河南大学

操作系统

计算机学院

张　帆

教授

13839965397

zhangfan@henu.edu.cn

计算机学院 501 房间

第5章

虚拟存储器

存储管理分类

- 实存管理
  - 分区（Partitioning）（连续分配方式）（包括固定分区、可变分区）
  - 分页（Paging）
  - 分段（Segmentation）
  - 段页式（Segmentation with paging）

存储管理分类

- 实存管理
  - 分区（Partitioning）（连续分配方式）（包括固定分区、可变分区）
  - 分页（Paging）
  - 分段（Segmentation）
  - 段页式（Segmentation with paging）
- 虚存管理
  - 请求分页（Demand paging）-- 主流技术
  - 请求分段（Demand segmentation）
  - 请求段页式（Demand SWP ）

存储管理分类

- 实存管理
  - 分区（Partitioning）（连续分配方式）（包括固定分区、可变分区）
  - 分页（Paging）
  - 分段（Segmentation）
  - 段页式（Segmentation with paging）
- 虚存管理
  - 请求分页（Demand paging）-- 主流技术
  - 请求分段（Demand segmentation）
  - 请求段页式（Demand SWP ）

离散分配

常规存储管理的问题

- 常规存储管理方式的共同点：要求一个作业全部装入内存后方能运行。可能出现的问题：
  - 有的作业很大，所需内存空间大于内存总容量，使作业无法运行。
  - 有大量作业要求运行，但内存容量不足以容纳下所有作业，只能让一部分先运行，其它在外存等待。
- 解决方法
  - 增加内存容量
  - 从逻辑上扩充内存容量：覆盖、对换、虚拟存储技术

常规存储器管理方式的特征和局部性原理

常规存储器管理方式的特征

1. 一次性：作业在运行前需一次性地全部装入内存。
2. 驻留性：作业装入内存后便一直驻留内存，直至结束。

局部性原理

1. 程序执行时，除了少部分的转移和过程调用指令外，在大多数情况下仍是顺序执行的。
2. 过程调用将会使程序的执行轨迹由一部分区域转至另一部分区域，但过程调用的深度在大多数不超过 5。
3. 程序中存在许多循环结构，它们将多次执行。
4. 程序中还包括许多对数据结构的处理，如对数组进行操作，它们往往都局限于很小的范围内。

常规存储器管理方式的特征和局部性原理

- 局部性又表现在下述两个方面
  1. 时间局部性。如果程序中的某条指令一旦执行，则不久以后该指令可能再次执行；如果某数据被访问过，则不久以后该数据可能再次被访问。产生时间局限性的典型原因，是由于在程序中存在着大量的循环操作。
  2. 空间局部性。一旦程序访问了某个存储单元，在不久之后，其附近的存储单元也将被访问，即程序在一段时间内所访问的地址，可能集中在一定的范围之内，其典型情况便是程序的顺序执行。

虚拟存储器的基本原理

- 根据局部性原理，在程序装入时，不需要将其全部读入到内存，而只需将当前需要执行的部分页或段读入到内存，就可让程序开始执行。
- 在程序执行过程中，如果需执行的指令或访问的数据尚未在内存（称为缺页或缺段），则由处理器通知操作系统将相应的页或段调入到内存，然后继续执行程序。（请求调入功能）
- 另一方面，操作系统将内存中暂时不使用的页或段调出保存在外存上，从而腾出空间存放将要装入的程序以及将要调入的页或段。（置换功能）

虚拟存储器的定义和特征

- 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加以扩充的一种存储器系统。
- 逻辑容量由内存容量和外存容量之和所决定。
- 运行速度接近于内存速度，成本接近于外存。

虚拟存储器的特征

1. 多次性：作业被分成多次调入内存运行。正是由于多次性，虚拟存储器才具备了逻辑上扩大内存的功能。多次性是虚拟存储器最重要的特征，其它任何存储器不具备这个特征。
2. 对换性：允许在作业运行过程中进行换进、换出。换进、换出可提高内存利用率。
3. 虚拟性：指能够从逻辑上扩充内存容量，使用户所看到的内存容量远大于实际内存容量。虚拟性是实现虚拟存储器最重要的目标。

- 虚拟性以多次性和对换性为基础，而多次性和对换性必须以离散分配为基础。

虚拟存储器的类比

电脑配件销售方法	虚拟存储技术
配件存放于柜台与仓库，是一种存放体系	虚拟存储器由内存和外存共同组成，是一种存储体系
柜台取配件快，仓库取配件慢	内存访问速度快，外存访问速度慢
容量由柜台和仓库容量之和决定	容量由内存和外存容量之和决定
柜台单位容量成本高，仓库则低	内存单位容量价格高，外存则低
配件的销售也存在局部性现象	程序访问的局部性原理
柜台摆放顾客购买频率最高的配件	内存存放经常访问的数据
顾客要购买柜台没有而仓库有的配件，则出现缺货现象	存在缺页（段）现象
缺货时，需从仓库拿配件	缺页（段）时则需要请求调页（段），由缺页（段）中断机构调入
具有柜台和仓库之间货物的置换策略	有置换算法
以人力和租金便宜的仓库来换取租金昂贵的柜台空间	以CPU时间和外存空间换取宝贵的内存空间

虚拟存储技术的概念

- 速度和容量：虚拟存储量的扩大是以牺牲 CPU 工作时间以及内外存交换时间为代价。
- 虚拟存储器的容量取决于主存与辅存的容量，最大容量是由计算机的地址结构决定。
- 虚拟存储器的逻辑地址空间理论上不受物理存储器的限制。

  如 32 位机器，虚拟存储器的最大容量就是 4G，再大 CPU 无法直接访问。
- 虚拟存储器的实现方法：请求分页、请求分段、请求段页式

请求分页系统

- 在分页系统的基础上，增加了请求调页功能、页面置换功能所形成的页式虚拟存储器系统。
- 它允许只装入若干页的用户程序和数据，便可启动运行，以后在硬件支持下通过调页功能和置换页功能，陆续将要运行的页面调入内存，同时把暂不运行的页面换到外存上，置换时以页面为单位。
- 系统须设置相应的硬件支持和软件：
  1. 硬件支持：请求分页的页表机制、缺页中断机构和地址变换机构。
  2. 软件：请求调页功能和页置换功能的软件。

请求分段系统

- 在分段系统的基础上，增加了请求调段功能及分段置换功能，所形成的段式虚拟存储器系统。
- 它允许只装入若干段的用户程序和数据，便可启动运行，以后再硬件支持下通过请求调段功能和分段置换功能，陆续将要运行的段调入内存，同时把暂不运行的段换到外存上，置换时以段为单位。
- 系统须设置相应的硬件支持和软件：
  1. 硬件支持：请求分段的段表机制、缺段中断机构和地址变换机构。
  2. 软件：请求调段功能和段置换功能的软件。

虚拟存储器的实现方法

	请求分页系统	请求分段系统
基本单位	页	段
长度	固定	可变
分配方式	固定分配	可变分配
复杂性	较简单	较复杂

请求页表机制

页号	块号	状态位P	访问字段A	修改位M	外存地址

请求页表机制

页号	块号	状态位P	访问字段A	修改位M	外存地址

1. 状态位 P：指示该页是否已调入内存。

   供程序访问时参考

请求页表机制

页号	块号	状态位P	访问字段A	修改位M	外存地址

1. 状态位 P：指示该页是否已调入内存。
   供程序访问时参考
2. 访问字段 A：记录本页在一段时间内被访问的次数或最近未被访问的时间。供选择页面换出时参考

请求页表机制

页号	块号	状态位P	访问字段A	修改位M	外存地址

1. 状态位 P：指示该页是否已调入内存。
   供程序访问时参考
2. 访问字段 A：记录本页在一段时间内被访问的次数或最近未被访问的时间。供选择页面换出时参考
3. 修改位 M：表示该页在调入内存后是否被修改过。若修改过，则置换该页时需重写该页至外存。
   供置换页面时参考

请求页表机制

页号	块号	状态位P	访问字段A	修改位M	外存地址

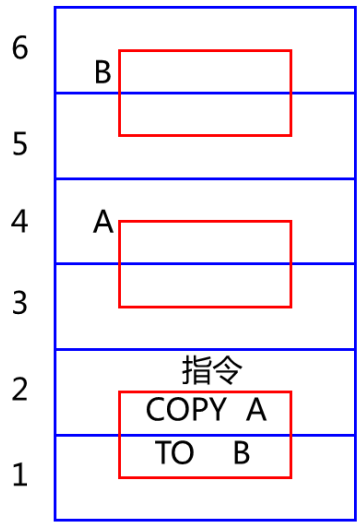
1. 状态位 P：指示该页是否已调入内存。
   供程序访问时参考
2. 访问字段 A：记录本页在一段时间内被访问的次数或最近未被访问的时间。供选择页面换出时参考
3. 修改位 M：表示该页在调入内存后是否被修改过。若修改过，则置换该页时需重写该页至外存。
   供置换页面时参考
4. 外存地址：指出该页在外存上的地址。
   供调入该页时参考

缺页中断机构

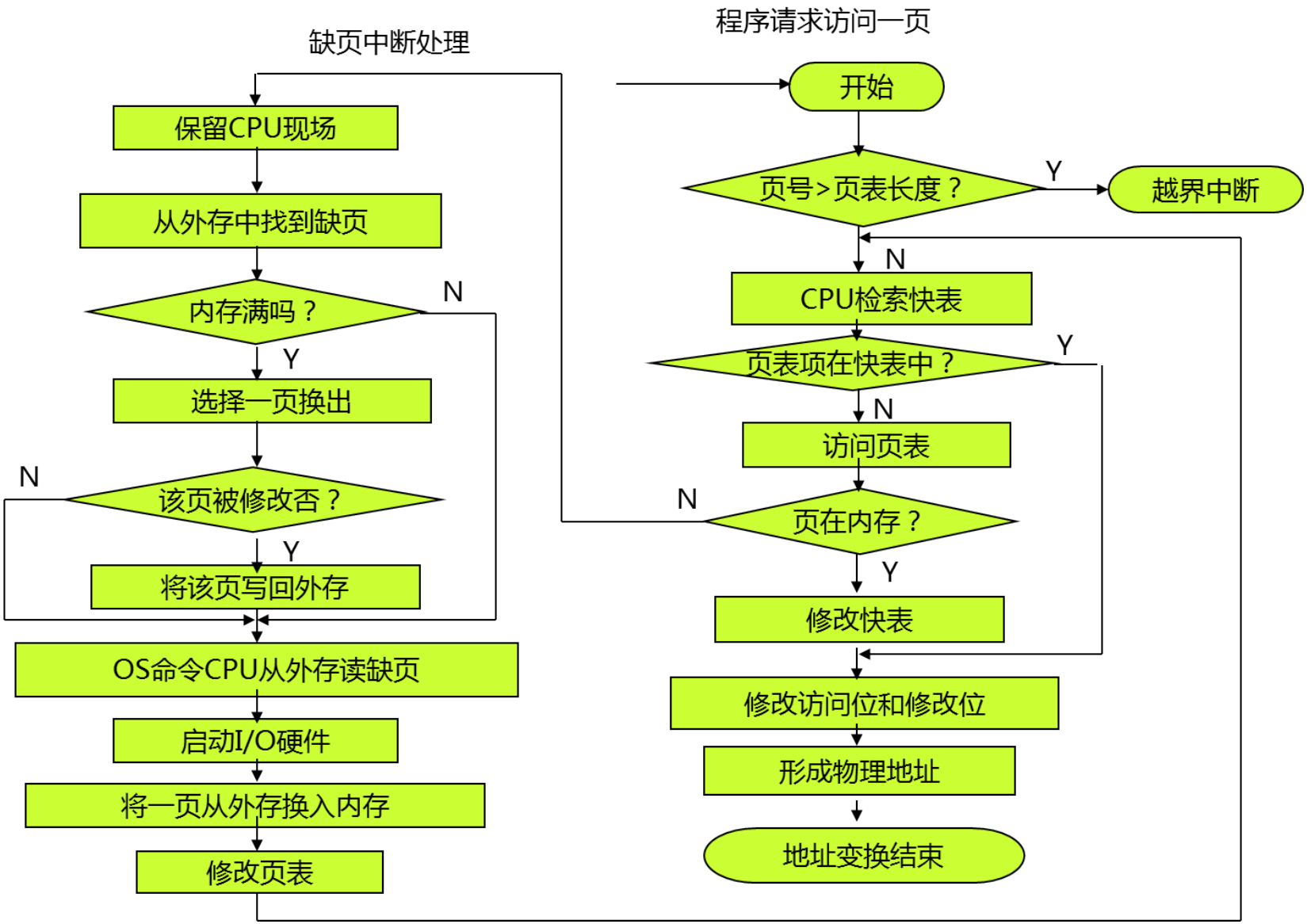
- 在请求分页系统中，当访问的页不在内存，便产生一个缺页中断。
- 缺页中断与一般中断的区别：
  1. 缺页中断是在指令执行期间产生和处理中断信号（要访问的指令或数据不在内存）。
  2. 一条指令在执行期间，可能产生多次缺页中断。

涉及 6 次缺页中断的指令

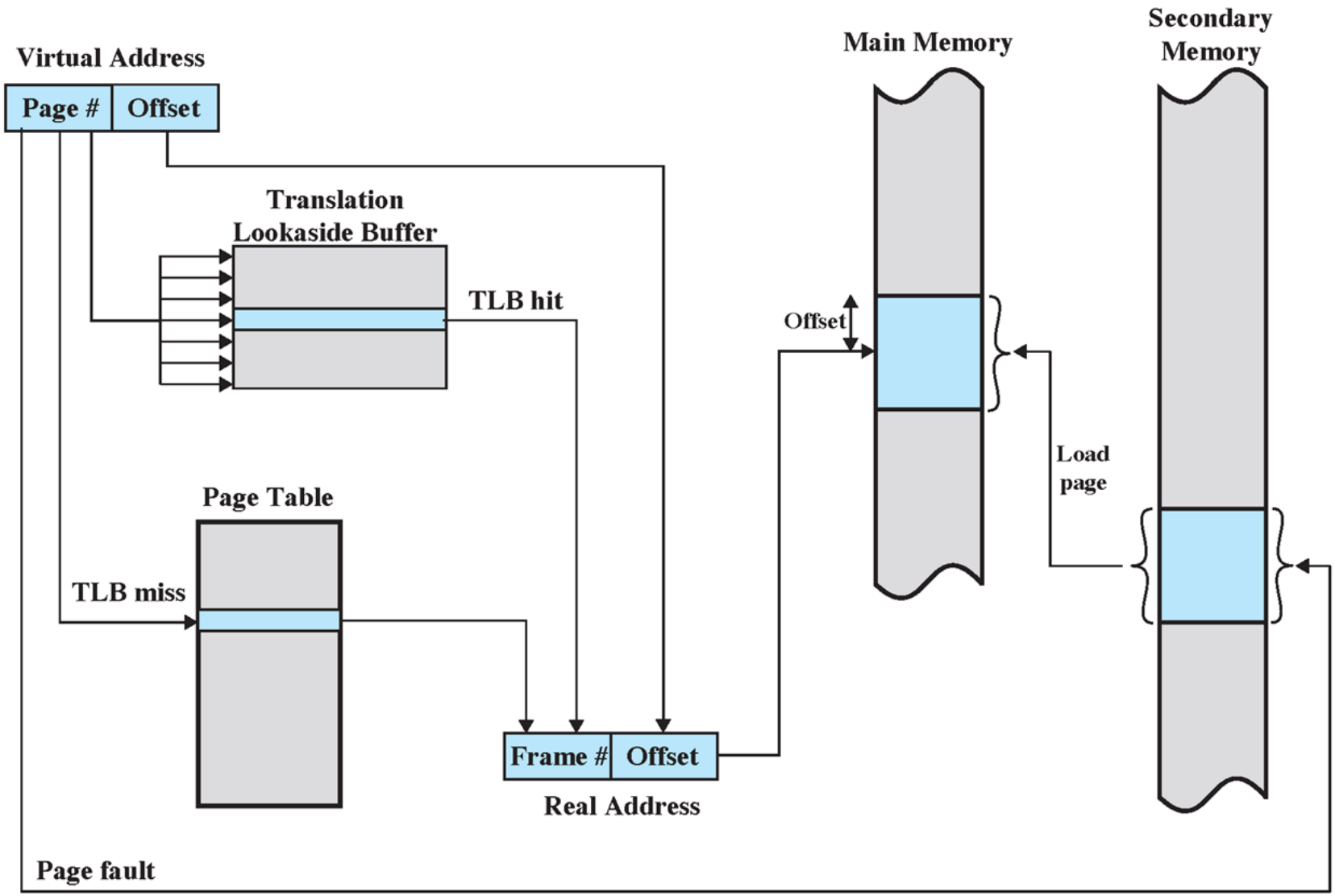
指令本身跨了两个页面（不在内存和跨页各中断一次），A 和 B 又分别跨了两个页面。



地址变换机构


缺页中断处理
保留CPU现场
从外存中找到缺页
内存满吗？
N
Y
选择一页换出
N
该页被修改否？
Y
将该页写回外存
OS命令CPU从外存读缺页
启动I/O硬件
将一页从外存换入内存
修改页表
程序请求访问一页
开始
页号>页表长度？
Y
越界中断
N
CPU检索快表
页表项在快表中？
Y
N
访问页表
N
页在内存？
Y
修改快表
修改访问位和修改位
形成物理地址
地址变换结束

地址变换机构


Virtual Address
Page #
Offset
Translation
Lookaside Buffer
TLB hit
Page Table
TLB miss
Frame #
Offset
Real Address
Page fault
Main Memory
Secondary
Memory
Offset
Load
page

请求分页存储管理

例：一个采用请求分页存储管理的计算机系统，其内存（实存）容量为 256M 字节，虚拟内存容量（给用户的最大地址空间）为 4G 字节，页面大小为 4K 字节，试问：

1. 实存物理地址应设为多少位?

