第9章　虚拟存储器

目录

9.1 虚拟存储器概念

- 前章中，物理存储器管理方式要求作业运行前全部装入内存，作业装入内存后一直驻留内存直至运行结束，或者该作业被换出。

- 这种物理存储器管理方式限制了大作业的运行。而物理扩充内存会增加成本，故应从逻辑上扩充内存。

打破作业全部装入内存才能运行的限制

关于程序执行的特点

1. 程序中只有少量分支和过程调用，大都是顺序执行的，即要取的下一条指令紧跟在当前执行指令之后。
2. 程序往往包含若干个循环，这些是由相对较少的几个指令重复若干次组成的，在循环过程中，计算被限制在程序中一个很小的相邻部分中（如计数循环）。
3. 很少会出现连续不断的过程调用序列，相反，程序中过程调用的深度限制在一个小的范围内，因而，一段时间内，指令引用被局限在很少几个过程中。
4. 对于连续访问数组之类的数据结构，往往是对存储区域中的数据或相邻位置的数据（如动态数组）的操作。
5. 程序中有些部分是彼此互斥的，不是每次运行时都用到的，如，出错处理程序，仅当在数据和计算中出现错误时才会用到，正常情况下，出错处理程序不放在主存，不影响整个程序的运行。。

局部性原理

- 上述种种情况说明，作业执行时没有必要把全部信息同时存放在主存储器中，而仅仅只需装入一部分。这一原理称为“局部性原理”。

- 局部性原理
  - CPU访问存储器时，无论是存取指令还是存取数据，所访问的存储单元都趋于聚集在一个较小的连续区域中。

局部性的体现

- 局部性体现为：
  - 时间局部性：一条指令的一次执行和下次执行，一个数据的一次访问和下次访问，都集中在一个较短时间内。
  - 空间局部性：当前执行的指令和将要执行的指令，当前访问的数据和将要访问的数据，都集中在一个较小范围内。
  - 顺序局部性：顺序执行与跳转比例5：1

- 虚拟存储器的理论基础是程序执行时的局部性原理。

虚拟存储器的基本原理

- 在程序运行之前，将程序的一部分放入内存后就启动程序执行。
- 在程序执行过程中，当所访问的信息不在内存时，由操作系统将所需要的部分调入内存，然后继续执行程序。
- 另一方面，操作系统将内存中暂时不使用的内容换出到外存上，从而腾出空间存放将要调入内存的信息。
- 从效果上看，这样的计算机系统好像为用户提供了一个存储容量比实际内存大得多的存储器，将这个存储器称为虚拟存储器。

虚拟存储器

- 虚拟存储器是指具有请求调入和置换功能，能从逻辑上对内存容量加以扩充的一种存储器系统。

- 虚拟存储器是一种以时间换空间的技术。

虚拟存储器的特征

- 离散性：
  - 不连续内存分配
- 多次性：
  - 一个作业分多次装入内存
- 对换性：
  - 允许运行中换进换出
- 虚拟性：
  - 逻辑上扩充内存

虚拟存储器的本质

- 虚拟存储器的本质是将程序的访问地址和内存的可用地址分离，为用户提供一个大于实际主存的虚拟存储器。
- 虚拟存储器的容量受限于：
  - 地址结构
  - 外存容量

实现虚拟存储技术的物质基础

- 相当数量的外存：足以存放多个用户的程序。
- 一定容量的内存：在处理机上运行的程序必须有一部分信息存放在内存中。
- 地址变换机构：动态实现逻辑地址到物理地址的变换。

虚拟存储器的实现方法

- 常用的虚拟存储技术有：
  - 请求分页存储管理
  - 请求分段存储管理

9.2 请求分页存储管理

- 请求分页存储管理方法是在分页存储管理的基础上增加了请求调页和页面置换功能。

- 实现思想：
  - 在作业运行之前只装入当前需要的一部分页面便启动作业运行。
  - 在作业运行过程中，若发现所要访问的页面不在内存，便由硬件产生缺页中断，请求OS将缺页调入内存。
  - 若内存无空闲存储空间，则根据某种置换算法淘汰已在内存的某个页面，以腾出内存空间装入缺页。

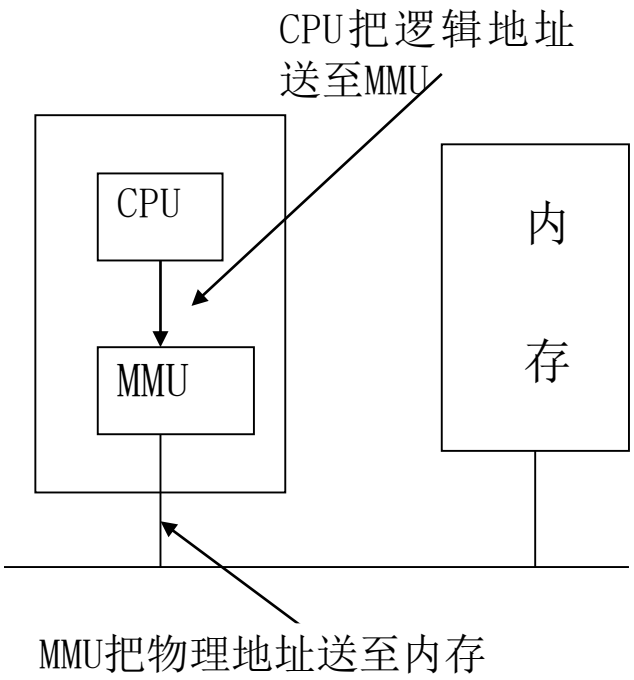
请求分页系统的支持机构

- 请求分页的支持机构有:
  - 物理部件：内存管理单元 MMU(Memory Management Unit)
  - 页表
  - 缺页中断机构
  - 地址变换机构
  - 请求调页和页面置换软件

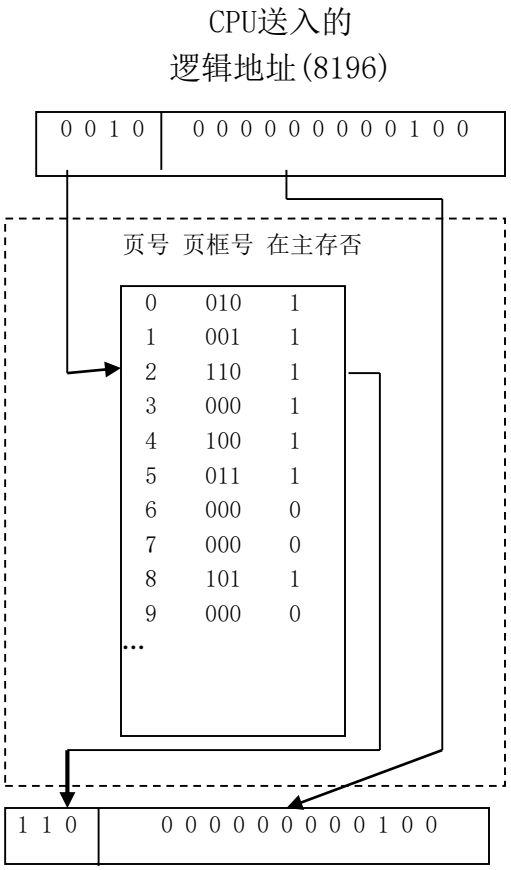
9.2.1内存管理单元MMU

- 内存管理单元MMU完成逻辑地址到物理地址的转换功能，它接受**虚拟地址作为输入，物理地址作为输出，直接送到总线上，对内存单元进行寻址**。

MMU原理



MMU的位置、功能和16个4KB页面情况下MMU的内部操作



MMU主要功能

① 管理硬件页表基址寄存器。
② 分解逻辑地址。
③ 管理快表TLB。
④ 访问页表。
⑤ 发出缺页中断或越界中断，并将控制权交给内核存储管理处理。
⑥ 管理特征位，设置和检查页表中各个特征位。

9.2.2 页表

- 请求分页系统中使用的主要数据结构仍然是页表。
- 但由于每次只将作业的一部分调入内存，还有一部分内容存放在磁盘上，故需要在页表中增加若干项。
- 扩充后的页表项如下所示：

扩充后的页表项

页号	物理块号	存在位	访问字段	修改位	外存地址

- 页号和物理块号：其定义同分页存储管理。
- 存在位：用于表示该页是否在主存中。
- 访问字段：用于记录本页在一段时间内被访问的次数，或最近已有多长时间未被访问。
- 修改位：用于表示该页调入内存后是否被修改过。
- 外存地址：用于指出该页在外存上的地址。

9.2.3 缺页中断与地址变换

- 在请求分页系统中，硬件查页表发现所访问的页不在内存时，便产生缺页中断，请求OS将缺页调入内存
- 操作系统执行缺页中断处理程序根据该页在外存的地址把它调入内存
  - 在调页过程中，若内存有空闲空间，则缺页中断处理程序只需把缺页装入并修改页表中的相应项
  - 若内存中无空闲物理块，则需要先淘汰内存中的某些页；若淘汰页曾被修改过，则还要将其写回外存

