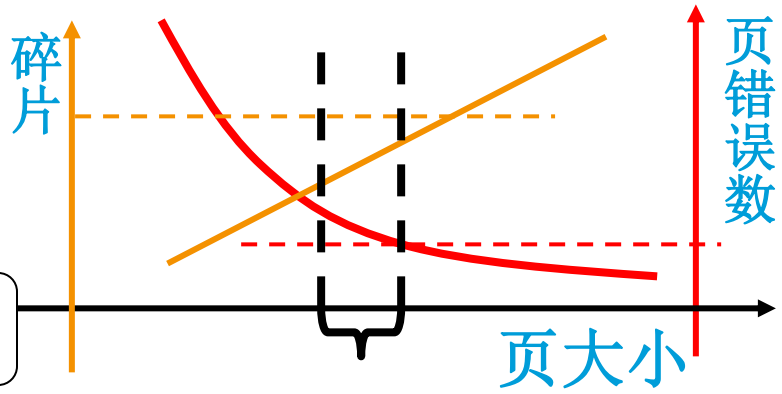
10 faults

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

一些技术

启动快，但可能有页用不着!

- 预调页：页可以不同自己的页错误而调入
  - 进程创建(换入)时一次调入多个页(可由WS确定)
- 页面尺寸：该定为多少?受许多因素影响!
  - 减少碎片，页应该小
  - 降低页错误，页应大



- 程序结构：

按行存储

  - for(j..128){for(i..128){A[i][j]=0;}} //页大小128
  - for(i..128){for(j..128){A[i][j]=0;}}

128*128 faults

128faults

操作系统难吗?不难吗?
难吗?不难吗?

整理一下前面的学习

温故而知新

- 虚拟内存的基本思想
  - 将进程的一部分(不是全部)放进内存
  - 其他部分放在磁盘
  - 需要的时候调入: 请求调页

what?

内存利用率高，
程序编制容易，
响应时间快…

why?

- 请求调页的基本思想
  - 发现页不在内存时，中断CPU
  - CPU处理此中断，找到一个空闲页框
  - CPU将磁盘上的页读入到该页框
  - 如果没有空闲页框需要置换某页(LRU)

how?

虚拟内存总结

- 内存的根本目的 ⇒ 把程序放在内存并让其执行
- 只要将部分程序放进内存即可执行 ⇒ 内存利用率高
- 可编写比内存大的程序 ⇒ 使用一个大地址空间(虚拟内存)
- 部分程序在内存 ⇒ 其他部分在磁盘 ⇒ 需要的时候调入内存
- 页表项存在P位 ⇒ 缺页产生中断 ⇒ 中断处理完成页面调入
- 调入页面需要一个空闲页框 ⇒ 如果没有空闲页框 ⇒ 置换
- 置换方法 ⇒ **FIFO→OPT→LRU→Clock算法**
- 需要给进程分配页框 ⇒ 全局、局部 ⇒ 抖动⇒ 工作集

参考资料

- 廖剑伟，操作系统，西南大学，重庆，2023