请求分页存储管理

例：一个采用请求分页存储管理的计算机系统，其内存（实存）容量为 256M 字节，虚拟内存容量（给用户的最大地址空间）为 4G 字节，页面大小为 4K 字节，试问：

1. 实存物理地址应设为多少位？ 28 位
2. 实存中有多少物理块？

请求分页存储管理

例：一个采用请求分页存储管理的计算机系统，其内存（实存）容量为 256M 字节，虚拟内存容量（给用户的最大地址空间）为 4G 字节，页面大小为 4K 字节，试问：

1. 实存物理地址应设为多少位？ 28 位
2. 实存中有多少物理块？ 64K
3. 实存中最大块号是多少？

请求分页存储管理

例：一个采用请求分页存储管理的计算机系统，其内存（实存）容量为 256M 字节，虚拟内存容量（给用户的最大地址空间）为 4G 字节，页面大小为 4K 字节，试问：

1. 实存物理地址应设为多少位？ 28 位
2. 实存中有多少物理块？ 64K
3. 实存中最大块号是多少？ 64K-1
4. 虚存地址应设多少位？

请求分页存储管理

例：一个采用请求分页存储管理的计算机系统，其内存（实存）容量为 256M 字节，虚拟内存容量（给用户的最大地址空间）为 4G 字节，页面大小为 4K 字节，试问：

1. 实存物理地址应设为多少位？ 28 位
2. 实存中有多少物理块？ 64K
3. 实存中最大块号是多少？ 64K-1
4. 虚存地址应设多少位？ 32 位
5. 虚拟地址空间最多可以有多少页？

请求分页存储管理

例：一个采用请求分页存储管理的计算机系统，其内存（实存）容量为 256M 字节，虚拟内存容量（给用户的最大地址空间）为 4G 字节，页面大小为 4K 字节，试问：

1. 实存物理地址应设为多少位？ 28 位
2. 实存中有多少物理块？ 64K
3. 实存中最大块号是多少？ 64K-1
4. 虚存地址应设多少位？ 32 位
5. 虚拟地址空间最多可以有多少页？ 1M

请求分页中的内存分配

请求分页中的内存分配

1. 最小物理块数的确定
2. 物理块的分配策略
3. 物理块分配算法

最小物理块数的确定

- 最小物理块数指能保证进程正常运行所需的最少的物理块数，最小物理块数与计算机的硬件结构有关，取决于指令的格式、功能和寻址方式。
- 采用直接寻址方式，所需的最少物理块数为 2。一块是用于存放指令，另一块用于存放数据。
- 间接寻址时，至少要求有三个物理块。(间接寻址中一些物理块放的是其它物理块的块号)

物理块的分配策略

物理块的分配策略

1. 固定分配局部置换
2. 可变分配全局置换
3. 可变分配局部置换

物理块的分配策略

1 固定分配局部置换

- 为每个进程分配固定数目 n 的物理块，在整个运行中都不改变。如出现缺页则从该进程的页面中置换一页。
- 每个进程分配多少个物理块难以确定。
- 若太少，会频繁地出现缺页中断，降低了系统的吞吐量。
- 若太多，内存中驻留的进程数目减少，可能造成 CPU 空闲或其它资源空闲的情况。

物理块的分配策略

2. 可变分配全局置换：为每个进程分配一定数目的物理块，但 OS 自留一空闲块队列，若发现缺页，则从空闲块队列中分配一空闲块与该进程，并调入缺页于其中。当空闲块队列用完时，OS 才从内存中任选择一页置换。
3. 可变分配局部置换：为每个进程分配一定数目的物理块，若发现缺页，则从该进程的页面中置换一页，不会影响其它进程的运行。根据进程缺页率高低，则可增加或减少分配给该进程的物理块。

物理块分配算法

在采用固定分配策略时，可采用以下几种算法：

1. 平均分配算法：平均分配给各个进程。未考虑进程大小，小进程浪费物理块，大进程严重缺页。
2. 按比例分配算法：根据进程的大小按比例分配给各个进程。如果共有 n 个进程，每进程页面数 S_i，系统可用物理块总数为 m，则每进程分到的物理块数 b_i：

$$b_i = m \times S_i / \sum_{i=1}^{n} S_i$$

3. 考虑优先权的分配算法：将系统提供的物理块一部分根据进程大小先按比例分配给各个进程，另一部分再根据各进程的优先权分配物理块数。

页面调入策略

页面调入策略

1. 何时调入页面
2. 从何处调入页面
3. 页面调入过程
4. 缺页率

何时调入页面

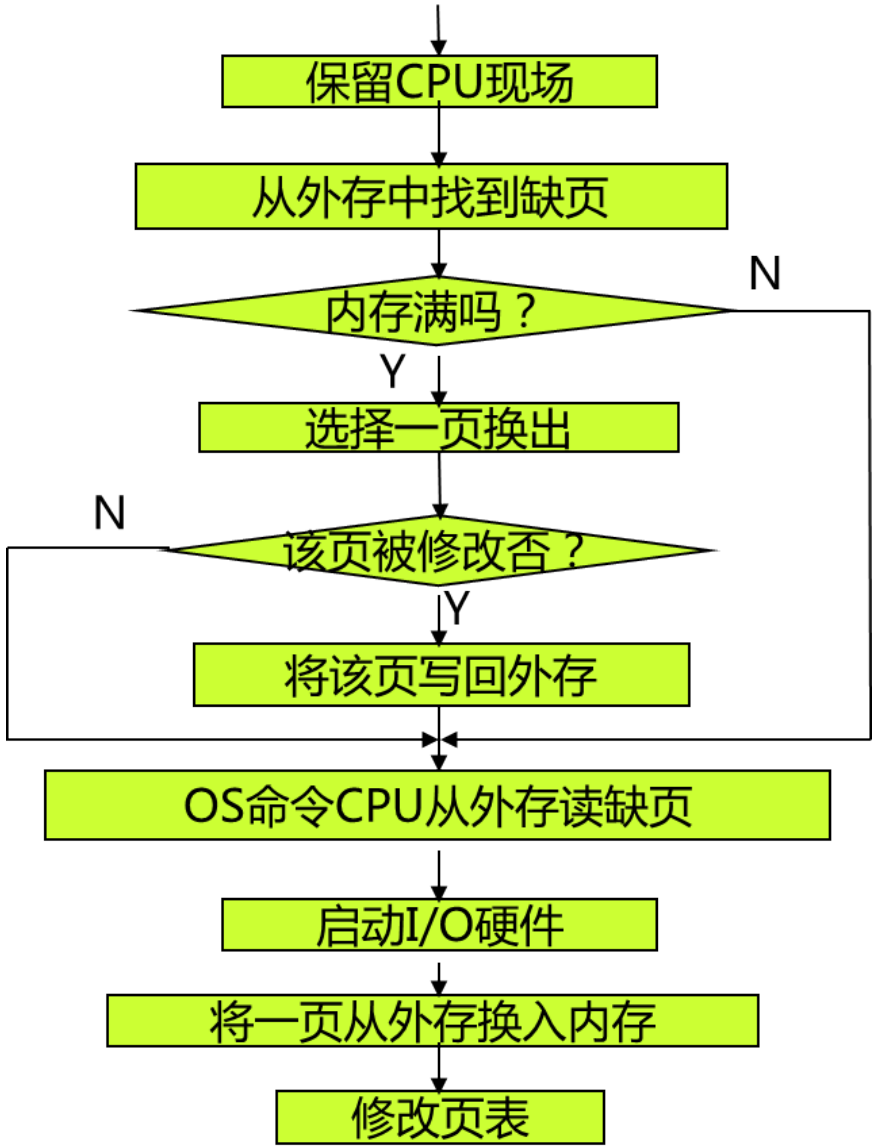
- 预调页策略
  - 预调页：将预计在不久之后便会被访问的页面预先调入内存。
  - 进程的页一般存放在外存的一个连续区域中。一次调入若干个相邻的页会比一次调入一页更高效。
  - 但如果调入的一批页面中的大多数都未被访问，则浪费了内存。
- 请求调页策略
  - 当进程在运行中发生缺页时，就立即提出请求，由系统将缺页调入内存。但这种策略每次仅调入一页，须花费较大的系统开销，增加了启动磁盘 I/O 的频率。

从何处调入页面

在请求分页系统中，外存分成了按离散分配方式存放文件的文件区和按连续分配方式存放对换页的对换区。进程发出缺页请求时，从何处将缺页调入内存呢？

- 对换区：如果系统有足够的对换区空间，运行前可将与进程相关的文件从文件区复制至对换区，以后缺页时全部从对换区调页。
- 文件区：如果系统没有足够的对换区空间，凡是不会被修改的文件，直接从文件区调页，不必回写（换出）。对可能会修改的文件第一次直接从文件区调页，换出时换至对换区，以后从对换区调页。
- UNIX 方式：凡未运行过的页面均从文件区调页，运行过的页面和换出的页面均从对换区调页。

页面调入过程


保留CPU现场
从外存中找到缺页
内存满吗？
N
Y
选择一页换出
N
该页被修改否？
Y
将该页写回外存
OS命令CPU从外存读缺页
启动I/O硬件
将一页从外存换入内存
修改页表

缺页率

如果一个进程的逻辑空间为 n 页，分配到的物理块为 m（m<n）。访问页面成功（页在内存）次数为 S，缺页（页不在内存）次数为 F，则缺页率为：

$$f = \frac{F}{F+S}$$

影响缺页率的因素：

- 页面大小：页面大则缺页率低，反之缺页率高。
- 分配到的物理块数：块数多则缺页率低。
- 页面置换算法：缺页率是衡量置换算法优劣的指标。
- 程序特性：局部化程度高则缺页率低，反之缺页率高。

缺页中断处理时间

- 页面置换时还需要考虑置换代价。
- 没有被修改过的页面可以直接放弃，而修改过的页面必须进行保存。
- 如果被置换页面被修改过的概率是 β，其缺页中断处理时间为 T_a，被置换页面没有被修改过的缺页中断处理时间为 T_b，显然 $T_a > T_b$。则缺页中断处理时间 T:

$$T = \beta \times T_a + (1 - \beta) \times T_b$$

页面置换算法

- 页面置换算法是用来选择换出页面的算法。
- 页面置换算法的优劣直接影响到系统的效率，若选择不合适，可能会出现抖动（Thrashing）现象。
- 抖动：刚被淘汰出内存的页面，过后不久又要访问它，需要再次将其调入，而该页调入内存后不久又再次被淘汰出内存，然后又要访问它，如此反复，使得系统把大部分时间用在了页面的调进换出上，这种现象称为抖动。

最佳置换算法 OPT

- 最佳置换算法 OPT：选择永远不再需要的页面或最长时间以后才需要访问的页面予以淘汰。
- 最佳置换算法是一种理想化的算法，性能最好，实际上这种算法无法实现，因为页面访问的未来顺序很难精确预测，但可用该算法评价其它算法的优劣。

最佳置换算法 OPT

例：假定系统为某进程分配了 3 个物理块，进程运行时的页面走向为 1,2,3,4,1,2,5,1,2,3,4,5，开始时 3 个物理块均为空，计算采用最佳置换算法时的缺页率?

最佳置换算法 OPT

例：假定系统为某进程分配了 3 个物理块，进程运行时的页面走向为 1,2,3,4,1,2,5,1,2,3,4,5，开始时 3 个物理块均为空，计算采用最佳置换算法时的缺页率?

页面走向	1	2	3	4	1	2	5	1	2	3	4	5
物理块1												
物理块2												
物理块3												
缺页												

最佳置换算法 OPT

例：假定系统为某进程分配了 3 个物理块，进程运行时的页面走向为 1,2,3,4,1,2,5,1,2,3,4,5，开始时 3 个物理块均为空，计算采用最佳置换算法时的缺页率?

页面走向	1	2	3	4	1	2	5	1	2	3	4	5
物理块1	1											
物理块2												
物理块3												
缺页	缺											

最佳置换算法 OPT

例：假定系统为某进程分配了 3 个物理块，进程运行时的页面走向为 1,2,3,4,1,2,5,1,2,3,4,5，开始时 3 个物理块均为空，计算采用最佳置换算法时的缺页率?

页面走向	1	2	3	4	1	2	5	1	2	3	4	5
物理块1	1	1	1									
物理块2		2	2									
物理块3			3									
缺页	缺	缺	缺									

最佳置换算法 OPT

例：假定系统为某进程分配了 3 个物理块，进程运行时的页面走向为 1,2,3,4,1,2,5,1,2,3,4,5，开始时 3 个物理块均为空，计算采用最佳置换算法时的缺页率?

页面走向	1	2	3	4	1	2	5	1	2	3	4	5
物理块1	1	1	1	1								
物理块2		2	2	2								
物理块3			3	4								
缺页	缺	缺	缺	缺								

最佳置换算法 OPT

例：假定系统为某进程分配了 3 个物理块，进程运行时的页面走向为 1,2,3,4,1,2,5,1,2,3,4,5，开始时 3 个物理块均为空，计算采用最佳置换算法时的缺页率?

页面走向	1	2	3	4	1	2	5	1	2	3	4	5
物理块1	1	1	1	1	1	1						
物理块2		2	2	2	2	2						
物理块3			3	4	4	4						
缺页	缺	缺	缺	缺								

最佳置换算法 OPT

例：假定系统为某进程分配了 3 个物理块，进程运行时的页面走向为 1,2,3,4,1,2,5,1,2,3,4,5，开始时 3 个物理块均为空，计算采用最佳置换算法时的缺页率?

页面走向	1	2	3	4	1	2	5	1	2	3	4	5
物理块1	1	1	1	1	1	1	1					
物理块2		2	2	2	2	2	2					
物理块3			3	4	4	4	5					
缺页	缺	缺	缺	缺			缺					

最佳置换算法 OPT

例：假定系统为某进程分配了 3 个物理块，进程运行时的页面走向为 1,2,3,4,1,2,5,1,2,3,4,5，开始时 3 个物理块均为空，计算采用最佳置换算法时的缺页率?

页面走向	1	2	3	4	1	2	5	1	2	3	4	5
物理块1	1	1	1	1	1	1	1	1	1			
物理块2		2	2	2	2	2	2	2	2			
物理块3			3	4	4	4	5	5	5			
缺页	缺	缺	缺	缺			缺					

最佳置换算法 OPT

例：假定系统为某进程分配了 3 个物理块，进程运行时的页面走向为 1,2,3,4,1,2,5,1,2,3,4,5，开始时 3 个物理块均为空，计算采用最佳置换算法时的缺页率?

页面走向	1	2	3	4	1	2	5	1	2	3	4	5
物理块1	1	1	1	1	1	1	1	1	1	3		
物理块2		2	2	2	2	2	2	2	2	2		
物理块3			3	4	4	4	5	5	5	5		
缺页	缺	缺	缺	缺			缺			缺		

最佳置换算法 OPT

例：假定系统为某进程分配了 3 个物理块，进程运行时的页面走向为 1,2,3,4,1,2,5,1,2,3,4,5，开始时 3 个物理块均为空，计算采用最佳置换算法时的缺页率?

页面走向	1	2	3	4	1	2	5	1	2	3	4	5
物理块1	1	1	1	1	1	1	1	1	1	3	3	
物理块2		2	2	2	2	2	2	2	2	2	4	
物理块3			3	4	4	4	5	5	5	5	5	
缺页	缺	缺	缺	缺			缺			缺	缺	

最佳置换算法 OPT

例：假定系统为某进程分配了 3 个物理块，进程运行时的页面走向为 1,2,3,4,1,2,5,1,2,3,4,5，开始时 3 个物理块均为空，计算采用最佳置换算法时的缺页率?

页面走向	1	2	3	4	1	2	5	1	2	3	4	5
物理块1	1	1	1	1	1	1	1	1	1	3	3	3
物理块2		2	2	2	2	2	2	2	2	2	4	4
物理块3			3	4	4	4	5	5	5	5	5	5
缺页	缺	缺	缺	缺			缺			缺	缺	

最佳置换算法 OPT

例：假定系统为某进程分配了 3 个物理块，进程运行时的页面走向为 1,2,3,4,1,2,5,1,2,3,4,5，开始时 3 个物理块均为空，计算采用最佳置换算法时的缺页率?