缺页中断的处理过程

① 发起对地址的访问，MMU到页表中检查引用情况

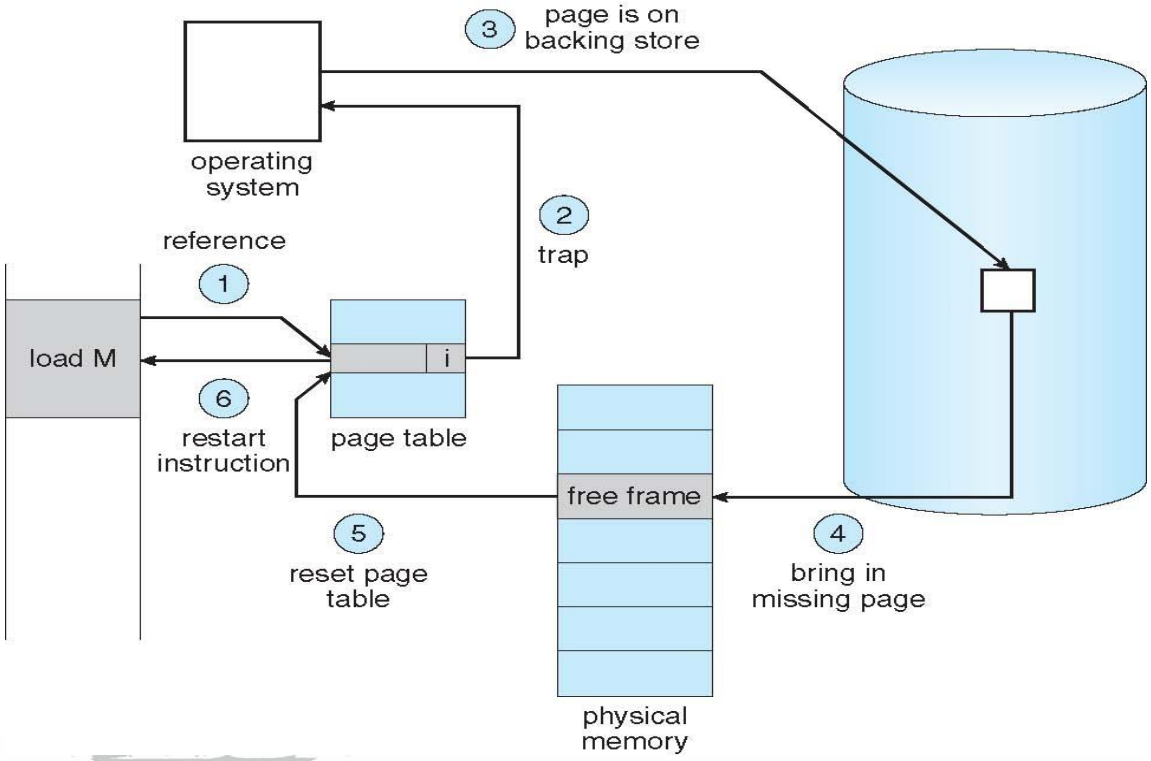
② 若不在内存，产生缺页中断，陷入缺页中断程序

③ OS在外存中寻找外存中的页面备份

④ 寻找空闲页帧（可能需要依据某种替换算法选择被替换的页帧），将页面调入内存

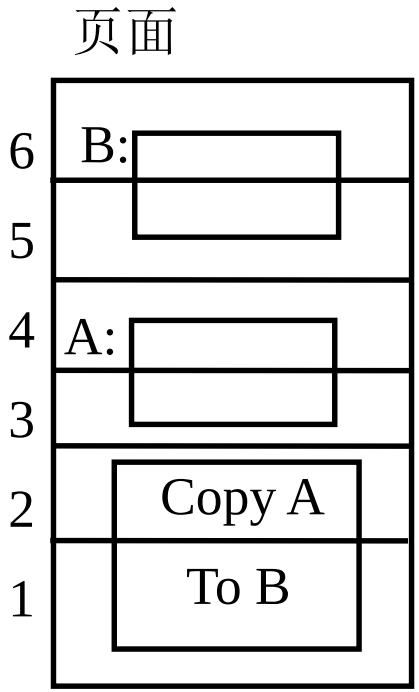
⑤ 修改页表项信息

⑥ 重新执行产生缺页的指令



缺页中断与一般中断的区别

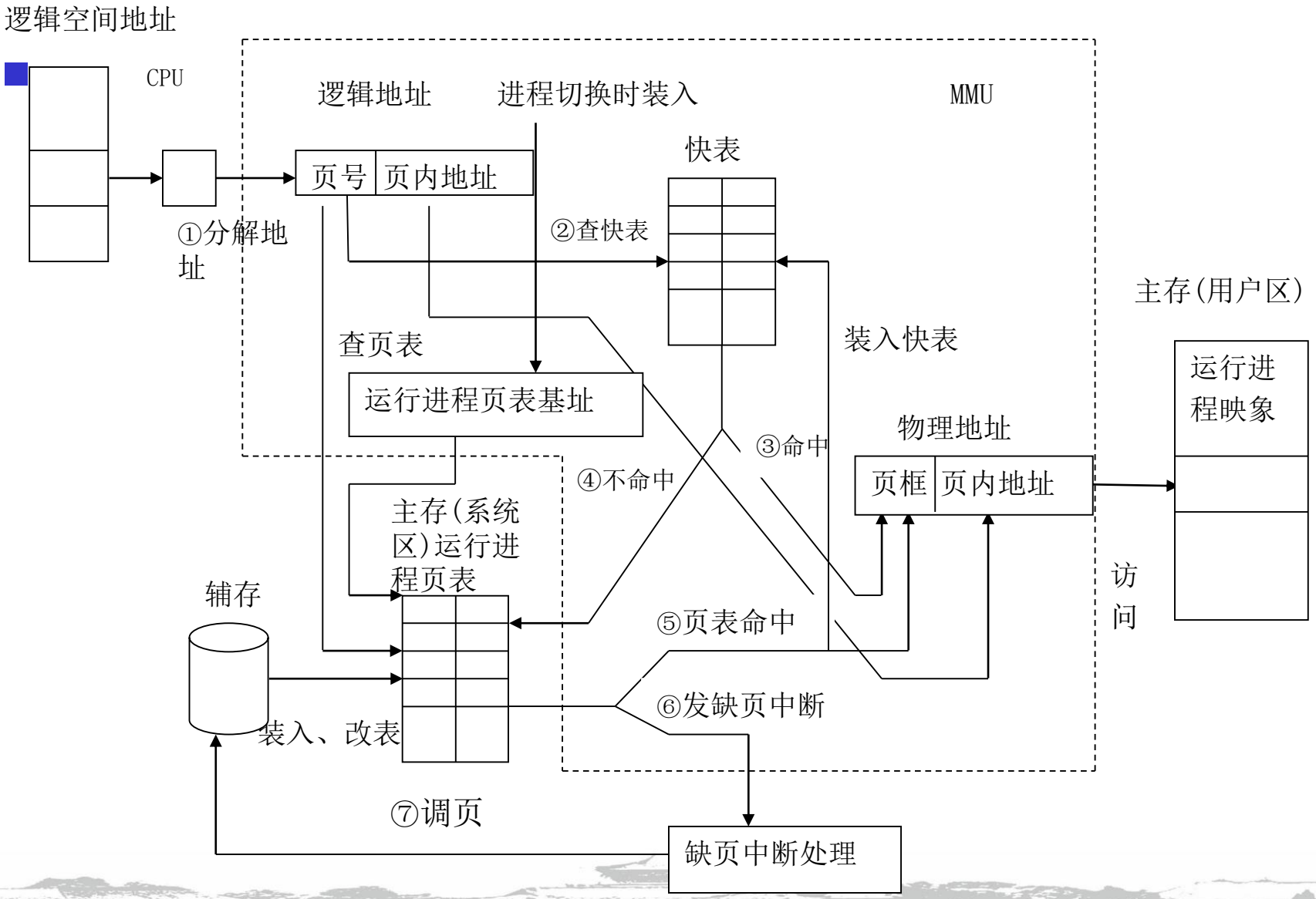
- 缺页中断与一般中断的区别主要有：
  - 在指令的执行期间产生和处理缺页中断。
  - 一条指令可以产生多个缺页中断。如：执行一条复制指令copy A to B
  - 缺页中断返回时执行产生中断的指令，一般中断返回时执行下条指令