页面走向	1	2	3	4	1	2	5	1	2	3	4	5
物理块1	1	1	1	1	1	1	1	1	1	3	3	3
物理块2		2	2	2	2	2	2	2	2	2	4	4
物理块3			3	4	4	4	5	5	5	5	5	5
缺页	缺	缺	缺	缺			缺			缺	缺	

缺页率 7/12

先进先出置换算法 FIFO

先进先出置换算法 FIFO：选择先进入内存的页面予以淘汰。

例：假定系统为某进程分配了3 个物理块，进程运行时的页面走向为 1,2,3,4,1,2,5,1,2,3,4,5，开始时 3 个物理块均为空，计算采用先进先出置换算法时的缺页率?

先进先出置换算法 FIFO

先进先出置换算法 FIFO：选择先进入内存的页面予以淘汰。

例：假定系统为某进程分配了3 个物理块，进程运行时的页面走向为 1,2,3,4,1,2,5,1,2,3,4,5，开始时 3 个物理块均为空，计算采用先进先出置换算法时的缺页率?

页面走向	1	2	3	4	1	2	5	1	2	3	4	5
物理块1	1	1	1	4	4	4	5	5	5	5	5	5
物理块2		2	2	2	1	1	1	1	1	3	3	3
物理块3			3	3	3	2	2	2	2	2	4	4
缺页	缺	缺	缺	缺	缺	缺	缺			缺	缺	

缺页率 9/12

先进先出置换算法 FIFO

例：假定系统为某进程分配了4 个物理块，进程运行时的页面走向为 1,2,3,4,1,2,5,1,2,3,4,5，开始时 4 个物理块均为空，计算采用先进先出置换算法时的缺页率?

先进先出置换算法 FIFO

例：假定系统为某进程分配了4 个物理块，进程运行时的页面走向为 1,2,3,4,1,2,5,1,2,3,4,5，开始时 4 个物理块均为空，计算采用先进先出置换算法时的缺页率?

页面走向	1	2	3	4	1	2	5	1	2	3	4	5
物理块1	1	1	1	1	1	1	5	5	5	5	4	4
物理块2		2	2	2	2	2	2	1	1	1	1	5
物理块3			3	3	3	3	3	3	2	2	2	2
物理块4				4	4	4	4	4	4	3	3	3
缺页	缺	缺	缺	缺			缺	缺	缺	缺	缺	缺

缺页率 10/12

先进先出置换算法 FIFO

例：假定系统为某进程分配了5 个物理块，进程运行时的页面走向为 1,2,3,4,1,2,5,1,2,3,4,5，开始时 5 个物理块均为空，计算采用先进先出置换算法时的缺页率?

先进先出置换算法 FIFO

例：假定系统为某进程分配了5 个物理块，进程运行时的页面走向为 1,2,3,4,1,2,5,1,2,3,4,5，开始时 5 个物理块均为空，计算采用先进先出置换算法时的缺页率?

页面走向	1	2	3	4	1	2	5	1	2	3	4	5
物理块1	1	1	1	1	1	1	1	1	1	1	1	1
物理块2		2	2	2	2	2	2	2	2	2	2	2
物理块3			3	3	3	3	3	3	3	3	3	3
物理块4				4	4	4	4	4	4	4	4	4
物理块5							5	5	5	5	5	5
缺页	缺	缺	缺	缺			缺					

缺页率 5/12

先进先出置换算法 FIFO

- 先进先出置换算法的出发点是最早调入内存的页面，其不再被访问的可能性会大一些。
- 被置换的页可能含有一个初始化程序段，用过后再也不会用到；但也可能含有一组全局变量，初始化时被调入内存，在整个程序运行过程中都将会用到。
- FIFO 算法易于理解与编程，但它的效率不高。

先进先出置换算法 FIFO

- 先进先出算法存在一种异常现象，即在某些情况下会出现分配给进程的物理块数增多，缺页次数有时增加，有时减少的奇怪现象，这种现象称为 Belady 异常现象 (Belady's Anomaly) 。

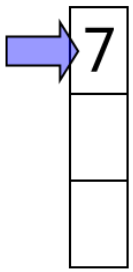
物理块数	3	4	5
缺页次数	9	10	5

页面置换算法例题

最佳置换算法

- 进程页访问序列长度：20
- 为该进程分配的物理块数：3

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

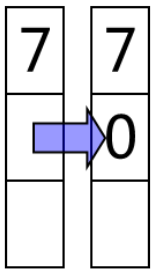


页面置换算法例题

最佳置换算法

- 进程页访问序列长度：20
- 为该进程分配的物理块数：3

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

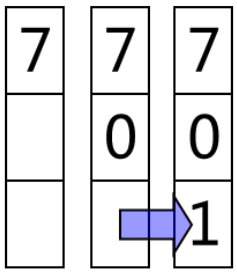


页面置换算法例题

最佳置换算法

- 进程页访问序列长度：20
- 为该进程分配的物理块数：3

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

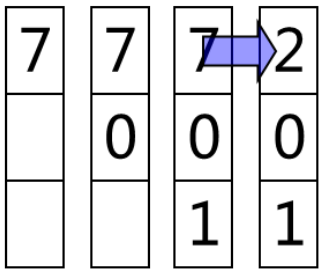


页面置换算法例题

最佳置换算法

- 进程页访问序列长度：20
- 为该进程分配的物理块数：3

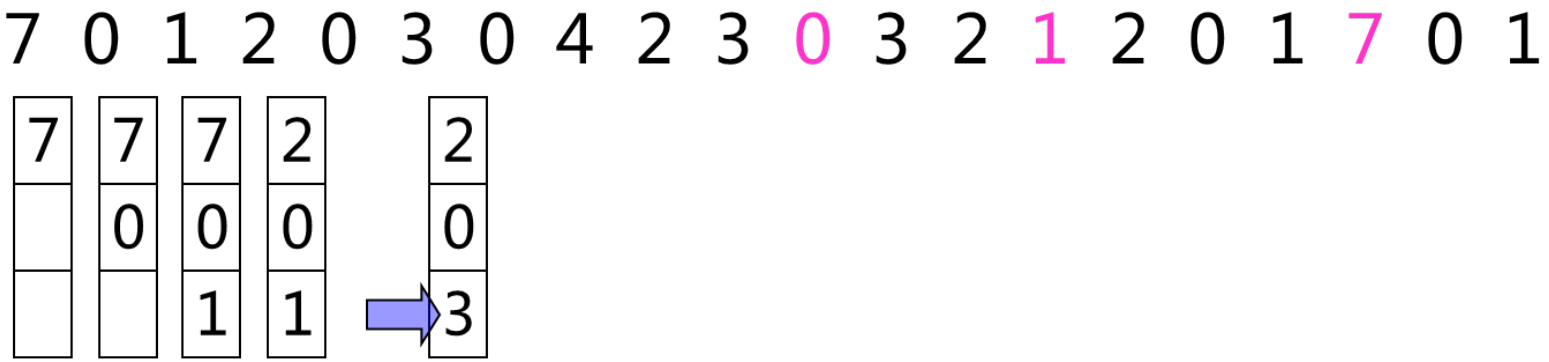
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1



页面置换算法例题

最佳置换算法

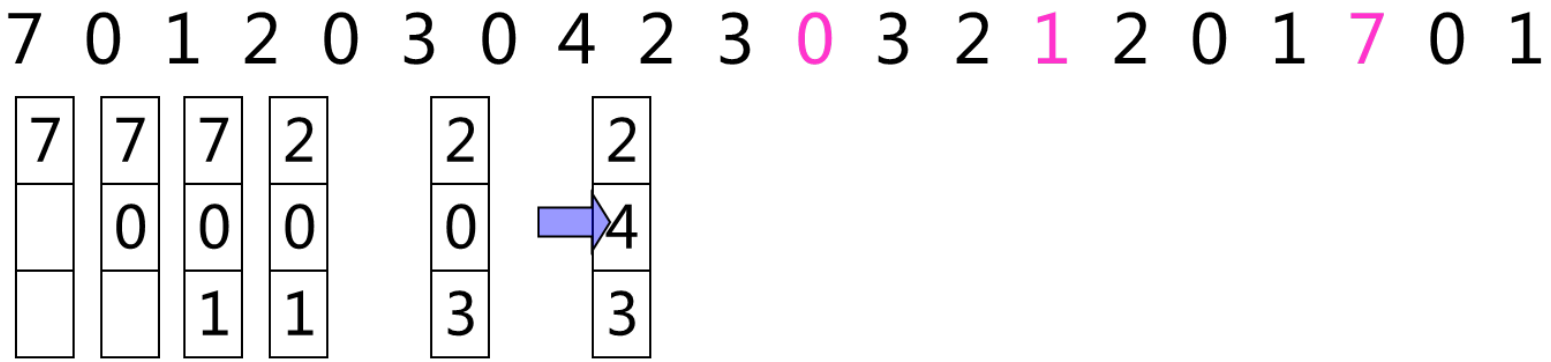
- 进程页访问序列长度：20
- 为该进程分配的物理块数：3



页面置换算法例题

最佳置换算法

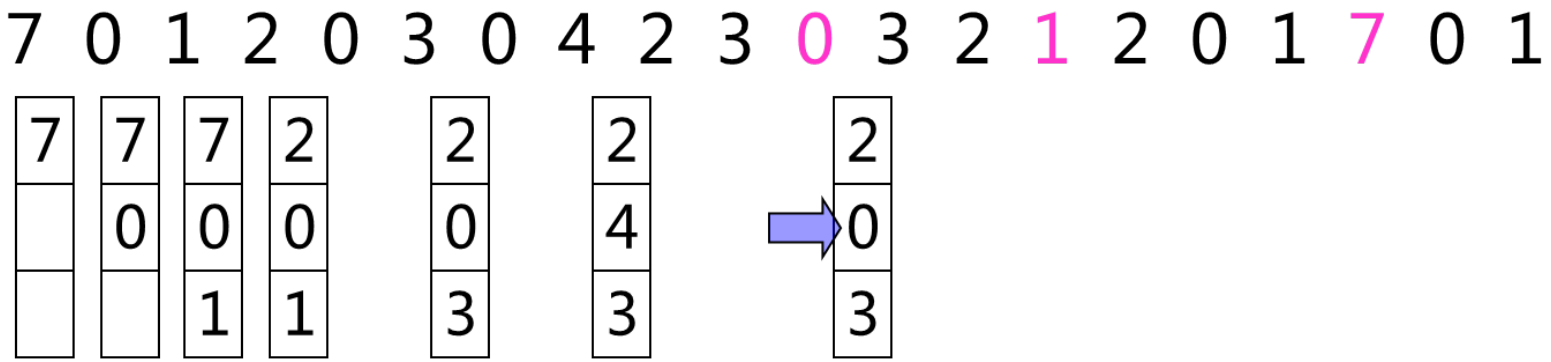
- 进程页访问序列长度：20
- 为该进程分配的物理块数：3



页面置换算法例题

最佳置换算法

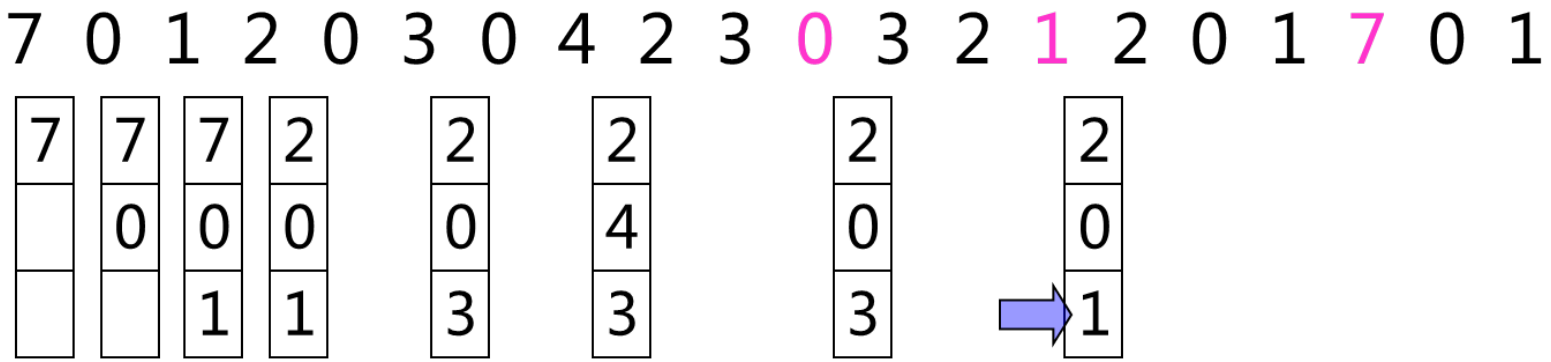
- 进程页访问序列长度：20
- 为该进程分配的物理块数：3



页面置换算法例题

最佳置换算法

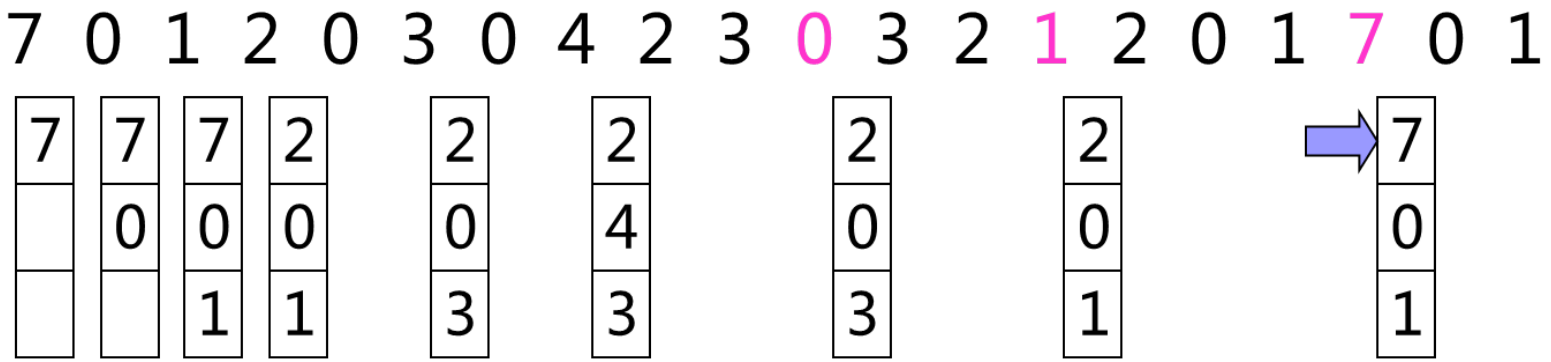
- 进程页访问序列长度：20
- 为该进程分配的物理块数：3



页面置换算法例题

最佳置换算法

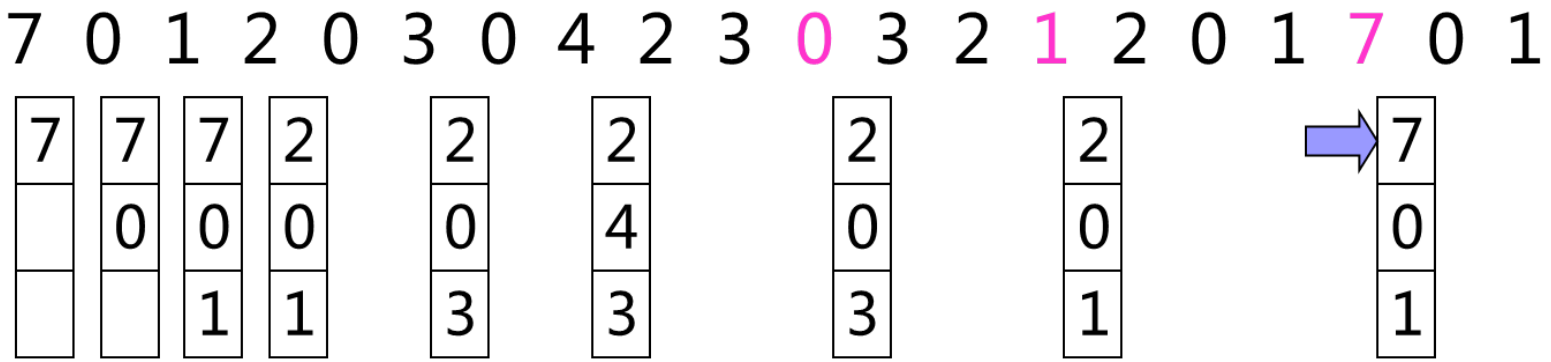
- 进程页访问序列长度：20
- 为该进程分配的物理块数：3



页面置换算法例题

最佳置换算法

- 进程页访问序列长度：20
- 为该进程分配的物理块数：3



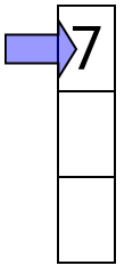
缺页次数：9，置换次数：6

页面置换算法例题

先进先出置换算法

- 进程页访问序列长度：20
- 为该进程分配的物理块数：3

7　0　1　2　0　3　0　4　2　3　0　3　2　1　2　0　1　7　0　1

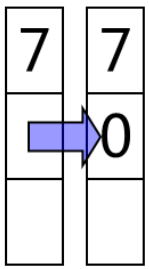


页面置换算法例题

先进先出置换算法

- 进程页访问序列长度：20
- 为该进程分配的物理块数：3

7　0　1　2　0　3　0　4　2　3　0　3　2　1　2　0　1　7　0　1

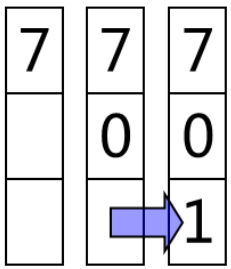


页面置换算法例题

先进先出置换算法

- 进程页访问序列长度：20
- 为该进程分配的物理块数：3

7　0　1　2　0　3　0　4　2　3　0　3　2　1　2　0　1　7　0　1

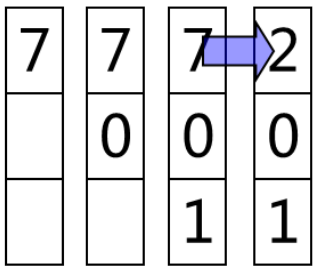


页面置换算法例题

先进先出置换算法

- 进程页访问序列长度：20
- 为该进程分配的物理块数：3

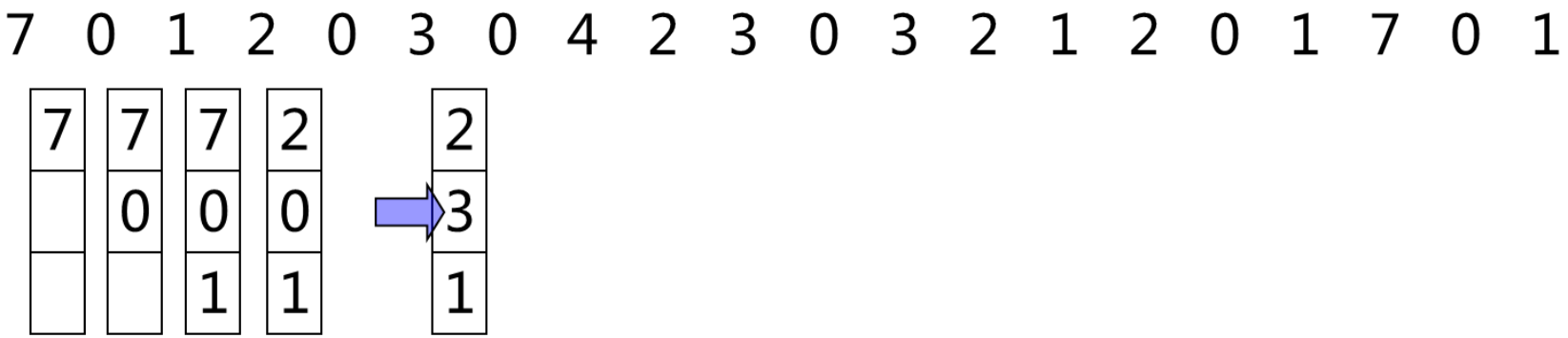
7　0　1　2　0　3　0　4　2　3　0　3　2　1　2　0　1　7　0　1



页面置换算法例题

先进先出置换算法

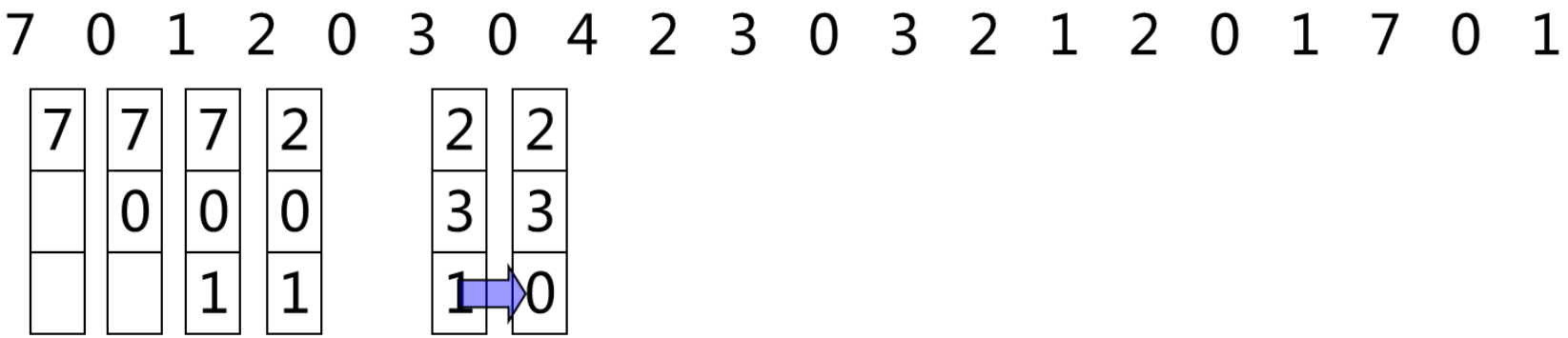
- 进程页访问序列长度：20
- 为该进程分配的物理块数：3



页面置换算法例题

先进先出置换算法

- 进程页访问序列长度：20
- 为该进程分配的物理块数：3



页面置换算法例题

先进先出置换算法

- 进程页访问序列长度：20
- 为该进程分配的物理块数：3

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2		2	2	4												
	0	0	0		3	3	3												
		1	1		1	0	0												

页面置换算法例题

先进先出置换算法

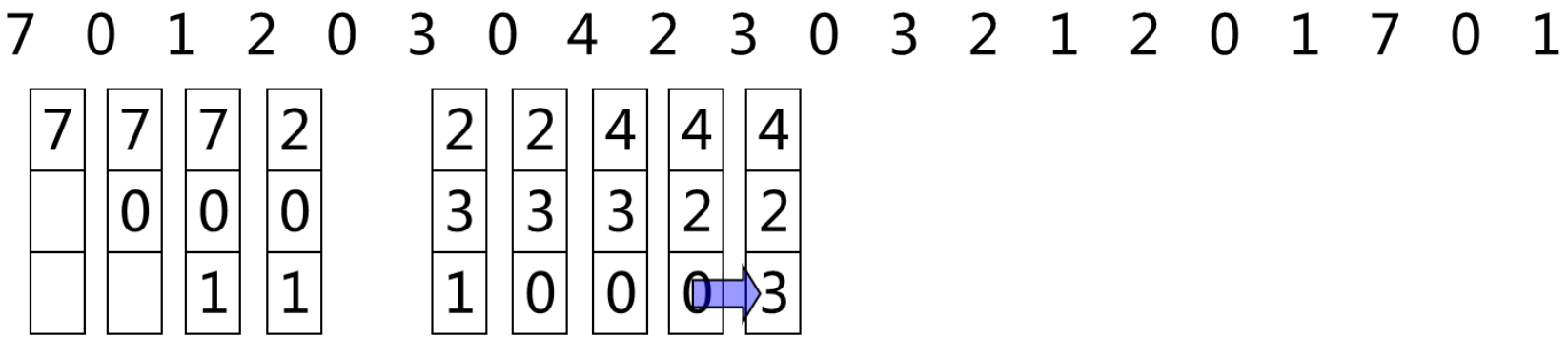
- 进程页访问序列长度：20
- 为该进程分配的物理块数：3

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2		2	2	4	4											
	0	0	0		3	3	3	2											
		1	1		1	0	0	0											

页面置换算法例题

先进先出置换算法

- 进程页访问序列长度：20
- 为该进程分配的物理块数：3



页面置换算法例题

先进先出置换算法

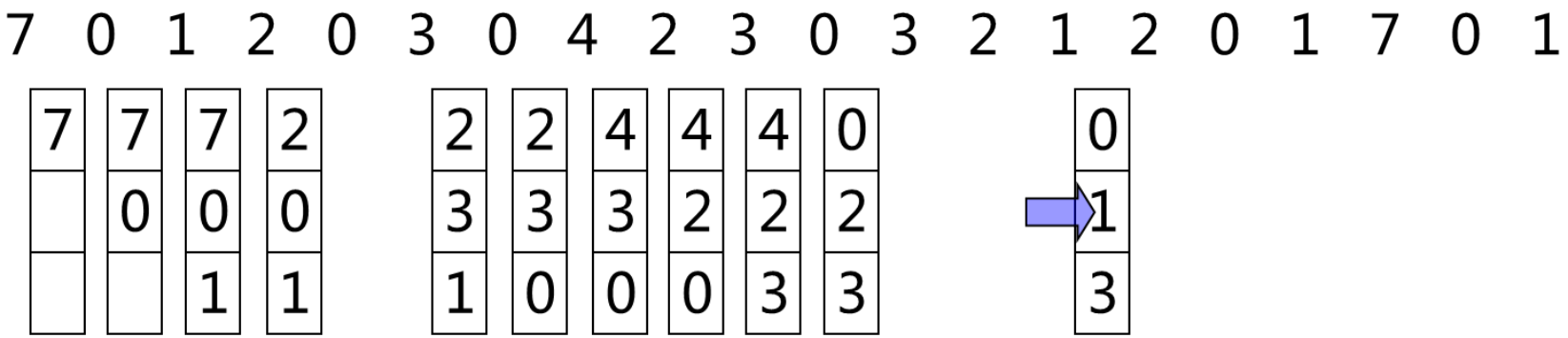
- 进程页访问序列长度：20
- 为该进程分配的物理块数：3

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2		2	2	4	4	4	0									
	0	0	0		3	3	3	2	2	2									
		1	1		1	0	0	0	3	3									

页面置换算法例题

先进先出置换算法

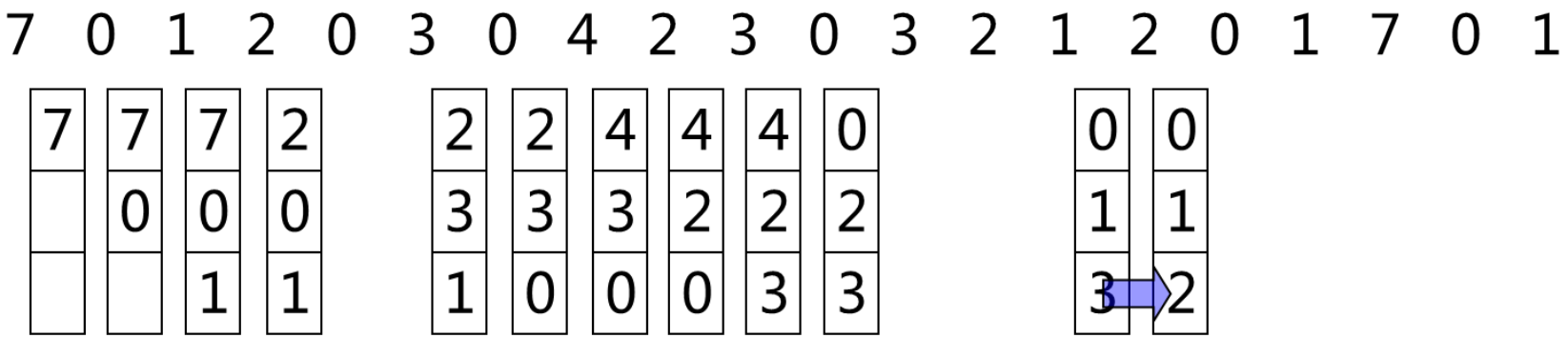
- 进程页访问序列长度：20
- 为该进程分配的物理块数：3



页面置换算法例题

先进先出置换算法

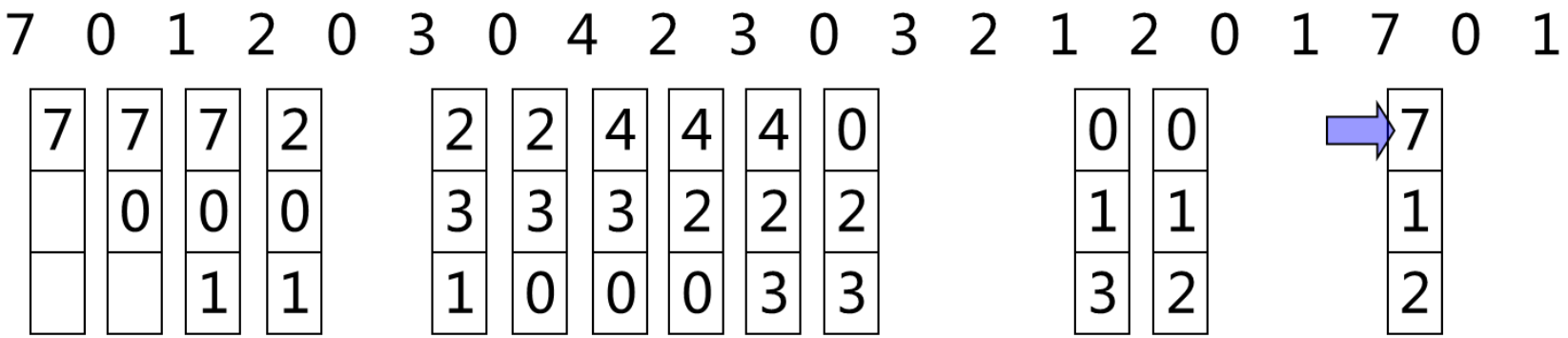
- 进程页访问序列长度：20
- 为该进程分配的物理块数：3



页面置换算法例题

先进先出置换算法

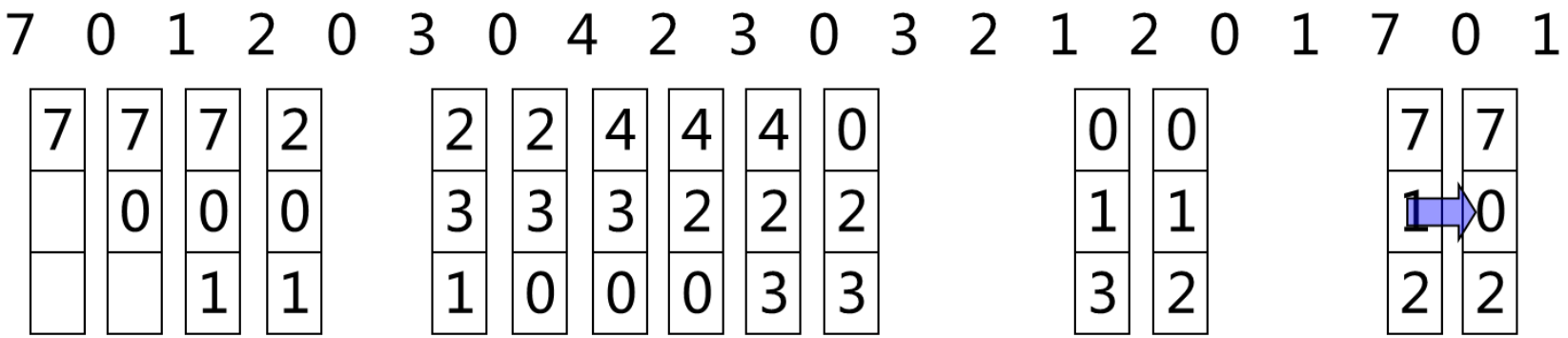
- 进程页访问序列长度：20
- 为该进程分配的物理块数：3



页面置换算法例题

先进先出置换算法

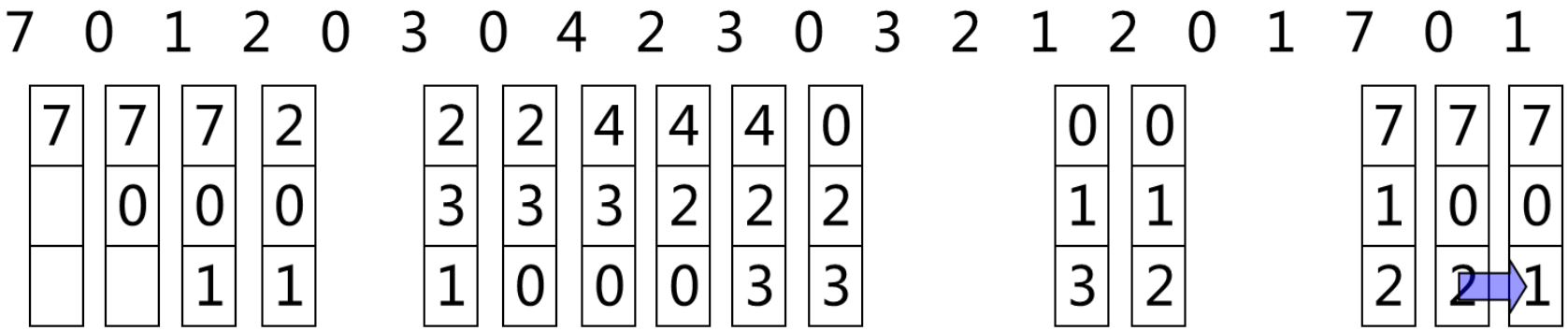
- 进程页访问序列长度：20
- 为该进程分配的物理块数：3



页面置换算法例题

先进先出置换算法

- 进程页访问序列长度：20
- 为该进程分配的物理块数：3



页面置换算法例题

先进先出置换算法

- 进程页访问序列长度：20
- 为该进程分配的物理块数：3

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2		2	2	4	4	4	0			0	0			7	7	7
	0	0	0		3	3	3	2	2	2			1	1			1	0	0
		1	1		1	0	0	0	3	3			3	2			2	2	1

缺页次数：15，置换次数：12

最近最久未使用算法（LRU）

- 最近最久未使用算法 LRU：选择最近一段时间最长时间没有被访问过的页面予以淘汰。
- 算法的出发点：如果某个页面被访问了，则它可能马上还要访问。如果很长时间未被访问，则它在最近一段时间也不会被访问。 OPT 向前看，LRU 向后看

最近最久未使用算法（LRU）

- 最近最久未使用算法 LRU：选择最近一段时间最长时间没有被访问过的页面予以淘汰。
- 算法的出发点：如果某个页面被访问了，则它可能马上还要访问。如果很长时间未被访问，则它在最近一段时间也不会被访问。OPT 向前看，LRU 向后看
- 该算法的性能接近于最佳算法，但实现起来较困难。因为要找出最近最久未使用的页面，必须为每一页设置相关记录项，用于记录页面的访问情况，并且每访问一次页面都须更新该信息。这将使系统的开销加大，所以在实际系统中往往使用该算法的近似算法。

最近最久未使用算法（LRU）

LRU：淘汰最近一段时间最长时间没有被访问过的页面。

例：假定系统为某进程分配了3 个物理块，进程运行时的页面走向为 1,2,3,4,1,2,5,1,2,3,4,5，开始时 3 个物理块均为空，计算采用最近最久未使用算法时的缺页率?