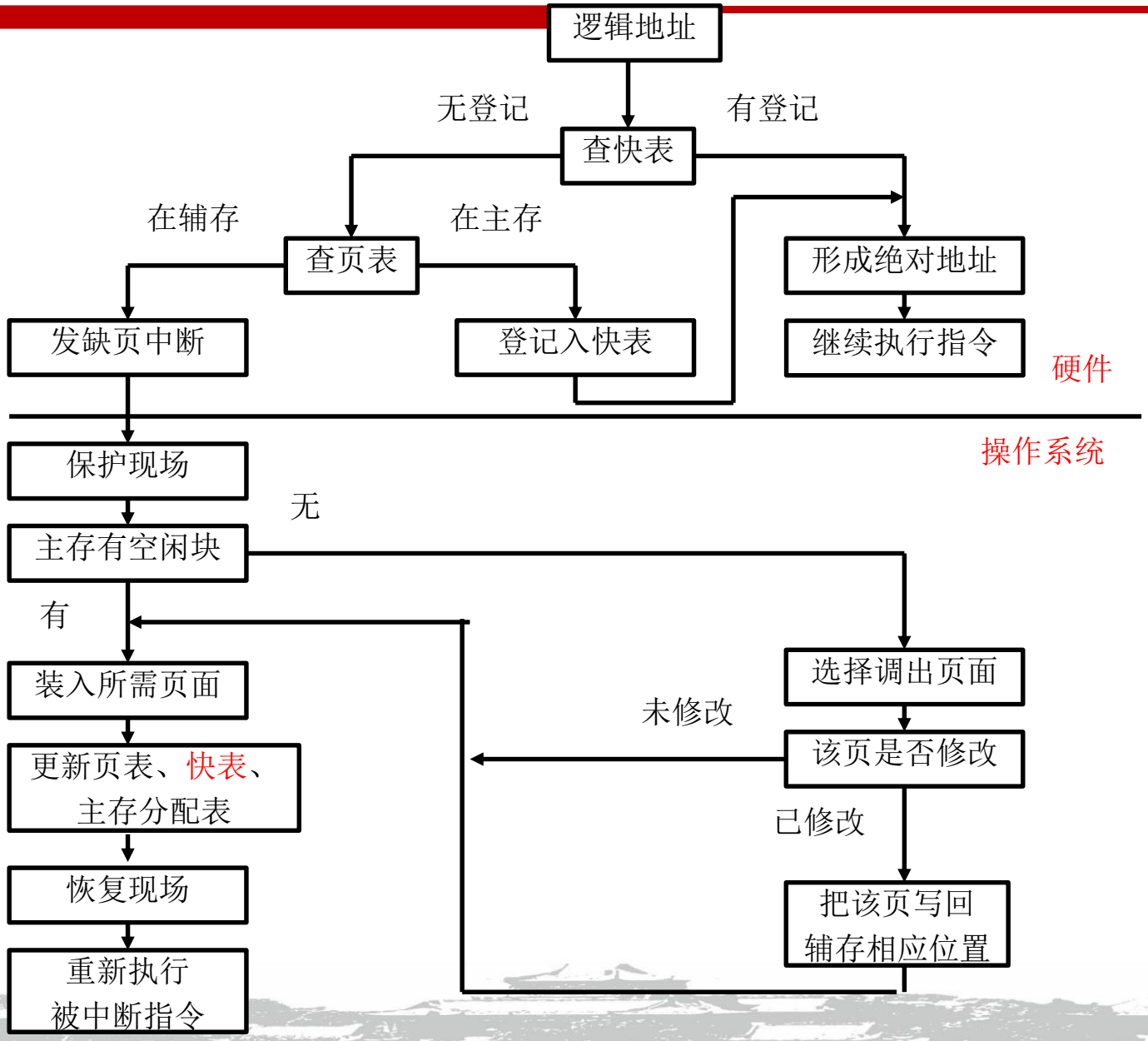
地址变换

- 请求分页虚拟存储管理系统的地址变换过程类似于分页存储管理，但**当被访问页不在内存时**应进行**缺页中断处理**。

地址变换过程


逻辑空间地址
CPU
逻辑地址
进程切换时装入
MMU
快表
页号
页内地址
①分解地址
②查快表
主存(用户区)
查页表
装入快表
运行进程页表基址
运行进程映象
物理地址
③命中
④不命中
页框
页内地址
主存(系统区)运行进程页表
辅存
访问
⑤页表命中
⑥发缺页中断
装入、改表
⑦调页
缺页中断处理

地址变换过程（续）


逻辑地址
无登记
有登记
查快表
在辅存
在主存
查页表
形成绝对地址
发缺页中断
登记入快表
继续执行指令
硬件
操作系统
保护现场
无
主存有空闲块
有
装入所需页面
选择调出页面
未修改
更新页表、快表、
主存分配表
该页是否修改
已修改
恢复现场
把该页写回
辅存相应位置
重新执行
被中断指令

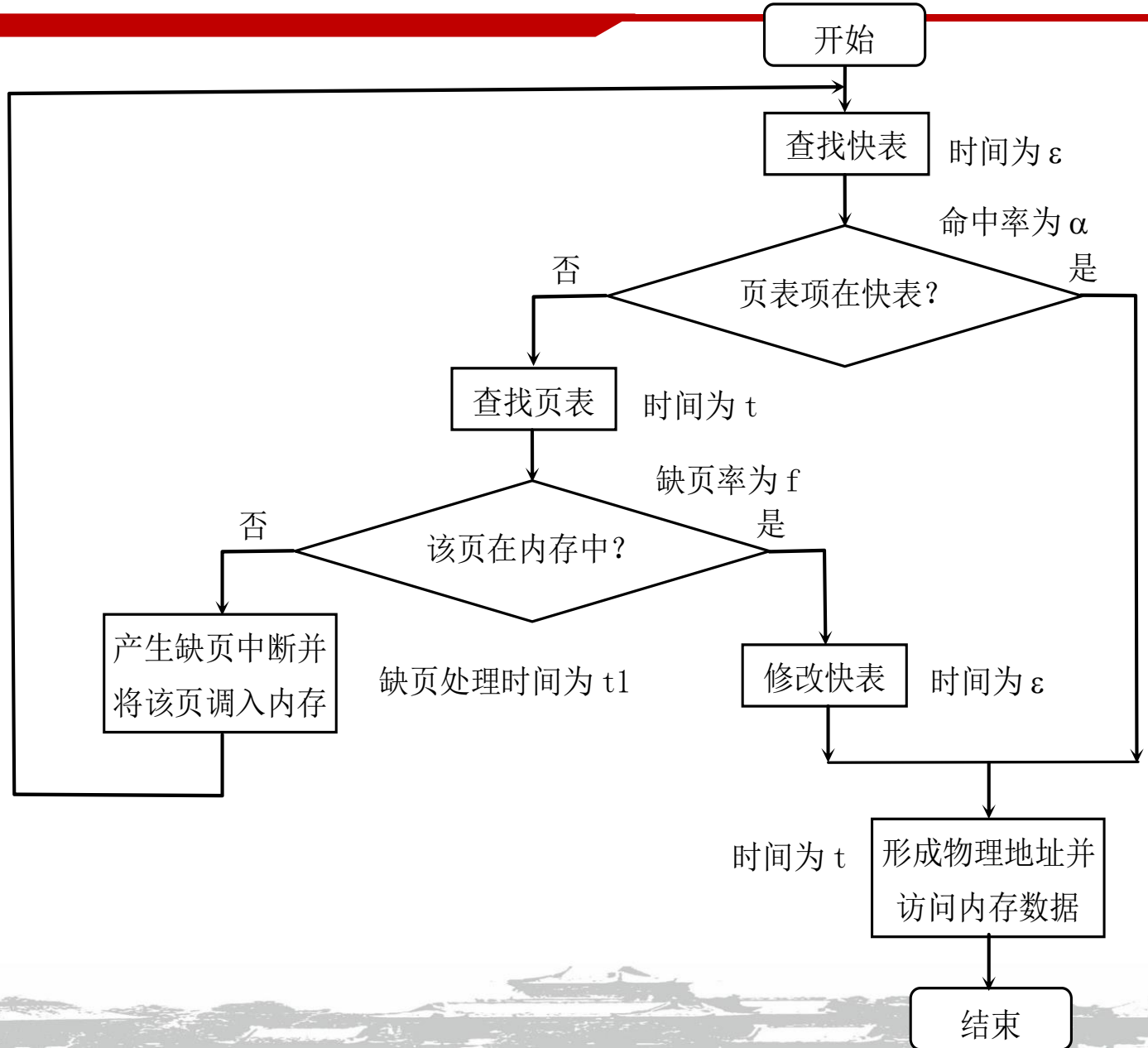
关于快表（TLB）的讨论

- 假设程序刚引用了一个虚存地址，描述一下的可能场景
  - TLB未命中，没有缺页错误
  - TLB未命中，有缺页错误
  - TLB命中，没有缺页错误
  - TLB命中，有缺页错误

TLB的刷新问题

- 为什么要刷新TLB
  - 进程切换
  - 请求调页与页面替换
    - 被替换页的页表项，如果存在于TLB中，则该TLB是脏的
- 现代计算机系统支持对TLB的刷新问题
  - 硬件管理TLB
    - TLB不命中后，CPU自动查找页表并更新TLB
  - 软件管理TLB
    - TLB不命中触发异常，OS负责查找页表并更新TLB

内存访问时间的评价


开始
查找快表
时间为 ε
命中率为 α
否
页表项在快表?
是
查找页表
时间为 t
缺页率为 f
否
该页在内存中?
是
产生缺页中断并
将该页调入内存
缺页处理时间为 t1
修改快表
时间为 ε
时间为 t
形成物理地址并
访问内存数据
结束

访问内存的时间

- 若页在主存中且页表项在快表中：
  访问时间=查快表时间+访问内存时间=ε+t。

- 若页在主存中且页表项不在快表中：
  - 访问时间=查快表时间+查页表时间+修改快表时间+访问内存时间=ε+t+ε+t=2(ε+t)

- 若页不在主存中，处理缺页中断的时间为t1(包含读入缺页、页表更新、快表更新时间)：
  - 访问时间=查快表时间+查页表时间+处理缺页中断时间t1+查快表时间+访问内存时间=ε+t+t1+ε+t=tl+2(ε+t)

有效访问时间EAT的计算

- “快表”的命中率为α，缺页中断率为f，则有效存取时间可表示为：
  - EAT= α *(ε +t)+(1- α)* [(1-f) *2 (ε +t) +f*(tl+ 2 (ε +t))]

请求页式虚拟存储系统优缺点

- 优点：
  - 作业的程序和数据可按页分散存放在内存中，减少移动开销，有效解决了碎片问题;
  - 既有利于改进主存利用率，又有利于多道程序运行。
- 缺点：
  - 要有硬件支持，要进行缺页中断处理，机器成本增加，系统开销加大。