最近最久未使用算法（LRU）

LRU：淘汰最近一段时间最长时间没有被访问过的页面。

例：假定系统为某进程分配了3 个物理块，进程运行时的页面走向为 1,2,3,4,1,2,5,1,2,3,4,5，开始时 3 个物理块均为空，计算采用最近最久未使用算法时的缺页率?

页面走向	1	2	3	4	1	2	5	1	2	3	4	5
物理块1	1	1	1	4	4	4	5	5	5	3	3	3
物理块2		2	2	2	1	1	1	1	1	1	4	4
物理块3			3	3	3	2	2	2	2	2	2	5
缺页	缺	缺	缺	缺	缺	缺	缺			缺	缺	缺

缺页率 10/12

最近最久未使用算法的硬件支持

LRU 置换算法虽然是一种比较好的算法，但实现开销很大，必须有硬件的支持。

最近最久未使用算法的硬件支持

LRU 置换算法虽然是一种比较好的算法，但实现开销很大，必须有硬件的支持。

1 寄存器

为了记录某个进程在内存中各页的使用情况，为每个在内存中的页面配置一个移位寄存器，可表示为

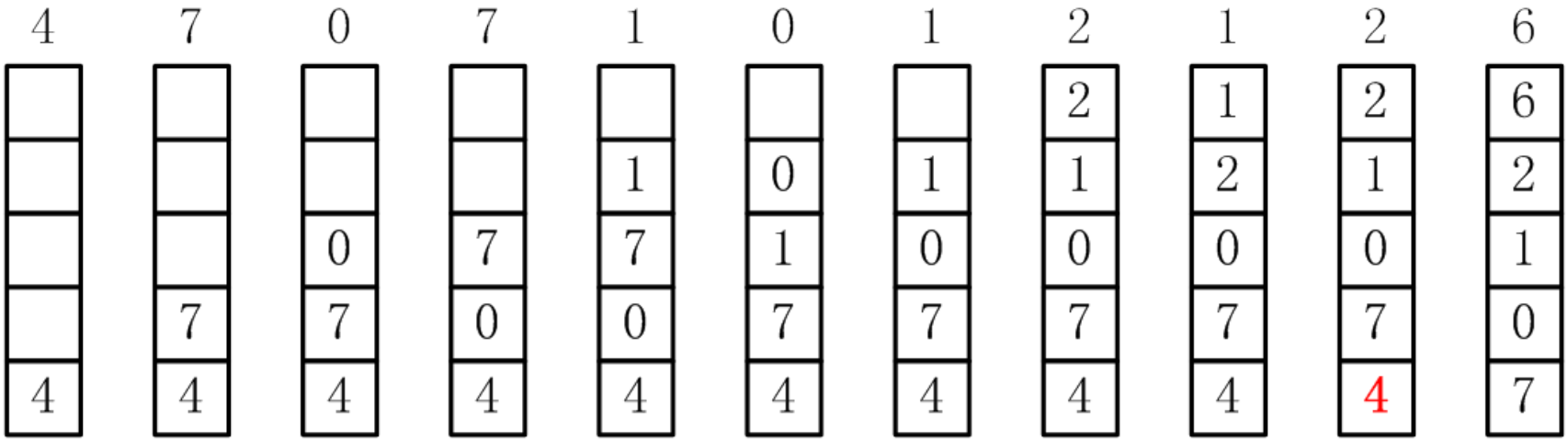
$$R = R_{n-1}R_{n-2}R_{n-3}\cdots R_2R_1R_0$$

每当进程访问某页面时，将该页面对应寄存器的最高位 (R_{n-1}) 置 1，系统定期 (如 100ms) 将寄存器右移一位并将最高位补 0，如果把 n 位寄存器的数看作是一个整数，于是寄存器数值最小的页面是最久未使用的页面。

最近最久未使用算法的硬件支持

2 栈

利用一特殊的栈保存当前使用的页号，每当进程访问某页面时，把被访问页面移到栈顶，于是栈底的页面就是最久未使用的页面。



最近最少使用置换算法（LFU）

- 最近最少使用置换算法 LFU：选择在最近时期使用最少的页面为淘汰页。
- LFU 置换算法为在内存中的每个页面设置一个移位寄存器来记录该页面被访问的频率。
- 每当进程访问某页面时，将该页面对应寄存器的最高位 (R_{n-1}) 置 1，系统定期 (如 100ms) 将寄存器右移一位并将最高位补 0。在一段时间内，ΣR_i 最小的页面就是最近最少使用的页面。

LRU 与 LFU 的区别

- LRU 是最近最久未使用页面置换算法 (Least Recently Used)，也就是首先淘汰最长时间未被使用的页面。
- LFU 是最近最少使用置换算法 (Least Frequently Used)，也就是淘汰一定时期内被访问次数最少的页。

LRU 与 LFU 的区别

- LRU 是最近最久未使用页面置换算法 (Least Recently Used)，也就是首先淘汰最长时间未被使用的页面。
- LFU 是最近最少使用置换算法 (Least Frequently Used)，也就是淘汰一定时期内被访问次数最少的页。
- 例如，进程分配了 3 个物理块，若所需页面走向为 2 1 2 1 2 3 4（请求页面 4 时会发生缺页中断）

LRU 与 LFU 的区别

- LRU 是最近最久未使用页面置换算法 (Least Recently Used)，也就是首先淘汰最长时间未被使用的页面。
- LFU 是最近最少使用置换算法 (Least Frequently Used)，也就是淘汰一定时期内被访问次数最少的页。
- 例如，进程分配了 3 个物理块，若所需页面走向为 2 1 2 1 2 3 4（请求页面 4 时会发生缺页中断）
  按 LRU 算法, 应换页面 1（1 页面最近最久未被使用)。
  但按 LFU 算法应换页面 3（页面 3 最近最少使用，最近只使用了一次)。
- LRU 是看时间长短，而 LFU 是看使用频率。

例题

例：程序要将一个二维数组 var A:array[1 … 100,1 … 100] of integer 置 0。二维数组按先行后列的次序存储，对一个采用 LRU 置换算法的页式虚拟存储系统，假设每页可存放 200 个整数，若分配给进程用于存放数组（不作它用）的内存块数是 2，开始时数据页尚未装入内存。请分别写出下列程序计算机执行过程中的缺页次数。

```
For i:=1 to 100 do
    for j:=1 to 100 do
        A[i,j]:=0
```

```
For j:=1 to 100 do
    for i:=1 to 100 do
        A[i,j]:=0
```

例题

例：程序要将一个二维数组 var A:array[1 … 100,1 … 100] of integer 置 0。二维数组按先行后列的次序存储，对一个采用 LRU 置换算法的页式虚拟存储系统，假设每页可存放 200 个整数，若分配给进程用于存放数组（不作它用）的内存块数是 2，开始时数据页尚未装入内存。请分别写出下列程序计算机执行过程中的缺页次数。

```
For i:=1 to 100 do
    for j:=1 to 100 do
        A[i,j]:=0
```

① 100∗100/200=50 次

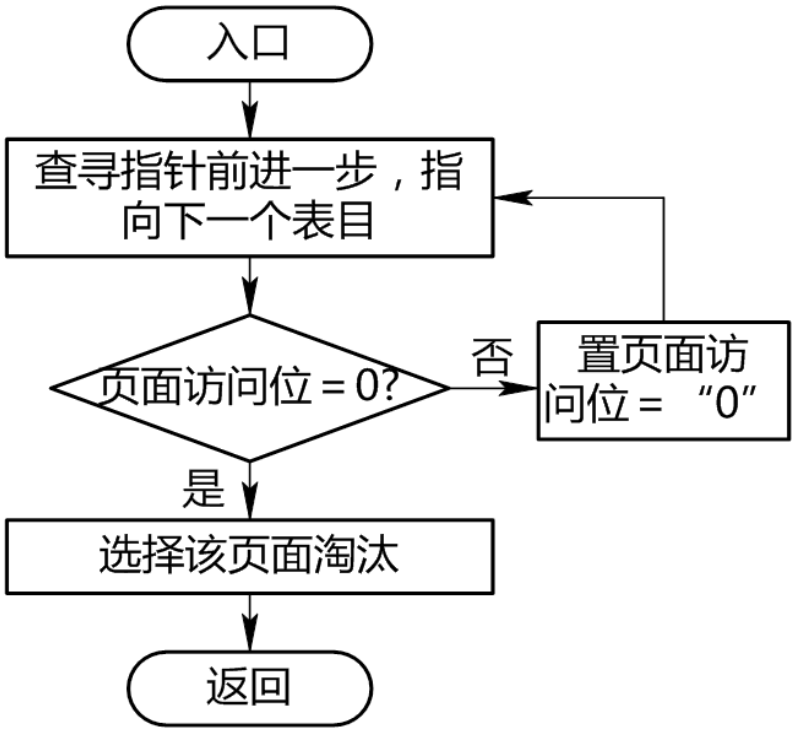
```
For j:=1 to 100 do
    for i:=1 to 100 do
        A[i,j]:=0
```

② 100∗100/2 = 5000 次

Clock 置换算法

- Clock 置换算法是 LRU 和 FIFO 的折衷（LRU 的近似算法，减少硬件消耗）。也称为最近未使用算法 (Not Recently Used，NRU)
- 该算法为每页设置一个访问位，并将内存中的所有页链接成一个循环队列。当某页被访问时，其访问位被置 1。
- 置换算法在选择一页淘汰时，只需检查页的访问位。如果是 0，就选择该页换出；若为 1，则重新将它置 0，暂不换出，而给该页第二次驻留内存的机会，再按照 FIFO 算法检查下一个页面（循环，不是回到队首）。
- 当检查到队列中的最后一个页面时，若其访问位仍为 1，则再返回到队首去检查第一个页面。

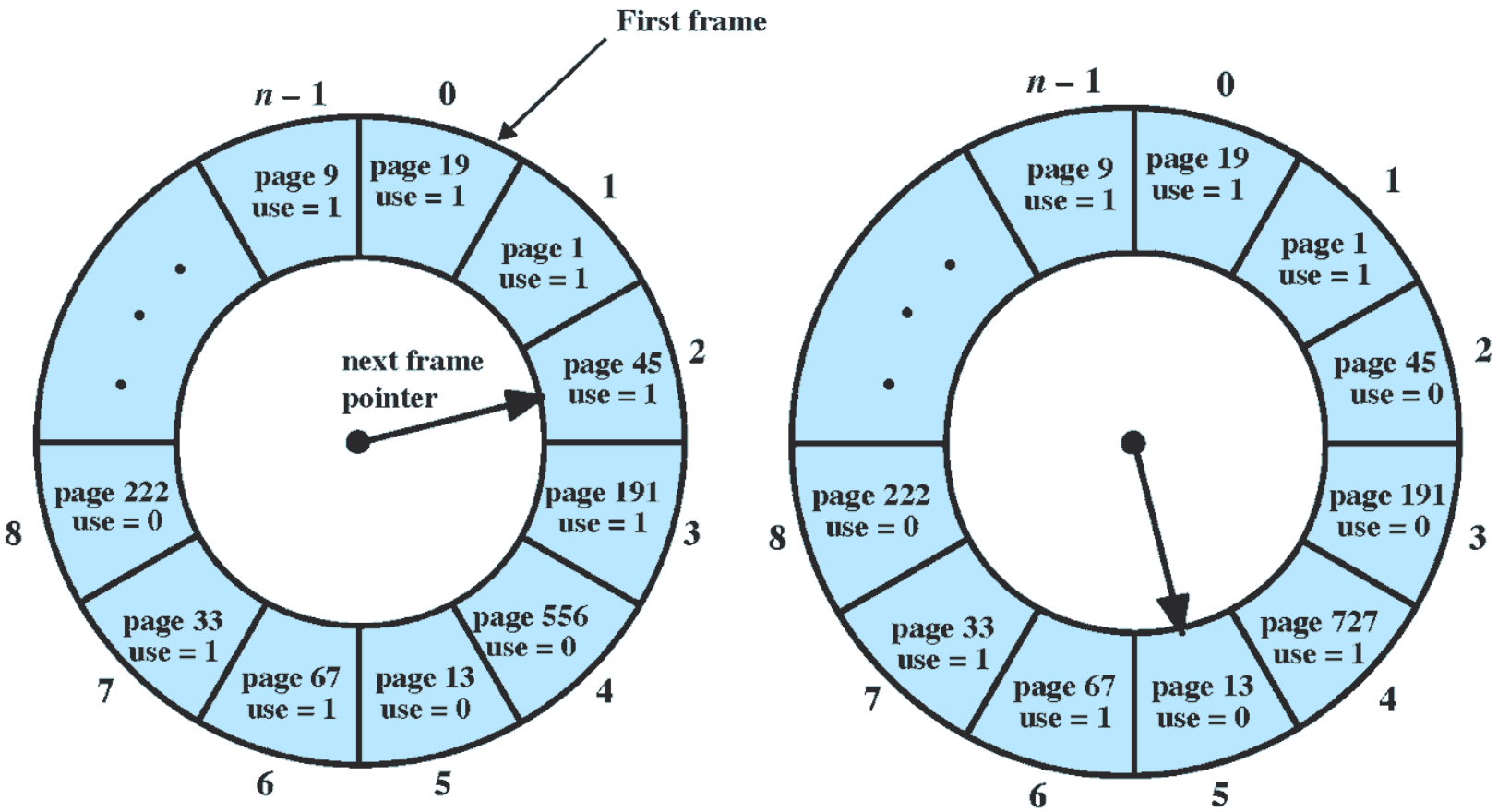
Clock 置换算法



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

Clock 置换算法



假如某个页被频繁访问，那么它就不会被置换出去

Clock 置换算法

*表示该页面的访问位为1

页面走向	2	3	2	1	5	2	4	5	3	2	5	2
物理块1	2*	2*	2*	2*	5*	5*	5*	5*	3*	3*	3*	3*
物理块2		3*	3*	3*	3	2*	2*	2*	2	2*	2*	2*
物理块3				1*	1	1	4*	4*	4	4	5*	5*
缺页	缺	缺		缺	缺	缺	缺		缺		缺	

第二轮淘汰2#页面

12次页面引用，8次缺页中断，5次页面置换

Clock 置换算法

*表示该页面的访问位为1

页面走向	2	3	2	1	5
物理块1	2*	2*	2*	2*	5*
物理块2		3*	3*	3*	3
物理块3				1*	1
缺页	缺	缺		缺	缺

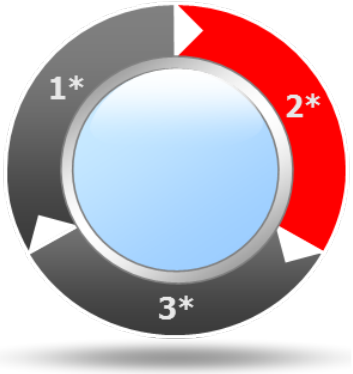
第二轮淘汰2#页面

Clock 置换算法

*表示该页面的访问位为1

页面走向	2	3	2	1	5
物理块1	2*	2*	2*	2*	5*
物理块2		3*	3*	3*	3
物理块3				1*	1
缺页	缺	缺		缺	缺

第二轮淘汰2#页面

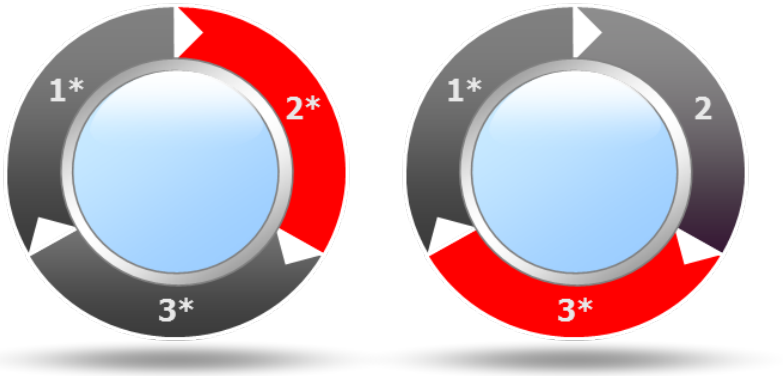


Clock 置换算法

*表示该页面的访问位为1

页面走向	2	3	2	1	5
物理块1	2*	2*	2*	2*	5*
物理块2		3*	3*	3*	3
物理块3				1*	1
缺页	缺	缺		缺	缺

第二轮淘汰2#页面

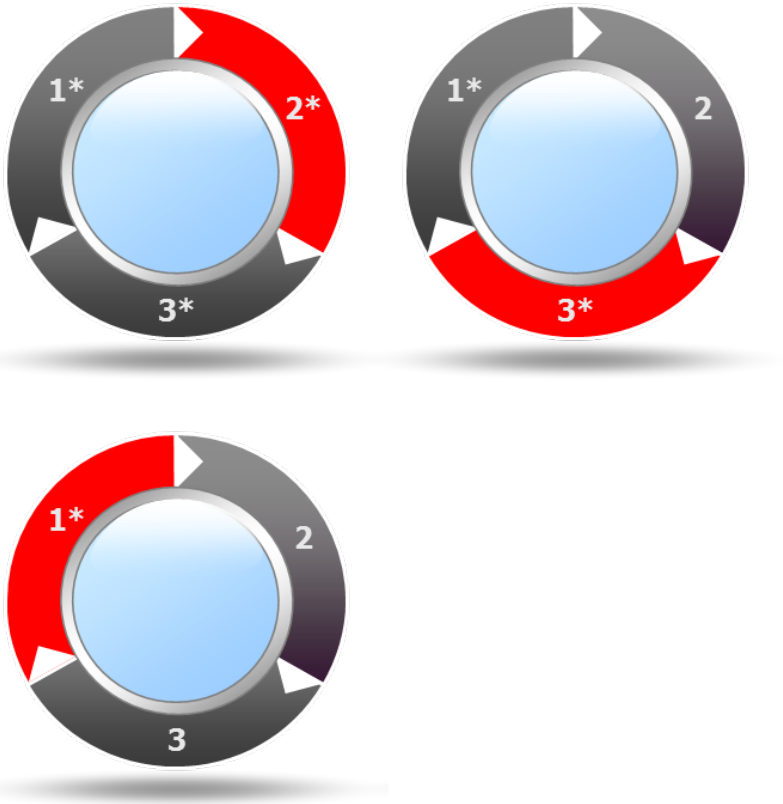


Clock 置换算法

*表示该页面的访问位为1

页面走向	2	3	2	1	5
物理块1	2*	2*	2*	2*	5*
物理块2		3*	3*	3*	3
物理块3				1*	1
缺页	缺	缺		缺	缺

第二轮淘汰2#页面

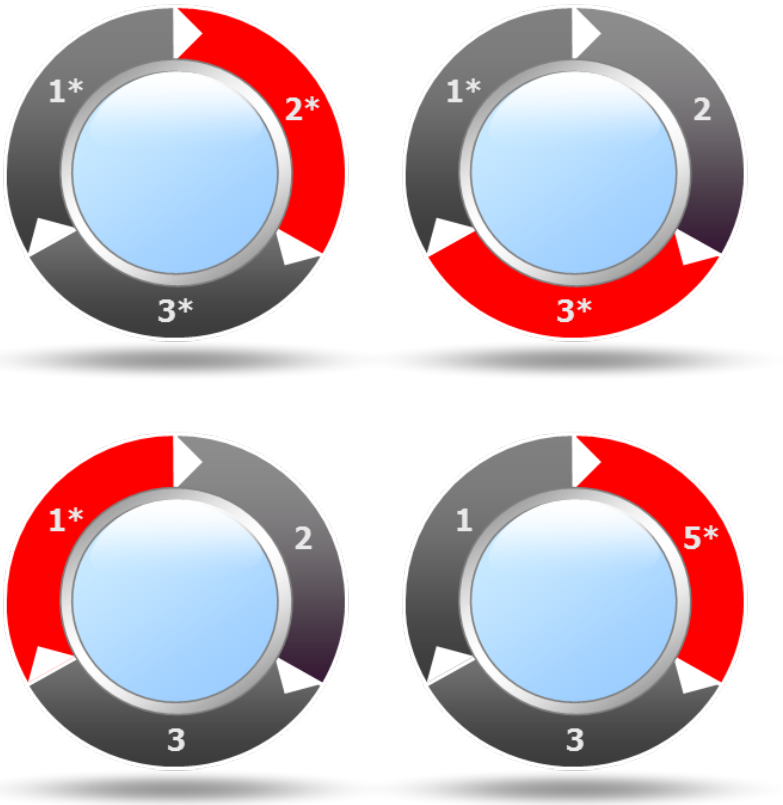


Clock 置换算法

*表示该页面的访问位为1

页面走向	2	3	2	1	5
物理块1	2*	2*	2*	2*	5*
物理块2		3*	3*	3*	3
物理块3				1*	1
缺页	缺	缺		缺	缺

第二轮淘汰2#页面



改进型 Clock 置换算法

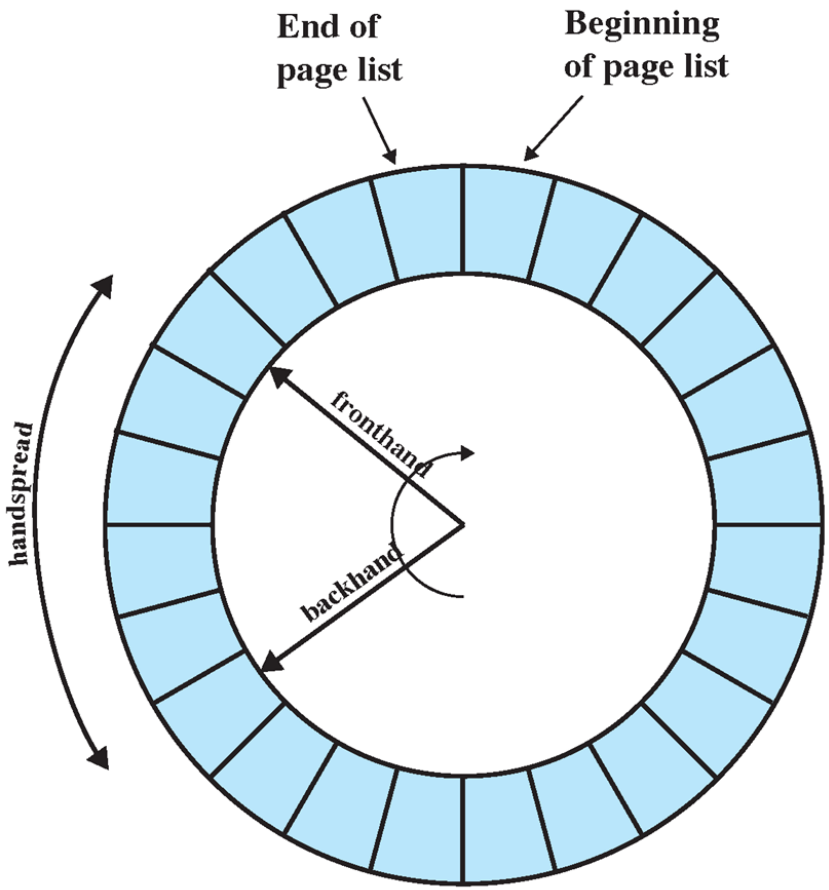
- 改进型 Clock 置换算法：除须考虑页面的使用情况外，还增加一个因素，即置换代价，这样选择页面换出时，既要是未使用过的页面，又要是未被修改过的页面。
- 由访问位 A 和修改位 M 可以组合成下面四种类型的页面：
  1. A=0, M=0: 最佳淘汰页
  2. A=0, M=1
  3. A=1, M=0
  4. A=1, M=1

改进型 Clock 置换算法的执行过程

1. 从指针所指示的当前位置开始扫描循环队列，寻找 A=0 且 M=0 的第一类页面，将所遇到的第一个页面作为所选中的淘汰页。在第一次扫描期间不改变访问位 A。
2. 如果第一步失败，开始第二轮扫描，寻找 A=0 且 M=1 的第二类页面，将所遇到的第一个这类页面作为淘汰页。在第二轮扫描期间，将所有扫描过的页面的访问位都置 0。
3. 如果第二步也失败，则将指针返回到开始的位置，并将所有的访问位复 0。然后重复第一步，如果仍失败，必要时再重复第二步。

双指针 Clock 置换算法

双指针 Clock 置换算法：定时淘汰页面。前指针清除访问位，后指针淘汰页面。



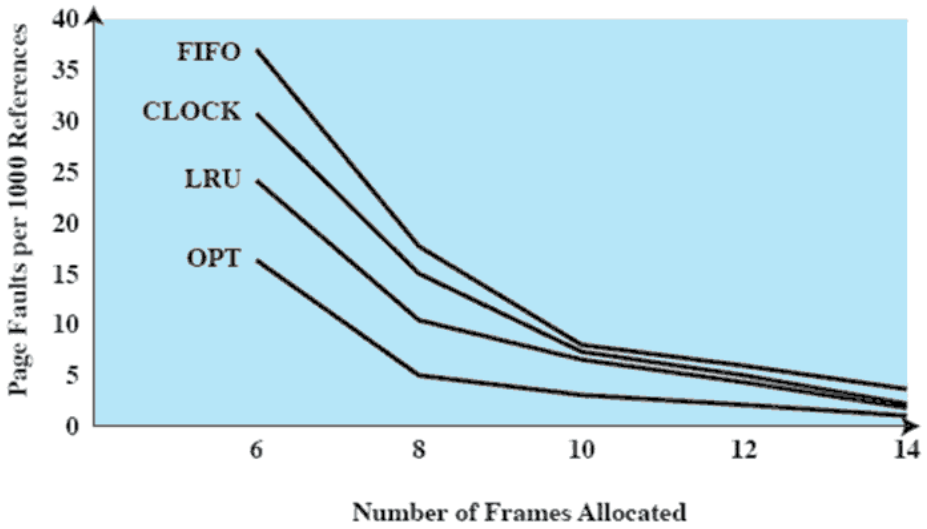
影响页面换入换出效率的若干因素

1. 页面置换算法。
2. 回写磁盘的频率。已经被修改过的页面，换出时应当回写磁盘。如果建立一个已修改换出页面链表，可以暂不回写。当达到一定数目后，再将它们一起写入磁盘，这样能大大减少 I/O 操作的次数。
3. 读入内存的频率。已修改换出页面链表上的页面在回写前，如果需要被再次访问，就不需要从外存调入，直接从链表中获取。

页面缓冲算法 PBA

- 页面缓冲算法：用 FIFO 算法选择被置换页，选择换出的页面不是立即换出，而是放入两个链表之一，如果页面未被修改，就将其归入到空闲页面链表的末尾，否则将其归入已修改页面链表末尾。
- 这些空闲页面和已修改页面会在内存中停留一段时间。如果这些页面被再次访问，只需将其从相应链表中移出，就可以返回进程，从而减少一次 I/O 开销。
- 需调入新页，则将新页读入到空闲页面链表的第一个页面中，然后将其从该链表中移出。
- 当已修改的页面达到一定数目后，再将它们一起写入磁盘。这样能大大减少 I/O 操作的次数。

页面置换算法的比较



算法	注释
OPT（最佳）算法	不可能实现，用于衡量其它算法
FIFO（先进先出）算法	公平，效率不高
LRU（最久未使用）算法	很优秀，但实现代价太高
LFU（最少使用）算法	类似于LRU
Clock（时钟）算法	现实的算法

访问内存的有效时间 EAT

1. 被访问页面在内存，且对应的页表项在快表中。

   设访问快表的时间为λ，访问内存的时间为t。

   $EAT = \lambda + t$

2. 被访问页面在内存，但对应的页表项不在快表中。

   这种情况不缺页，但需两次访问内存。一次读页表，并更新快表，一次读数据。

   $EAT = \lambda + t + \lambda + t$　或　$EAT = \lambda + t + t$

   更新快表 λ 与读数据t可并行。

3. 被访问页面不在内存。

   缺页。设缺页中断处理时间为ϵ（含更新页表的时间）。

   $EAT = \lambda + t + \epsilon + \lambda + t$

访问内存的有效时间 EAT

- 上面的几种讨论没有考虑快表的命中率和缺页率等因素。
- 如果考虑快表的命中率 a和缺页率 f

$$EAT = a\ (\lambda + t) + (1 - a)\ [\ \lambda + t + \lambda + f\ (\epsilon + t\ ) + (1 - f)\ t\ ]$$

- 如果不考虑快表，仅考虑缺页率 f，即令上式中 $\lambda = 0$ 和 $a = 0$,
- $$EAT = t + f\ (\epsilon + t\ ) + (1 - f)\ t$$

访问内存的有效时间 EAT

- 上面的几种讨论没有考虑快表的命中率和缺页率等因素。
- 如果考虑快表的命中率 a和缺页率 f

$$EAT = a\ (\lambda + t) + (1 - a)\ [\ \lambda + t + \lambda + f\ (\epsilon + t\ ) + (1 - f)\ t\ ]$$

- 如果不考虑快表，仅考虑缺页率 f，即令上式中 $\lambda = 0$ 和 $a = 0$,
- $EAT = t + f\ (\epsilon + t\ ) + (1 - f)\ t$

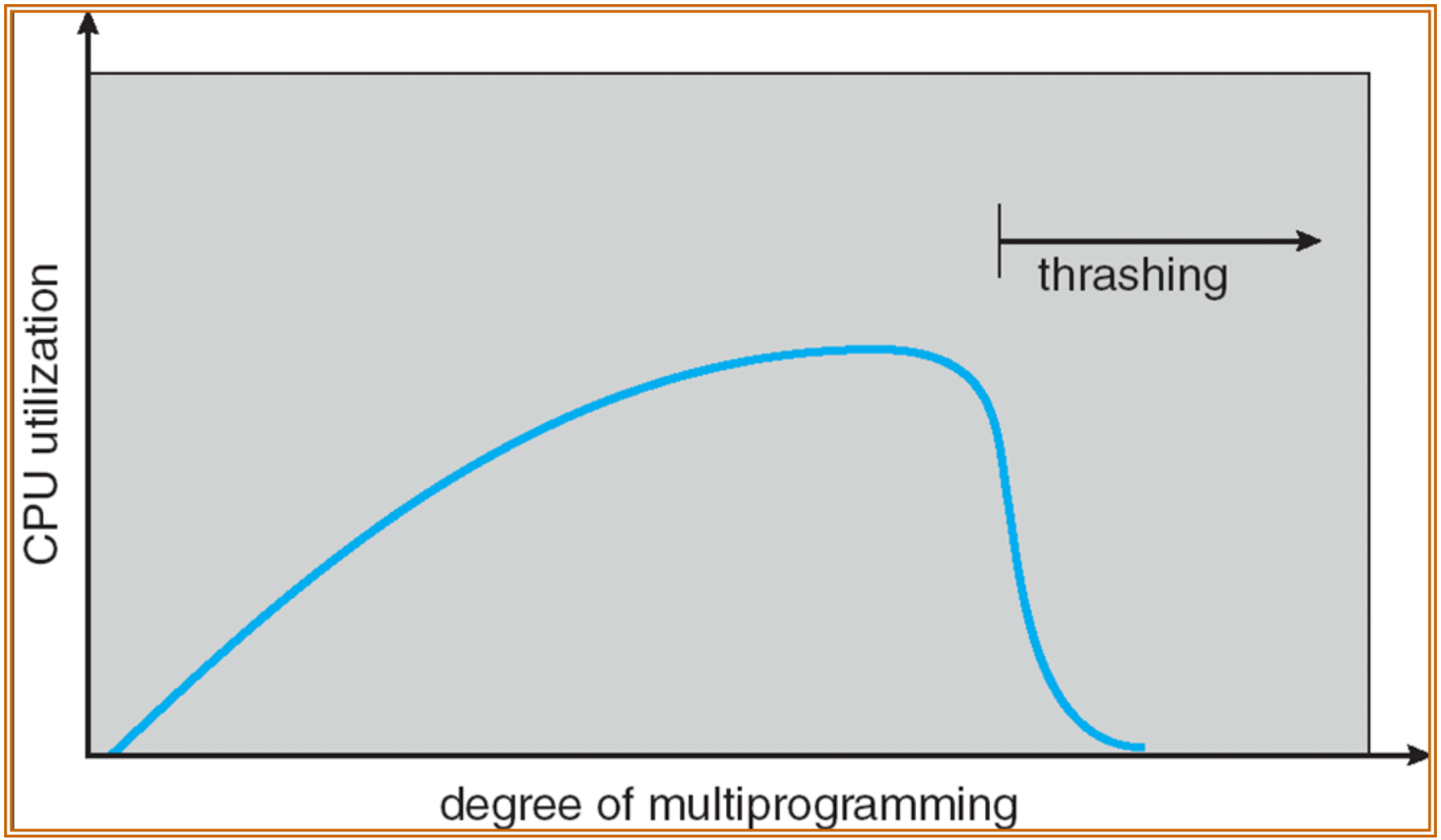
内存访问时间 t　约为 100ns(纳秒)=0.1μs (微秒)