9.3 页面管理的策略问题

1. 页面装入与清除策略

2. 页面的分配与替换策略

1、页面装入和清除策略

- 页面装入策略：决定何时把页面装入主存
- 有两种策略：
  - 请求式调度(demand paging)
    - 按需装入
    - 频繁磁盘I/O
  - 预调式调度(prepaging)
    - 局部性原理，动态预测，预先装入；
    - Windows代码页面预调3-8页，数据页面2-4页

页面装入和清除策略

- 页面清除策略：决定**何时**把一个修改过的页面**写回辅存储器**
- 有两种策略：
  - 请求式清除
    - 当一页被选中进行替换，且被修改过时，则进行写回磁盘
    - 缺点：效率低下
  - 预约式清除
    - 对所有修改的页面，替换前，提前成批写回
    - 要写回的页仍然在主存，直到被替换算法选中此页从主存中移出。
    - 若该页面在刚被写回后，在替换回前，再次被大量修改，则该策略失效。

页面装入和清除策略

- 页面装入策略还需要考虑从何处调入页面?
- 文件区
  - 用于存放文件;
  - 采用离散分配方式。
- 对换区
  - 用于存放对换页面;
  - 采用连续分配方式。
  - 故对换区的磁盘I/O速度比文件区的高。

从何处调入页面（续）

- 发生缺页请求时，系统应从何处将缺页调入内存，可分成如下三种情况：
- (1) 系统拥有足够的对换区空间。
  - 这时可以全部从对换区调入所需页面，以提高调页速度。
  - 在进程运行前， 便须将与该进程有关的文件，从文件区拷贝到对换区。

从何处调入页面（续）

- (2) 系统缺少足够的对换区空间。
  - 这时凡是不会被修改的文件，都直接从文件区调入；而当换出这些页面时，由于它们未被修改而不必再将它们换出，以后再调入时，仍从文件区直接调入。
  - 但对于那些可能被修改的部分，在将它们换出时，便须调到对换区，以后需要时，再从对换区调入。

从何处调入页面（续）

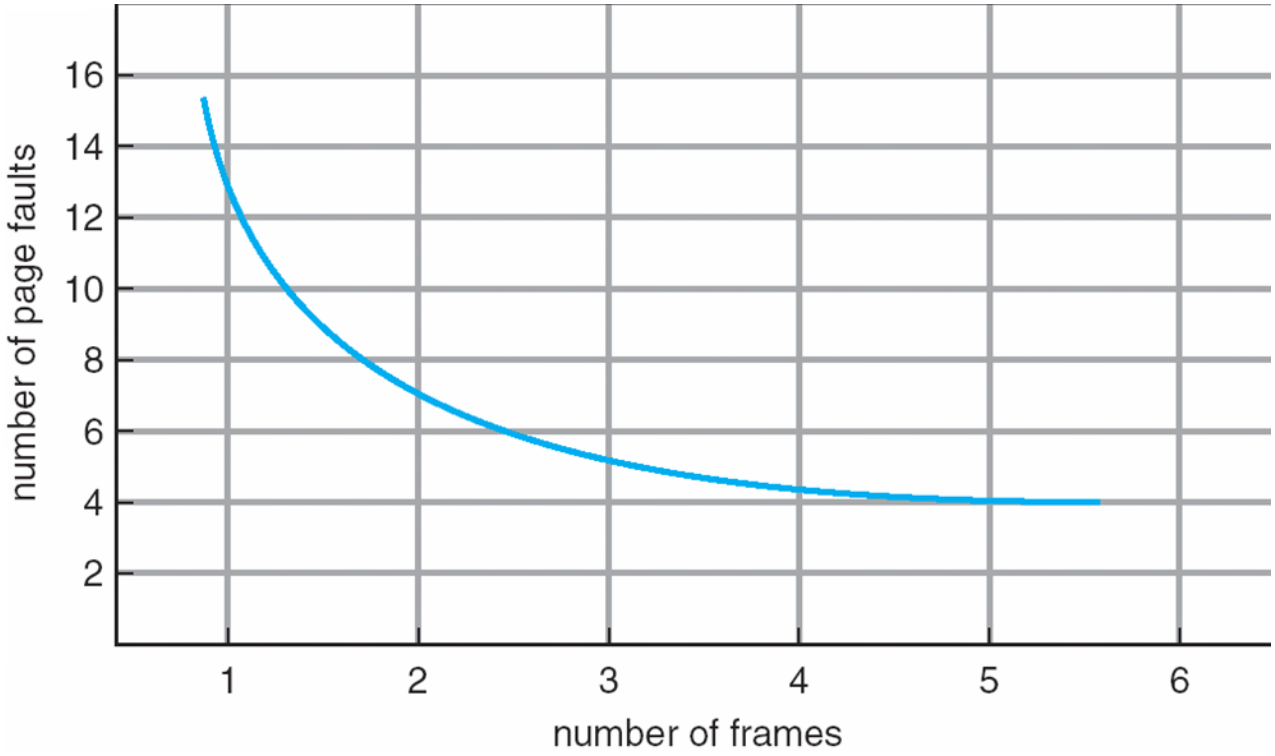
- (3) UNIX方式。
  - 由于与进程有关的文件都放在文件区，故凡是未运行过的页面，都应从文件区调入。
  - 而对于曾经运行过但又被换出的页面，由于是被放在对换区，因此在下次调入时，应从对换区调入。
  - 由于UNIX系统允许页面共享，因此， 某进程所请求的页面有可能已被其它进程调入内存，此时也就无须再从对换区调入。

2、页面分配和替换策略

- 如何为进程分配物理页框?
  - 给进程分配多少个物理块?
    - 进程需要的最少物理块数?
  - 进程的物理块数是固定还是可变?
  - 按什么原则为进程分配物理块数?
- 缺页时，如何替换页面?
  - 替换范围：在什么范围内替换?
  - 替换算法：选择哪一个替换?

物理块数与缺页率

- 为每个进程分配多少物理块?



(1)最小物理块数的确定

- 最小物理块数是指能保证进程正常运行所需的最少物理块数，若小于此值，进程无法运行。
- 一般情况下：
  - 单地址指令且采用直接寻址，则所需最少物理块数为2
  - 若单地址指令且允许采用间接寻址，则所需最少物理块数为3。
  - 某些功能较强的机器，指令及源地址和目标地址均跨两个页面，则最少需要6个物理块。

(2)页面分配策略：固定分配与可变分配

- 固定分配
  - 进程保持页框数固定不变，称固定分配;
  - 进程创建时，根据进程类型和程序员的要求决定页框数，只要有一个缺页中断产生，进程就会有一页被替换。

- 可变分配
  - 进程分得的页框数可变， 称可变分配;
  - 进程执行的某阶段缺页率较高，说明目前局部性较差，系统可增加分配页框以降低缺页率，反之说明进程目前的局部性较好，可减少分配进程的页框数

(3)页面替换策略：局部替换和全局替换

- 全局替换
  - 如果页面替换算法的作用范围是整个系统，称全局页面替换算法，它可以在运行进程间动态地分配页框。
- 局部替换
  - 如果页面替换算法的作用范围局限于本进程，称为局部页面替换算法，它实际上需要为每个进程分配固定的页框。

(4)分配和替换算法的配合

- 固定分配+局部替换
- 可变分配+全局替换
- 可变分配+局部替换

固定分配+局部替换

- 实现要点: 每个进程分得的页框数不变，发生缺页中断，只能从该进程的页面中选页替换，保证进程的页框总数不变。
- 策略难点：应给每个进程分配多少页框?
  - 给少了，缺页中断率高;
  - 给多了，使主存中能同时执行的进程数减少，进而造成处理器和其它设备空闲。
- 采用固定分配算法，系统把页框分配给进程，可采用一些策略，如:
  - 平均分配
  - 按比例分配
  - 优先权分配

容易，但性能差。

可变分配+全局替换

- 实现要点
  - 每个进程分配一定数目页框，OS保留若干空闲页框。
  - 进程发生缺页中断时，从系统空闲页框中选一个给进程，这样产生缺页中断进程的主存空间会逐渐增大，有助于减少系统的缺页中断次数。
  - 系统拥有的空闲页框耗尽时，会从主存中选择一页淘汰，该页可以是主存中任一进程的页面，这样又会使那个进程的页框数减少，缺页中断率上升。

- 策略难点：如何选择哪个页面作为替换？
  - 应用某种淘汰策略选择页面，并未确定哪个进程会失去页面
  - 如果选择某个进程，此进程工作集合的缩小会严重影响其运行，那么这个选择就不是最佳的了。

较为容易实现，目前已用于若干操作系统中。

可变分配+局部替换

- 实现要点
  - 新进程装入主存时，根据应用类型、程序要求，分配给一定数目页框。
  - 产生缺页中断时，从该进程驻留集中选一个页面替换。
  - 不时重新评价进程的分配，增加或减少分配给进程的页框以改善系统性能。

比较复杂，但性能较好。

9.4 页面替换算法

- 页面替换算法又称为页面淘汰算法，是用来选择换出页面的算法。

- 下面介绍几种比较常用的全局页面替换算法
  - 最佳替换算法
  - 先进先出算法
  - 最近最久未使用算法
  - 二次机会算法
  - 简单时钟算法

1．最佳替换算法(Belady 1966)

- Optimal replacement, OPT
- 核心思想：淘汰掉**将来**不再访问，或者距现在最长时间后才可能会访问的页面，
- 换言之是从内存中选择将来最长时间不会使用的页面予以淘汰。
- 缺点：
  - 因页面访问的未来顺序很难精确预测
  - 该算法具有理论意义，可以用来评价其他算法的优劣。