缺页中断时间 ϵ　约为 25ms (毫秒) =25000 μs (微秒)

多道程序度与处理机的利用率

- 由于虚拟存储器系统能从逻辑上扩大内存，人们希望在系统中能运行更多的进程，即增加多道程序度，以提高处理机的利用率。
- 如果多道程度过高，页面在内存与外存之间频繁调度，以至于调度页面所需时间比进程实际运行的时间还多，此时系统效率急剧下降，甚至导致系统崩溃。这种现象称为颠簸或抖动（thrashing）。

多道程序度与处理机的利用率



多道程序度与处理机的利用率

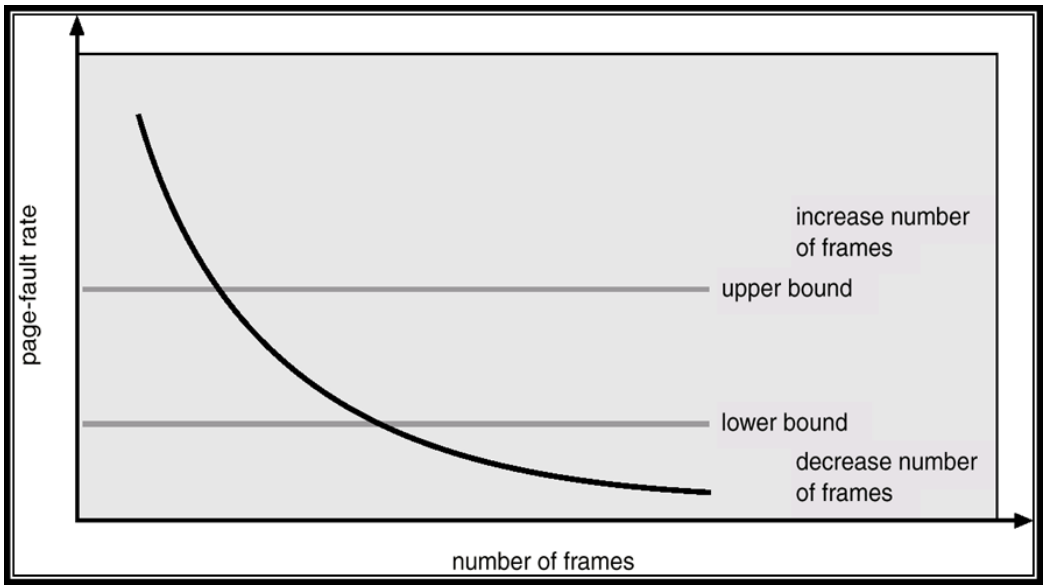
- 抖动的后果：缺页率急剧增加，内存有效存取时间加长，系统吞吐量骤减（趋近于零）；系统已基本不能完成什么任务。
- 抖动产生原因：同时运行的进程数过多，进程频繁访问的页面数高于可用的物理块数，造成进程运行时频繁缺页。CPU 利用率太低时，调度程序就会增加多道程序度，将新进程引入系统中，反而进一步导致处理机利用率的下降。
- 操作系统需要一种降低缺页率、防止抖动的内存管理方法：工作集策略

驻留集和工作集管理

- 驻留集：指请求分页存储管理中给进程分配的物理页面（块）的集合。
- 驻留集大小即是这个集合中元素的个数。
- 每个进程的驻留集越小，则同时驻留内存的进程就越多，CPU 利用率越高。
- 进程的驻留集太小的话，则缺页率高，请求调页的开销增大。
- 抖动的原因：多道程序度过高，导致平均驻留集过小。

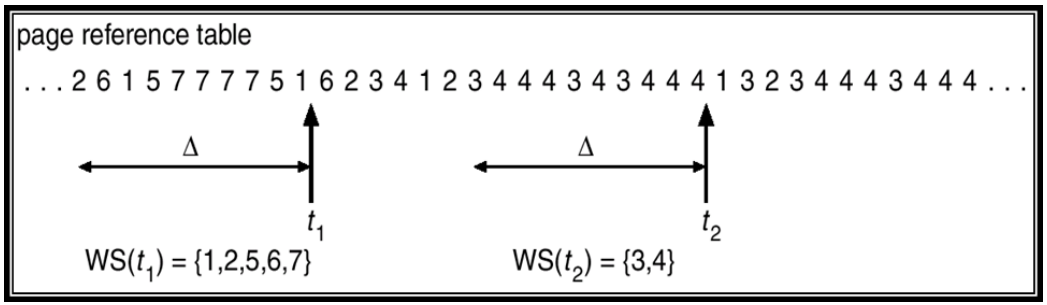
驻留集和工作集管理

- 缺页率与系统为进程分配物理块的多少（驻留集的大小）有关。缺页率随着分配物理块的增加而减少。
- 当物理块数达到某个数值时，物理块数的增加对缺页率没有明显影响。当物理块数小于某个数值时，减少一块都会对缺页率有较大影响。



工作集

- 1968 年 Denning 提出了工作集(Working Set) 理论。
- 工作集是指在某段时间间隔 Δ 里，进程实际要访问的页面的集合。
- 把进程在某段时间间隔 Δ 里，在时间 t 的工作集记为 w(t,Δ)，变量 Δ 称为工作集“窗口尺寸”。
- 对于给定的页面走向，如果 Δ = 10 次存储访问，在 t1 时刻的工作集是 W(t1,10)=(1,2,5,6,7)，在 t2 时刻，工作集是 W(t2,10)=(3,4)



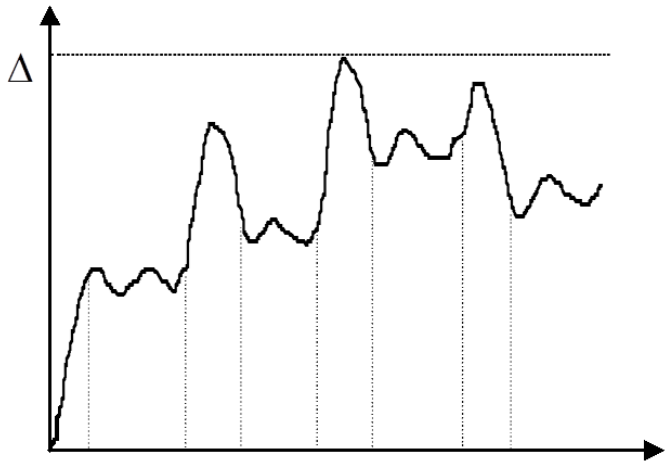
工作集

工作集是 Δ 的非降函数。$w(t, \Delta) \subseteq w(t, \Delta + 1)$

Sequence of Page References	Window Size, D			
	2	3	4	5
24	24	24	24	24
15	24,15	24,15	24,15	24,15
18	15,18	24,15,18	24,15,18	24,15,18
23	18,23	15,18,23	24,15,18,23	24,15,18,23
24	23,24	18,23,24	*	*
17	24,17	23,24,17	18,23,24,17	15,18,23,24,17
18	17,18	24,17,18	*	18,23,24,27
24	18,24	*	24,17,18	*
18	*	18,24	*	24,17,18
17	18,17	24,18,17	*	*
17	17	18,17	*	*
15	17,15	17,15	18,17,15	24,18,17,15
24	15,24	17,15,24	17,15,24	*
17	24,17	*	*	17,15,24
24	*	24,17	*	*
18	18,24	17,24,18	24,17,18	15,24,17,18

工作集

- 工作集的大小是变化的。
- 相对比较稳定的阶段和快速变化的阶段交替出现。
- 根据局部性原理，进程会在一段时间内相对稳定在某些页面构成的工作集上。
- 当局部性区域的位置改变时，工作集大小快速变化。
- 当工作集窗口滑过这些页面后，工作集又稳定在一个局部性阶段。



工作集

- 工作集精确度与窗口尺寸 Δ 的选择有关。如果 Δ 太小，那么它不能表示进程的局部特征；如果 Δ 为无穷大，那么工作集合是进程执行需要的所有页面的集合。
- 如果页面正在使用，它就落在工作集中；如果不再使用，它将不出现在相应的工作集中。
- 工作集是局部性原理的近似表示。
- 如果能找出一个作业的各个工作集，并求出其页面数最大者，就可估计出该进程所需的物理块数。
- 利用工作集模型可以进行页面置换。工作集页面置换法的基本思想：找出一个不在工作集中的页面，把它淘汰。

工作集策略（Working Set Strategy）

- 类似于 LRU 算法，工作集用进程过去某段时间内的行为作为未来某段时间内行为的近似。

利用工作集进行驻留集调整的策略

1. 操作系统监视每个进程的工作集变化情况。
2. 只有当一个进程的工作集在内存中时才执行该进程。
3. 定期淘汰驻留集中不在工作集中的页面。
4. 总是让驻留集包含工作集（不能包含时则增大驻留集）

工作集策略（Working Set Strategy）

工作集策略存在的问题

1. 工作集过去的变化未必能够预示工作集未来的变化。
2. 记录每个进程的工作集变化开销太大。（需要为每个进程维护一个基于时间顺序的访问页面队列）
3. 工作集窗口尺寸 Δ 大小的最优值难以确定。

- 工作集策略的思想是合理的，很多操作系统试图采用近似的工作集策略。
- 操作系统可以不直接监视工作集大小，而是通过监视缺页率来达到类似效果。

抖动的预防方法

1. **采取局部置换策略**

   仅允许进程在自身范围内进行置换。即使发生抖动，也可以把影响限制在较小范围内。

2. **在处理机调度中引入工作集策略**

   - 操作系统跟踪每个进程的工作集，并为进程分配大于其工作集的物理块。
   - 如果还有空闲物理块，则可以再调度一个进程进内存以增加多道程序度。
   - 如果所有工作集之和增加到超过了可用物理块的总数，操作系统会暂停一个进程，将其页面调出并且将其物理块分配给其他进程，防止出现抖动现象。

抖动的预防方法

3. 用 L=S 准则调节缺页率（Denning, 1980）
   - L：缺页之间的平均时间。S：平均缺页服务时间
   - L 大于 S，很少缺页，磁盘能力没有被充分利用。
   - L 小于 S，频繁缺页，超过磁盘的处理能力。
   - 调整并发程序度，使得 L 与 S 接近。这种情况下，磁盘和处理机到可以达到最佳利用率。
   - 一种类似的策略称为“50% 准则”策略：让磁盘保持 50% 的利用率，这时 CPU 也达到最高的利用率。
4. 挂起若干进程
   - 当多道程序度偏高，已影响到处理机的利用率时，为了防止发生抖动，系统必须减少多道程序的数目。把某些低优先级的进程挂起，从而腾出内存空间。

请求分段存储管理方式

- 请求分段存储管理系统与请求分页存储管理系统一样，为用户提供了一个比内存空间大得多的虚拟存储器。
- 在请求分段存储管理系统中，作业运行之前，只要求将当前需要的若干个分段装入内存，便可启动作业运行。

1. 调段功能
2. 置换功能
3. 紧缩功能

请求分段中的硬件支持与共享保护

请求分段中的硬件支持

1. 段表机制
2. 缺段中断机构
3. 地址变换机构

分段共享与保护

1. 共享段表
2. 共享段的分配与回收
3. 分段保护（越界检查、存取控制检查、环保护机构）

段表机制

段名	段长	段的基址	存取方式	访问字段A	修改位M	存在位P	增补位	外存始址

标志本分段的存取属性(读,写,执行)