最佳替换算法例

- 假定系统为某进程分配了3个物理块，页面访问序列为：1、2、3、4、1、2、5、1、2、3、4、5，开始时3个物理块均为空闲，采用最佳替换算法时的页面替换情况如下所示：

走向	1	2	3	4	1	2	5	1	2	3	4	5
块1	1	1	1	1			1			3	3	
块2		2	2	2			2			2	4	
块3			3	4			5			5	5	
	缺	缺	缺	缺			缺			缺	缺	

- 共发生了7次缺页，其缺页率为7/12＝58.3% 。

2. 先进先出算法（FIFO）

- 前提假设：
  - 程序是按线性顺序来访问物理空间的
- 核心思想：
  - 选择调入主存时间最长的页面予以淘汰，认为驻留时间最长的页面不再使用的可能性较大。

FIFO例1：无异常现象的页面序列

- 假定系统为某进程分配了3个物理块，页面访问序列为：7、0、1、2、0、3、0、4、2、3、0、3，开始时3个物理块均为空闲，采用先进先出替换算法时的页面替换情况如下所示：

走向	7	0	1	2	0	3	0	4	2	3	0	3
块1	7	7	7	2		2	2	4	4	4	0	
块2		0	0	0		3	3	3	2	2	2	
块3			1	1		1	0	0	0	3	3	
	缺	缺	缺	缺		缺	缺	缺	缺	缺	缺	

- 共发生了10次缺页中断。其缺页率为10/12＝83.3% 。

为进程分配4个物理块

- 从上表中可以看出，共发生了7次缺页中断。其缺页率为7/12 = 58.3% 。

走向	7	0	1	2	0	3	0	4	2	3	0	3
块1	7	7	7	7		3		3			3	
块2		0	0	0		0		4			4	
块3			1	1		1		1			0	
块4				2		2		2			2	
	缺	缺	缺	缺		缺		缺			缺	

FIFO例2：有异常现象的页面序列

- 假定系统为某进程分配了3个物理块，页面访问序列为：1、2、3、4、1、2、5、1、2、3、4、5，开始时3个物理块均为空闲，采用先进先出替换算法时的页面替换情况如下所示：

走向	1	2	3	4	1	2	5	1	2	3	4	5
块1	1	1	1	4	4	4	5			5	5	
块2		2	2	2	1	1	1			3	3	
块3			3	3	3	2	2			2	4	
	缺	缺	缺	缺	缺	缺	缺			缺	缺	

- 共发生了9次缺页中断。其缺页率为9/12＝75% 。

为进程分配4个物理块

- 共发生了10次缺页中断。其缺页率为10/12 = 83.3% 。
- 分配的物理页框增加了，缺页率反而升高了！

走向	1	2	3	4	1	2	5	1	2	3	4	5
块1	1	1	1	1			5	5	5	5	4	4
块2		2	2	2			2	1	1	1	1	5
块3			3	3			3	3	2	2	2	2
块4				4			4	4	4	3	3	3
	缺	缺	缺	缺			缺	缺	缺	缺	缺	缺

FIFO算法特点

- FIFO算法特点：
  - 实现比较简单
  - 对按线性顺序访问的程序比较合适，对其他特性的程序则效率不高，Why?
  - 但可能产生异常现象：
    - Belady 现象：在某些情况下，分配给进程的页面数增多，缺页次数反而增加。
    - 原因：FIFO算法的置换特征与进程访问内存的动态特征是矛盾的，即被置换的页面并不是进程不会访问的，因而FIFO并不是一个好的置换算法。
    - Belady现象与抖动现象是不完全相同的。
  - 很少使用纯粹的FIFO

FIFO的改进：页面缓冲算法

- 页面缓冲替换算法采用FIFO选择被替换页面，选择出的页面不是立即换出，而是放入两个链表之一。
- 算法采用FIFO选择淘汰页，但不立即把该页内容抛弃掉，而是按照修改与否，放到相应队列末尾。
  - 空闲队列：页面未修改则放入空闲队列末尾，该链表也是可直接装入页面的页框所构成
  - 修改队列：页面已修改则放入修改队列末尾
- 需要装入的页面被读进空闲队列队首页框中，而不用等待淘汰页写回再装入。
- 当修改页面到一定数量，就成批写回，并把所占用页框挂到空闲链上。

3.最近最久未使用替换算法(LRU)

- Least Recently Used:
  - 基于局部性原理：刚被使用过的页面可能还会立即被使用，较长时间内未被使用的页面可能不会立即使用。
  - 核心思想：选择最近一段时间内最长时间未被访问过的页面予以淘汰。

- 应赋予每个页面一个访问字段，用于记录页面自上次访问以来所经历的时间，同时维护一个淘汰队列

LRU算法例

- 假定系统为某进程分配了3个物理块，页面访问序列为：1、2、3、4、1、2、5、1、2、3、4、5，开始时3个物理块均为空闲，采用LRU替换算法时的页面替换情况如下所示：

走向	1	2	3	4	1	2	5	1	2	3	4	5
块1	1	1	1	4	4	4	5			3	3	3
块2		2	2	2	1	1	1			1	4	4
块3			3	3	3	2	2			2	2	5
	缺	缺	缺	缺	缺	缺	缺			缺	缺	缺

- 从上表中可以看出，共发生了10次缺页，其缺页率为10/12＝83.3%。

为进程分配4个物理块

- 共发生了8次缺页中断。其缺页率为8/12＝66.7%。

走向	1	2	3	4	1	2	5	1	2	3	4	5
块1	1	1	1	1			1			1	1	5
块2		2	2	2			2			2	2	2
块3			3	3			5			5	4	4
块4				4			4			3	3	3
	缺	缺	缺	缺			缺			缺	缺	缺

LRU算法的实现

- LRU算法实现时需要较多的硬件支持，以记录进程中各页面自上次访问以来有多长时间未被访问，主要实现方法有：
  ① 基于计数器的方法
  ② 基于栈的方法

①基于计数器的方法

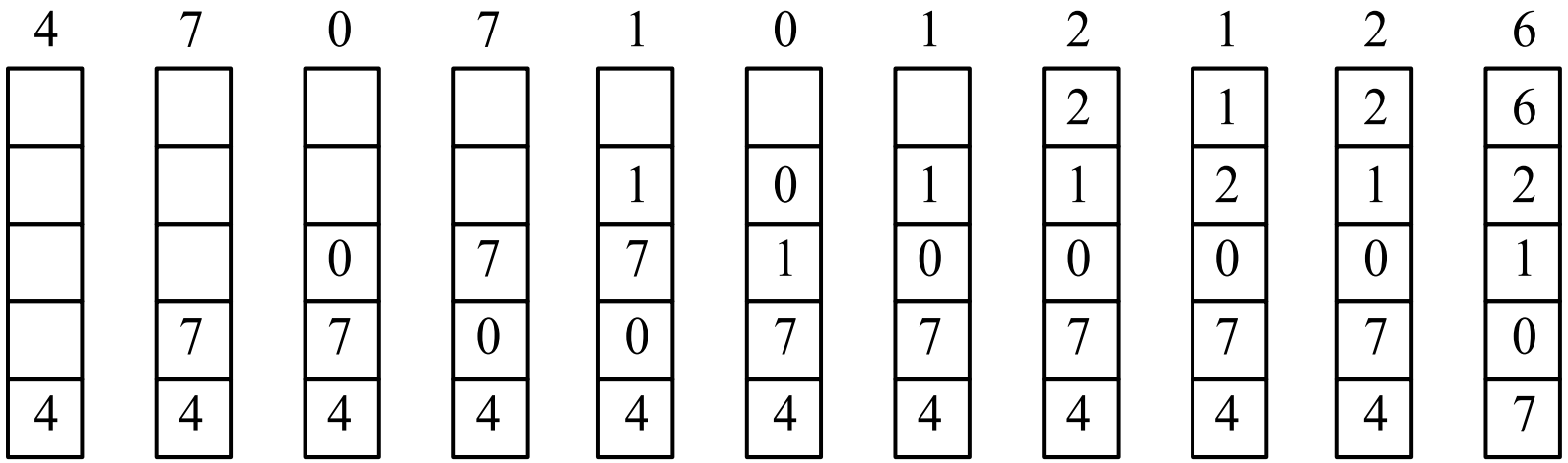
- 为每个页表项关联一个时间域字段
- 为CPU增加一个计数器或者逻辑时钟，每次时钟中断，计数器加1
- 每当访问一页时，将计数器值复制到相应页所对应页表项的时间域内。
- 当发生缺页中断时，可选择时间域数值最小的对应页面淘汰。

②基于栈的方法

- 用一个特殊的栈保存当前进程所访问的各页面号。每当进程访问某页面，便将它对应的页面号从栈中移出，压入栈顶。
- 那么栈顶始终是最近访问的页面，而栈底则是最近最久未使用的页面。

②基于栈的方法（续）

- 例如：设分配给某进程5个物理块，页面访问序列为：4,7,0,7,1,0,1,2,1,2,6，则栈的变化情况如下：



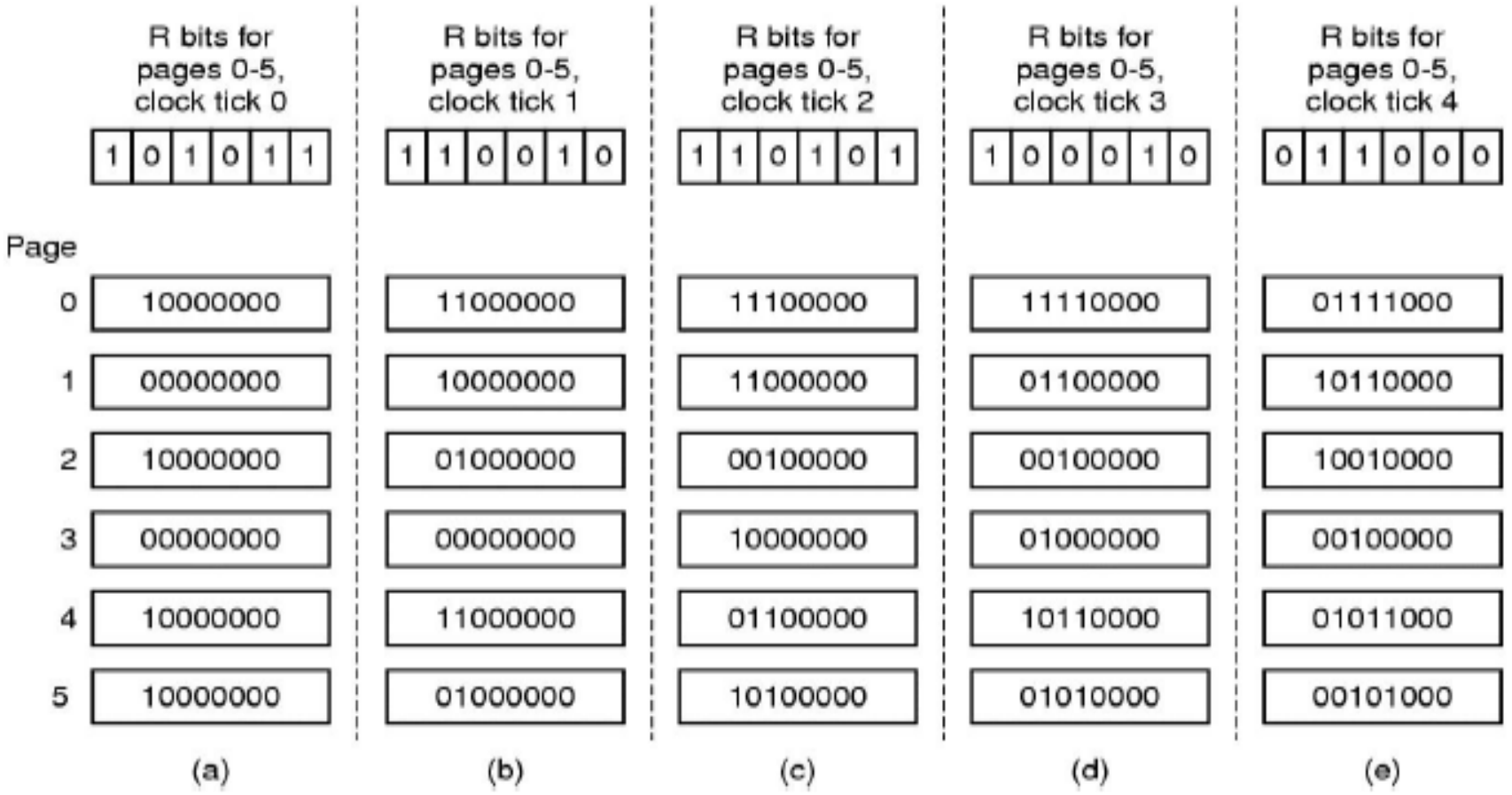
4. LRU算法近似算法

- LRU算法需要足够的硬件支持，且仍然很慢

- 变通：设置引用位
  - 每一个页面关联一个bit，初始为0
  - 当页面被引用时，设置为1
  - 虽然不了解访问顺序，但是了解哪些没被访问过
  - 是许多近似LRU算法的基础

（1）附加引用位法

- 每页都有引用位，并为每页设一个8位内存信息
- 每隔规定时间（如100ms），时钟定时器触发中断，将控制权交给OS
- OS将每个页的引用位转移到8位字节的高位，并将其他位右移1位，抛弃最低位
- 这8位就表明了最近8个时间周期，页面的使用情况
- 发生缺页时，挑选最小的为LRU页替换
- 在这里，最高位是引用位，其他位为历史位
- 也称为老化（Aging）算法



页面3对应的值最小，当发生缺页，首先将它置换出去

(2) 二次机会算法

- 二次机会算法(Second Chance Replacement, SCR)的基本算法是FIFO算法，为避免将经常使用的页面淘汰掉，参考了页面访问位。

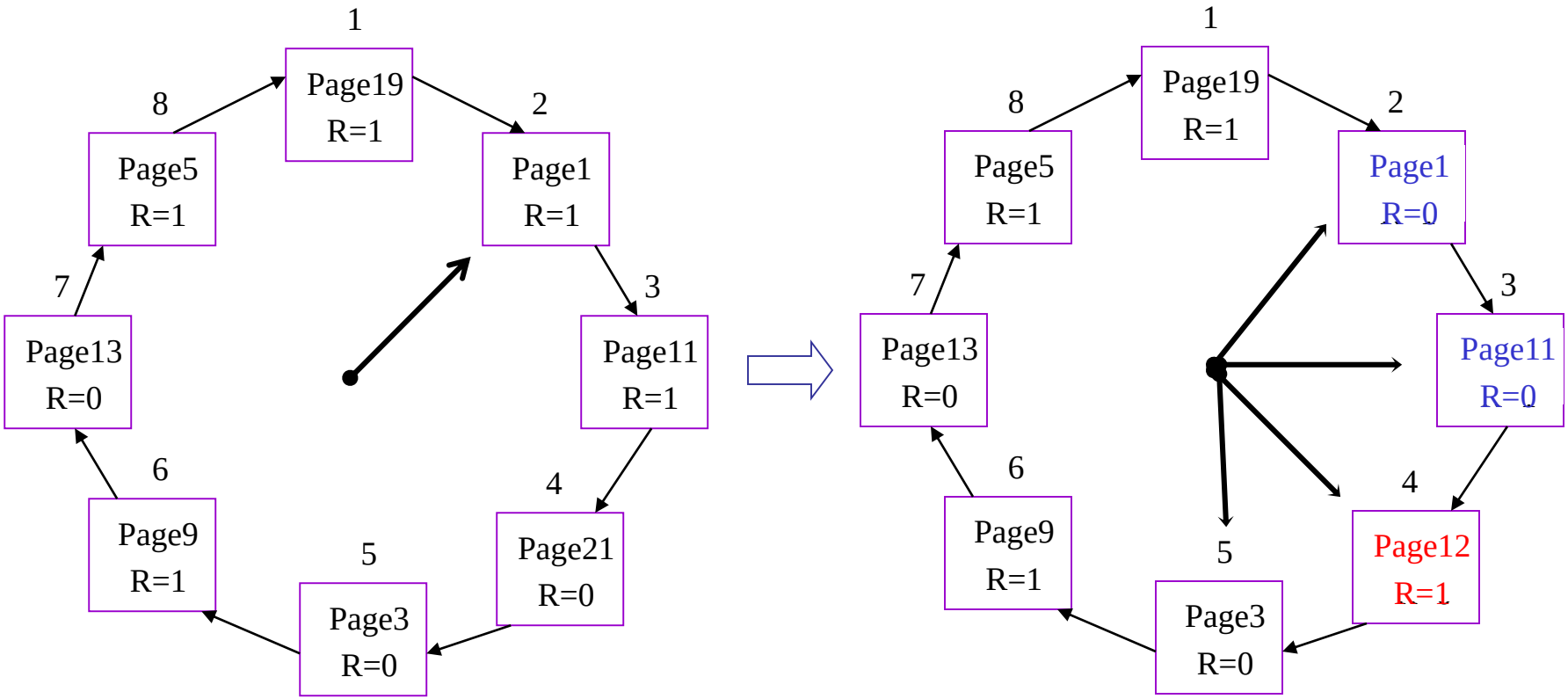
- 算法思想：使用FIFO算法选择一页淘汰时，先检查该页的访问位
  - 如果是0就立即淘汰该页
  - 如果是1就给它第二次机会，将其访问位清0，并将它放入页面链的末尾，将其装入时间置为当前时间，然后选择下一个页面。

（3）简单时钟（clock）算法

- 简单时钟替换算法是对二次机会算法的改进。该算法也要求为每页设置一个访问位。

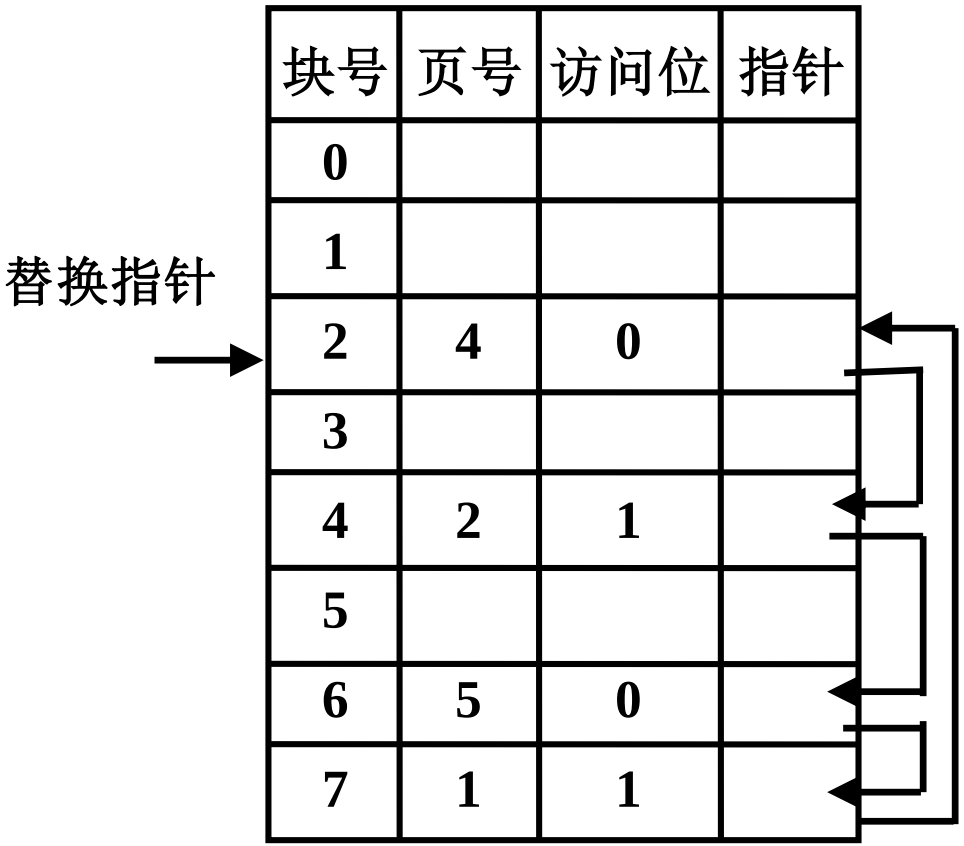
- 实现思想：将页面排成一个循环队列，类似于时钟表面，并使用一个替换指针。
  - 当发生缺页时，检查指针指向的页面；
  - 若其访问位为0则淘汰该页；
  - 否则将该页的访问位清0，指针前移并重复上述过程，直到找到访问位为0的淘汰页为止；
  - 最后指针停留在被替换页的下一页上。

简单时钟置换算法示意图

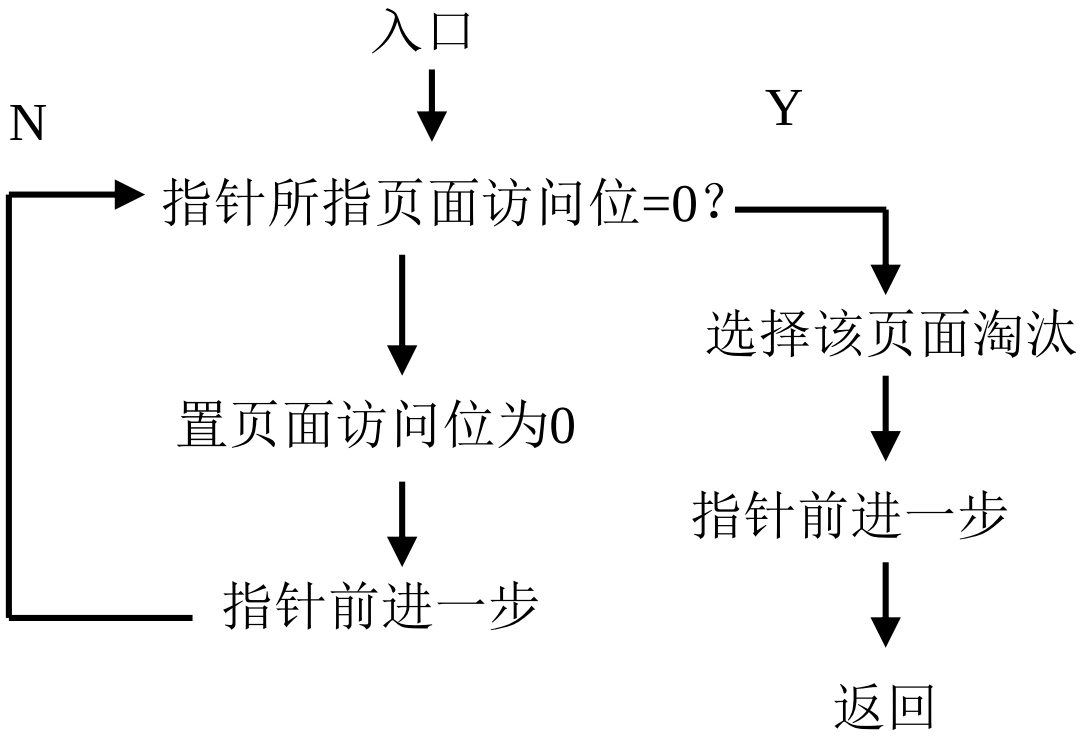


如果此时要求访问page12，应选择哪一页淘汰？替换后状态如何？

简单时钟替换算法页面链



简单时钟替换算法流程



简单时钟替换例

- * 标志表示访问位为1
- 从上表中可以看出，共发生了9次缺页中断。其缺页率为9/12＝75%。

此时为何块2、块3均无*标记？

走向	7	0	1	2	0	3	0	4	2	3	0	3
块1	7*	7*	7*	2*	2*	2*	2*	4*	4*	4*	4	3*
块2		0*	0*	0	0*	0	0*	0	2*	2*	2	2
块3			1*	1	1	3*	3*	3	3	3*	0*	0*
	缺	缺	缺	缺		缺		缺	缺		缺	缺

说明

- 在这里要注意，简单时钟，实际上是一个FIFO的改进，即二次机会的另外一种实现形式，因此，对于那些已经进入引用的页面，他们不影响当前指针的情况，即不是最近使用过的就需要调整。

- 算法的极端情况：
  - 所有页都被引用，则选择时，需要先把所有页面遍历，清除所有引用位，则退化为FIFO

（4）改进的时钟算法

- 问题：将一个修改过的页面换出需要写磁盘，其开销大于未修改页面，为此在改进型时钟算法中应考虑页面修改情况，将访问情况与修改情况结合使用。

- 设R为访问位/引用位，M为修改位，将页面分为以下4种类型：
  - 1类(R=0，M=0)：未被访问又未被修改
  - 2类(R=0，M=1)：未被访问但已被修改
  - 3类(R=1，M=0)：已被访问但未被修改
  - 4类(R=1，M=1)：已被访问且已被修改

改进型时钟算法描述

- 步骤1：从指针当前位置开始扫描循环队列，寻找R=0，M=0的页面，将满足条件的第一个页面作为淘汰页，本轮扫描不修改“访问位R”。（若失败，则R=0,M=1; R=1, M=0/1）

- 步骤2：若第1步失败，则开始第2轮扫描，寻找R=0，M=1的页面，将满足条件的第一个页面作为淘汰页，并将所有经历过页面的访问位R置0。（若失败，R=1,M=0/1）

- 步骤3：若第2步失败，则将指针返回到开始位置，然后重复第1步，若仍失败则必须重复第2步，此时一定能找到淘汰页面。

- 特点：
  - 被称为“第三次机会时钟替换算法”。
  - 减少了磁盘I/O次数，但算法本身开销增加。

5. 工作集算法

（1）工作集模型

- 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)

（1）工作集模型 （续）

- 工作集：一个进程当前正在使用的页框集合
- 工作集W（t，Δ）= 该进程在过去的Δ个虚拟时间单位(Δ个页面引用)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

（1）工作集模型 （续）

例：

2615777516234123444343444 1327

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

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

(2) 工作集算法

- 基本思路：找出一个不在工作集中的页面并置换它

工作集算法例，Δ=4，W0={5,4,1}

时间	0	1	2	3	4	5	6	7	8	9	10
引用序列	1	3	3	4	2	3	5	3	5	1	4
1	√	√	√	√						√	√
2					√	√	√	√			
3		√	√	√	√	√	√	√	√	√	√
4	√	√	√	√	√	√	√				√
5	√	√					√	√	√	√	√
换入		3			2		5			1	4
换出			5		1			4	2		
工作集合	5,4,1	5,4,1,3	4,1,3	1,3,4	3,4,2	3,4,2	2，3，4，5	2，3，5	3，5	1，3，5	1，3，4，5

缺页数：5，缺页率50%

工作集算法的实现

- 实现思路：
  - 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

- 工作集策略在概念上好，但监督驻留页面变化的开销很大，估算合适的窗口Δ大小也是个难题

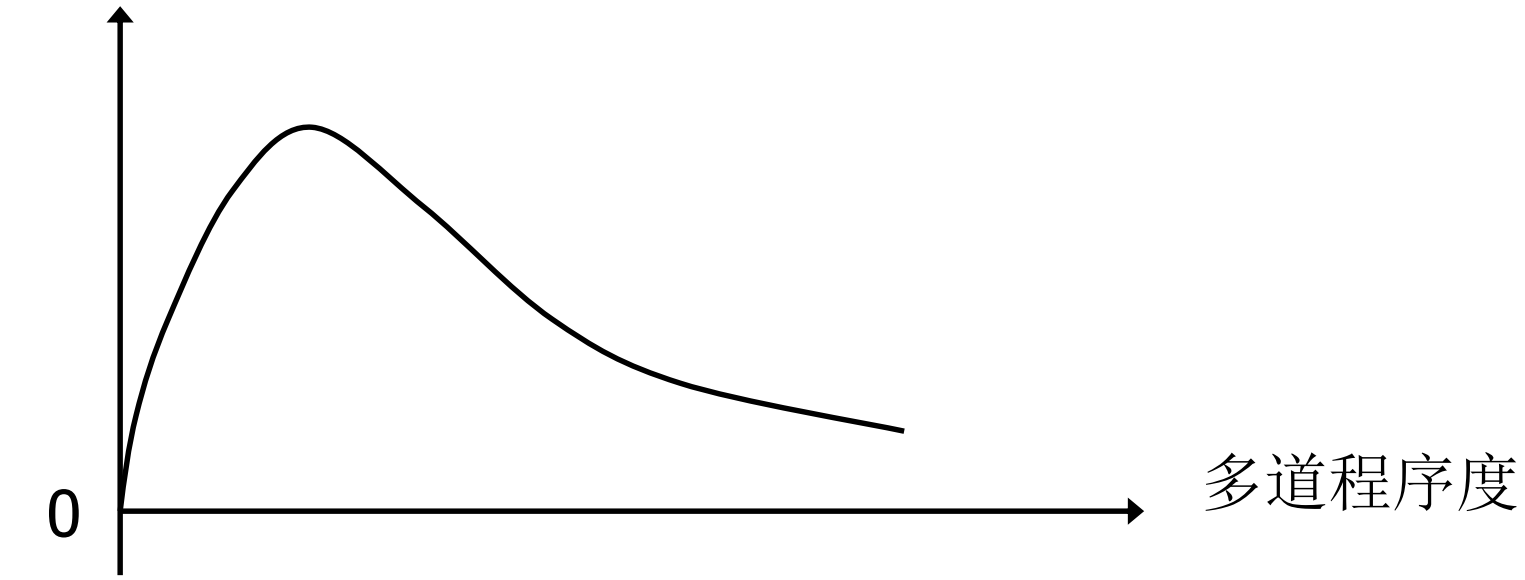
9.5 抖动问题

- 由于频繁缺页，导致运行进程的大部分时间都用于页面的换入/换出，而几乎不能完成任何有效的工作，则称此进程处于抖动状态。抖动又称为颠簸、颤动。
- 抖动分为：
  - 局部抖动
  - 全局抖动

CPU利用率与多道程序度的关系

- 左图是CPU利用率与程序道数的关系。

- 开始时CPU利用率会随着程序道数的增加而增加，当程序道数增加到一定数量以后，程序道数的增加会使CPU的利用率急剧下降。



抖动产生的原因

- 抖动产生的原因有：
  ① 进程分配的物理块太少

  ② 替换算法选择不当

  ③ 全局替换使抖动传播

抖动的预防及解除

- 采用局部替换策略可以防止抖动传播
- 通过挂起进程来解除抖动
- 选择挂起进程的条件
  - 优先级最低：符合进程调度原则
  - 发生缺页中断的进程：内存不含工作集，缺页时应阻塞
  - 最后被激活的进程：工作集可能不在内存
  - 最大的进程：可释放较多空间

9.6 影响缺页率的因素

影响进程缺页率的因素

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

程序结构对缺页率影响例

- 设页面大小为128字节，二维数组为128×128，初始时未装入数据，需将数组初始化为0。**若数组按行存放**，问下述两个程序段的缺页率各为多少？
- 程序1

```
short int a[128][128];
for (j=0;j<=127;j++)
for (i=0;i<=127;i++)
    a[i][j]=0;
```

程序2

```
 short int a[128][128];
for (i=0;i<=127;i++)
for (j=0;j<=127;j++)
   a[i][j]=0;
```

程序1的缺页次数

- 因数组以行为主存放，页面大小为128字节，故每行占一个页面。
- 程序1的内层循环将每行中的指定列置为0，故产生128次中断。
- 外层循环128次，总缺页次数为128×128。

程序1

```
short int a[128][128];
for (j=0;j<=127;j++)
  for (i=0;i<=127;i++)
    a[i][j]=0;
```

程序2的缺页次数

- 因数组以行为主存放，页面大小为128字节，故每行占一个页面。
- 程序2的内层循环将每行的所有列置为0，故产生1次中断。
- 外层循环128次，总缺页次数为128。

程序2

```
short int a[128][128];
for (i=0;i<=127;i++)
  for (j=0;j<=127;j++)
    a[i][j]=0;
```

- 程序结构对缺页率的影响：好的程序结构可以降低缺页率。

页面大小的选择

- 页面大小的选择涉及诸多因素，需要在这些因素之间权衡利弊。
  - 页表大小：页面越小，页表越长。依此应选择大页面。
  - 页内碎片问题：页面越大，碎片越大。依此应选择小页面。
  - 内外存传输：大小页面传输时间基本相同，依此应选择大页面。但小页面可以减少总的I/O。
  - 页面大小与主存关系：主存大则页面大。
  - 页面大小对缺页率的影响：大页面可以降低缺页率。
  - 页面大小与快表命中率的关系：页面小快表命中率低。

页面大小选择的分析

- 设系统内每个进程的平均长度为s，页面大小为p，每个页表项需e个字节，内存大小为m。
  - 则内存中的进程数为：m/s
  - 内存中页面数为：m/p
  - 页表项占用空间：me/p
  - 每进程碎片的平均大小：p/2
  - 总碎片空间为：(p/2)×(m/s) = pm/2s
  - 分页系统总开销：pm/2s + me/p

页面大小选择的分析（续）

- 分页系统总开销：pm/2s + me/p
- 则内存的浪费率为：

$$f(p) = (pm/2s+me/p)/m$$
$$= p/2s+e/p$$

- 对p求导数：$f'(p)=1/2s-e/p^2$
- 令导数为0：$1/2s-e/p^2=0$，得到$p=\sqrt{2es}$
- 则页面大小为：$p=\sqrt{2es}$时，f取得最小值 $(\sqrt{2/s})$。

课堂练习

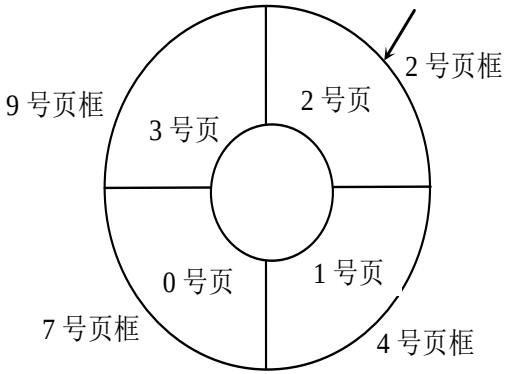
- 使用OPT、FIFO、LRU、简单时钟四种方法，比较如下执行序列的性能。假设采用固定分配策略，进程可分页框总数为3个，执行中所引用到的页面总数为5个，执行序列为:

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

小作业 - 第1题

- 设某计算机的逻辑地址空间和物理地址空间均为64KB，按字节编址。若某进程最多需要6页数据存储空间，页的大小为1KB。操作系统采用固定分配局部置换策略为此进程分配4个页框（Page frame），如下表所示。

页号	页框号	装入时刻	访问位
0	7	130	1
1	4	230	1
2	2	200	1
3	9	160	1



当该进程执行到260时刻时，要访问逻辑地址为17CAH的数据，请回答下列问题：

（1）该逻辑地址对应的页号是多少？

（2）若采用先进先出（FIFO）置换算法，该逻辑地址对应的物理地址是多少？要求给出计算过程。

（3）若采用时钟（CLOCK）置换算法，该逻辑地址对应的物理地址是多少？要求给出计算过程。（设搜索下一页的指针沿顺时针方向移动，且当前指向2号页框，如图所示）。

小作业 - 第2题

- 请求分页管理系统中，假设某进程的页表内容如下表所示：

页号	页框号	有效位
0	101H	1
1	-	0
2	254H	1

页面大小为4KB，一次内存的访问时间是100ns，一次快表（TLB）的访问时间是10ns，处理一次缺页的平均时间是10^8ns（已含更新TLB表和页表的时间），进程的驻留集大小固定为2，采用最近最久未使用替换算法（LRU）和局部淘汰策略。假设：

（1）TLB初始为空

（2）地址转换时先访问TLB，若TLB未命中，再访问页表（忽略访问页表之后的TLB更新时间）

（3）有效位为0表示页面不在内存，产生缺页中断，缺页中断处理后，返回到产生缺页中断的指令处重新执行。设有虚地址访问序列2362H，1565H，25A5H，请问：

（1）依次访问上述三个虚地址，各需多少时间？给出计算过程。

（2）基于上述访问序列，虚地址1565H的物理地址是多少？请说明理由

小作业 第3题

- 某请求分页系统的页面置换策略如下：从0时刻开始扫描，每隔5个时间单位扫描一轮驻留集（扫描时间忽略不计）且在本轮没有被访问过的页框将被系统回收，并放入到空闲页框链尾，其中内容暂时不清空，当发生缺页时，如果该页曾被使用过且还在空闲页框链表中，则将其重新放回进程的驻留集中；否则，从空闲页框链表头部取出一个页框。
- 忽略其他进程的影响和系统开销。初始时进程驻留集为空。目前系统空闲页的页框号依次为32、15、21、41。进程P依次访问的<虚拟页号,访问时刻>为<1,1>、<3,2>、<0,4>、<0,6>、<1,11>、<0,13>、<2,14>。请回答以下问题：

(1) 当虚拟页为<0,4>时，对应的页框号是什么？

(2) 当虚拟页为<1,11>时，对应的页框号是什么？说明理由。

(3) 当虚拟页为<2,14>时，对应的页框号是什么？说明理由。

(4) 这种方法是否适合于时间局部性好的程序？说明理由。