段表机制

段名	段长	段的基址	存取方式	访问字段A	修改位M	存在位P	增补位	外存始址

记录该段被访问的频繁程度
（与分页相应字段同）

段表机制

段名	段长	段的基址	存取方式	访问字段A	修改位M	存在位P	增补位	外存始址

该段在调入内存后是否被修改过，供置换时参考

段表机制

段名	段长	段的基址	存取方式	访问字段A	修改位M	存在位P	增补位	外存始址

指示本段是否已调入内存，供程序访问时参考

段表机制

段名	段长	段的基址	存取方式	访问字段A	修改位M	存在位P	增补位	外存始址

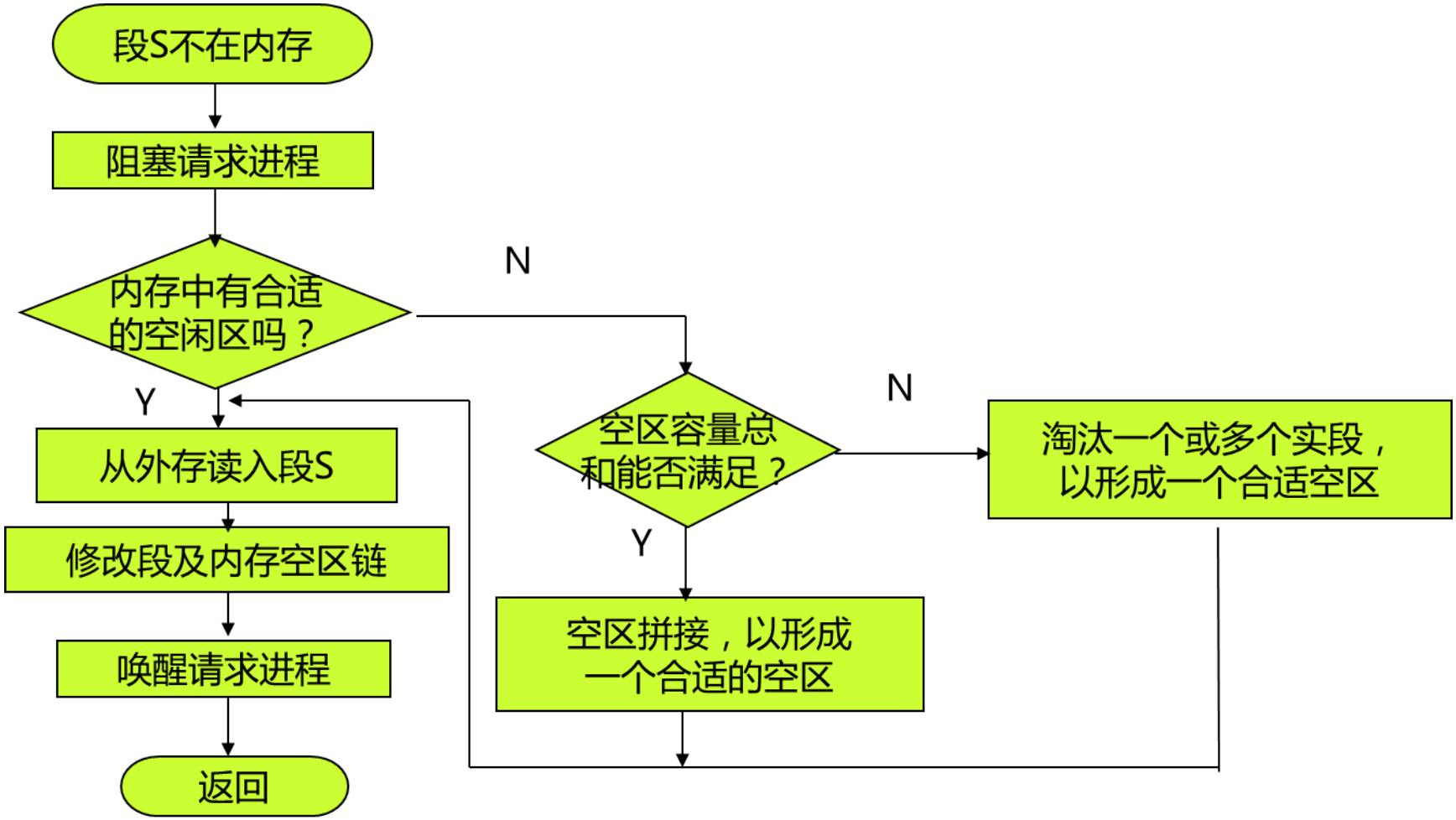
本段在运行过程中，是否做过动态增长

段表机制

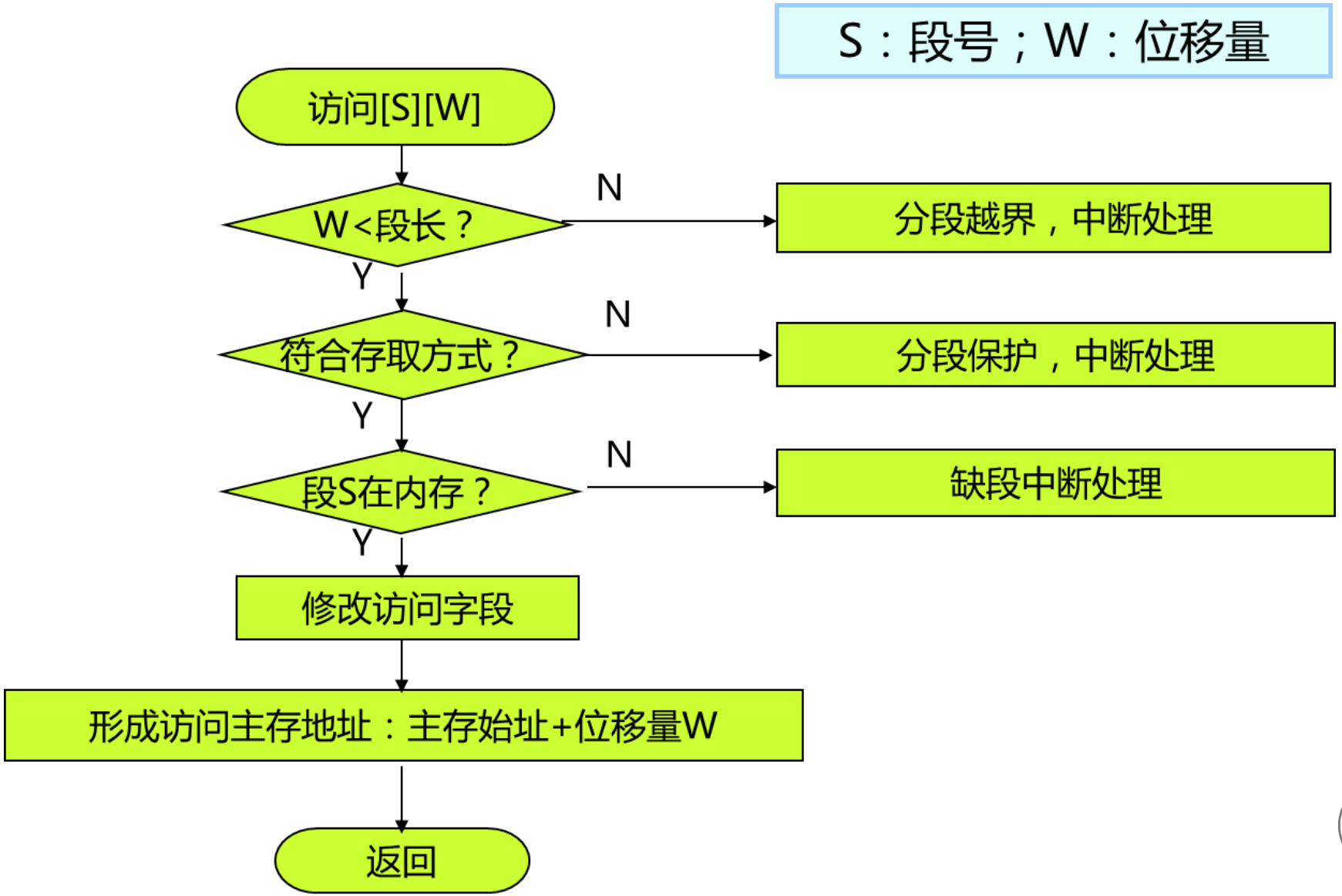
段名	段长	段的基址	存取方式	访问字段A	修改位M	存在位P	增补位	外存始址

本段在外存中的起始地址

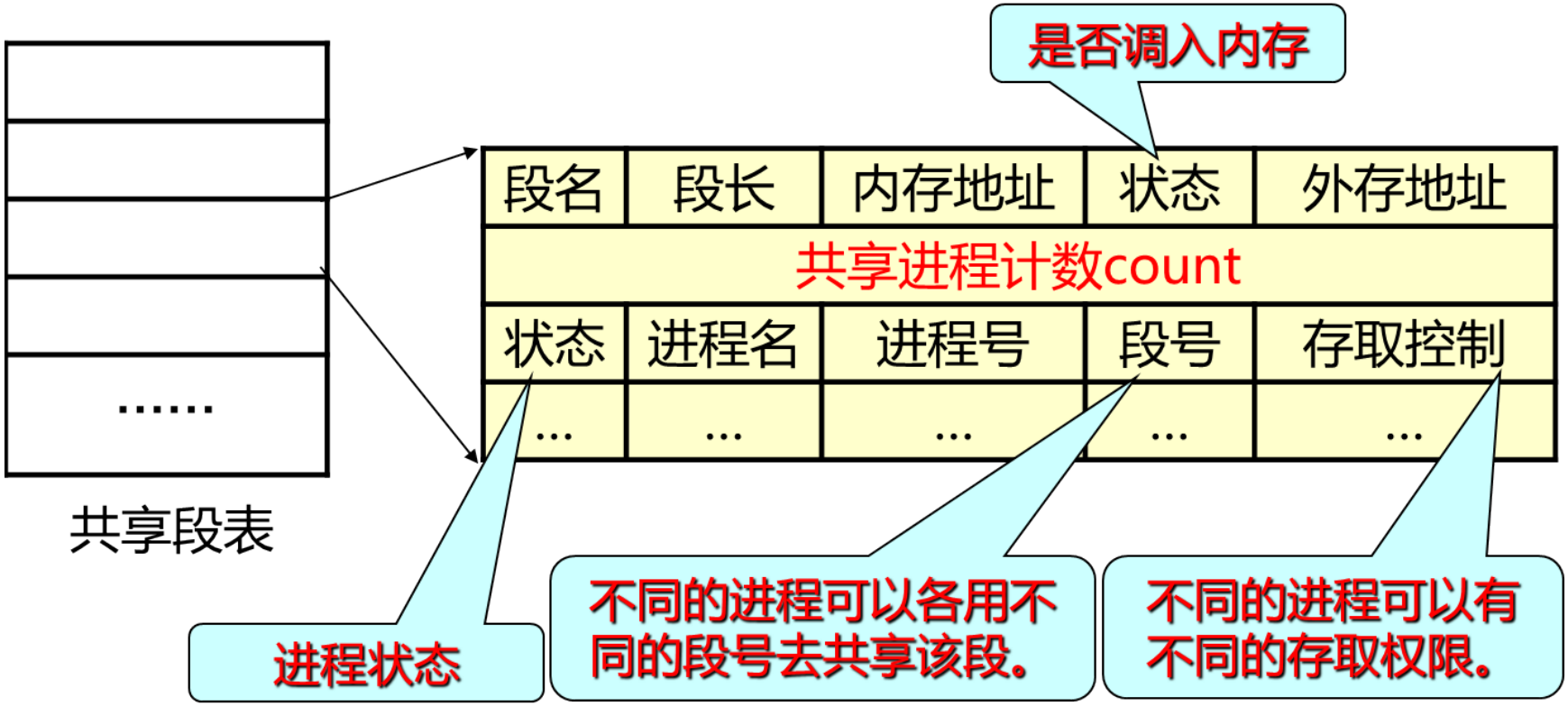
缺段中断机构


段S不在内存
阻塞请求进程
内存中有合适的空闲区吗？
N
Y
空区容量总和能否满足？
N
Y
淘汰一个或多个实段，以形成一个合适空区
从外存读入段S
修改段及内存空区链
唤醒请求进程
返回
空区拼接，以形成一个合适的空区

地址变换机构


S：段号；W：位移量
访问[S][W]
W<段长？
N
分段越界，中断处理
Y
符合存取方式？
N
分段保护，中断处理
Y
段S在内存？
N
缺段中断处理
Y
修改访问字段
形成访问主存地址：主存始址+位移量W
返回

共享段表


是否调入内存
段名
段长
内存地址
状态
外存地址
共享进程计数count
状态
进程名
进程号
段号
存取控制
…
…
…
…
…
……
共享段表
进程状态
不同的进程可以各用不同的段号去共享该段。
不同的进程可以有不同的存取权限。

共享段的分配与回收

- **共享段的分配**

  当第一个使用共享段的进程提出请求时，由系统为该共享段分配一物理区，并调入该共享段，同时修改相应的段表（该段的内存地址）和共享段表，把 count 置为 1。当其它进程需要调用此段时，不需再调入，只需修改相应的段表和共享段表，再执行 count :=count+1 操作。

- **共享段的回收**

  当共享共享段的某进程不再使用该共享段时，修改相应的段表和共享段表，执行 count :=count-1 操作。当最后一共享此段的进程也不再需要此段时，则系统回收此共享段的物理区，同时修改共享段表（删除该表项）。

分段管理的保护

- 地址越界保护
  - 先利用段表寄存器中的段表长度与逻辑地址中的段号比较，若段号超界则产生越界中断。
  - 再利用段表项中的段长与逻辑地址中的段内位移进行比较，若段内位移大于段长，也会产生越界中断。
  - 注：在允许段动态增长的系统中，允许段内位移大于段长。
- 访问控制保护（存取控制保护）

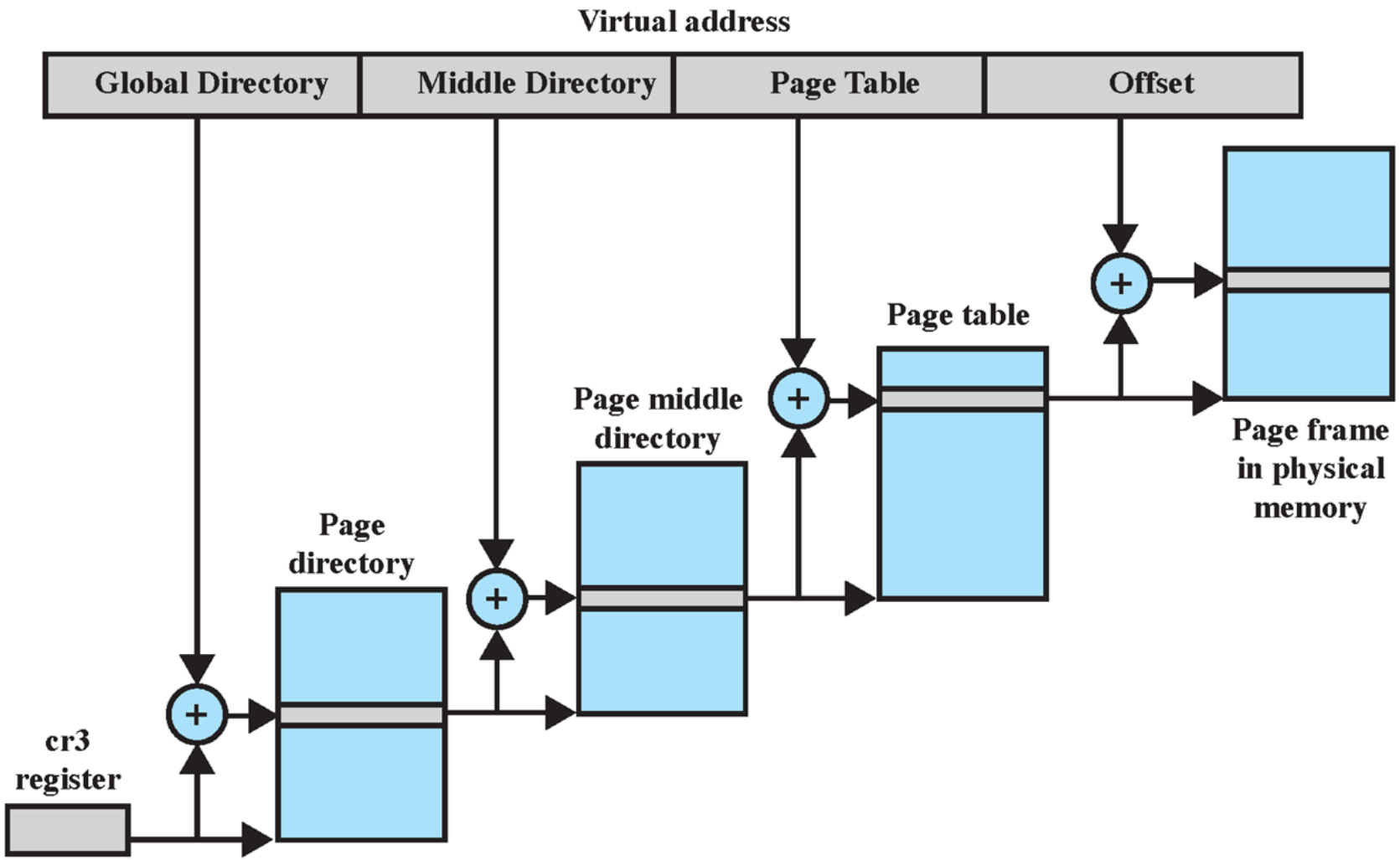
  在段表中设置存取控制字段，用于规定对该段的访问方式。

分段管理的保护

- 环保护机构
  - 环保护机构是一种功能较完善的保护机制。在该机制中规定：低编号的环具有高优先权。
  - OS 核心处于 0 环内；某些重要的实用程序和操作系统服务占居中间环；而一般的应用程序则被安排在外环上。
  - 在环系统中，程序的访问和调用应遵循一定的规则：

1. 一个程序可以访问同环或较低特权环的数据
2. 一个程序可以调用同环或较高特权环的服务

Linux 系统中的存储器管理



最佳页面大小的推导

- 设进程的平均大小为 s 字节，页面尺寸为 p 字节，每个页表项占 e 字节。
- 则每个进程需要的页数大约为 s/p，占用 s·e /p 字节的页表空间。
- 每个进程的内部碎片平均为 p/2。因此，由页表和内部碎片带来的总开销是：

$$s \cdot e/p + p/2$$

- 我们希望由页表和内部碎片带来的总开销最小。

最佳页面大小的推导

- 由页表和内部碎片带来的总开销：

$$s \cdot e/p + p/2$$

- 上式中对 p 求导，令其等于 0，得到方程：

$$-s \cdot e/p^2 + 1/2 = 0$$

- 由上式解出 p，得到最佳页面尺寸公式

$$p = \sqrt{2s \cdot e}$$

- 例如：如果进程的平均大小 s = 1 MB，e = 8 B，则最佳页面尺寸是 4 KB。

课堂练习

1. 若处理器有 32 位地址，则虚拟地址空间为（　　）。
   - A、2GB　　B、4GB
   - C、32MB　　D、32GB
2. 请求分页存储管理中，若把页面尺寸增加一倍，在程序顺序执行时，则一般缺页中断次数会（　　）。
   - A、增加　　B、减少
   - C、不变　　D、可能增加也可能减少
3. 下面哪些存储分配方法可能使系统抖动（　　）。
   - A、页式　　B、段式
   - C、请求分页　　D、段页式

课堂练习

1 若处理器有 32 位地址，则虚拟地址空间为（　　）。

- A、2GB　　B、4GB
- C、32MB　　D、32GB

2 请求分页存储管理中，若把页面尺寸增加一倍，在程序顺序执行时，则一般缺页中断次数会（　　）。

- A、增加　　B、减少
- C、不变　　D、可能增加也可能减少

3 下面哪些存储分配方法可能使系统抖动（　　）。

- A、页式　　B、段式
- C、请求分页　　D、段页式

1B　2B　3C

课堂练习

4 采用虚拟存储器的前提是程序执行时某些部分的互斥性和（　　）。

- A、顺序性　　B、局部性
- C、并发性　　D、并行性

5 虚拟内存的最大容量只受（　　）的限制。

- A、物理内存大小　　B、磁盘空间大小
- C、内存和磁盘容量大小　　D、计算机地址结构

课堂练习

4 采用虚拟存储器的前提是程序执行时某些部分的互斥性和（　　）。

- A、顺序性　　B、局部性
- C、并发性　　D、并行性

5 虚拟内存的最大容量只受（　　）的限制。

- A、物理内存大小　　B、磁盘空间大小
- C、内存和磁盘容量大小　　D、计算机地址结构

4B　5D

课堂练习（判断正误）

1. 在页表中，页号不能大于块号。

课堂练习（判断正误）

1. 在页表中，页号不能大于块号。　×
2. 在请求分页存储管理中，为了提高内存利用率，允许用户使用不同大小的页面。

课堂练习（判断正误）

1. 在页表中，页号不能大于块号。　×
2. 在请求分页存储管理中，为了提高内存利用率，允许用户使用不同大小的页面。　×
3. 在请求分页存储管理中，当访问的页面不在内存时产生缺页中断，缺页中断是属于 I/O 中断。

课堂练习（判断正误）

1. 在页表中，页号不能大于块号。　×
2. 在请求分页存储管理中，为了提高内存利用率，允许用户使用不同大小的页面。　×
3. 在请求分页存储管理中，当访问的页面不在内存时产生缺页中断，缺页中断是属于 I/O 中断。　✓
4. 设主存容量为 1GB，辅存容量为 500GB，那么虚拟内存的最大容量是 501GB。

课堂练习（判断正误）

1. 在页表中，页号不能大于块号。　×
2. 在请求分页存储管理中，为了提高内存利用率，允许用户使用不同大小的页面。　×
3. 在请求分页存储管理中，当访问的页面不在内存时产生缺页中断，缺页中断是属于 I/O 中断。　✓
4. 设主存容量为 1GB，辅存容量为 500GB，那么虚拟内存的最大容量是 501GB。　×
5. 在请求分页存储管理中，若采用 FIFO 页面淘汰算法，则当分配的页面数增加时，缺页中断的次数减少。

课堂练习（判断正误）

1. 在页表中，页号不能大于块号。 ×
2. 在请求分页存储管理中，为了提高内存利用率，允许用户使用不同大小的页面。 ×
3. 在请求分页存储管理中，当访问的页面不在内存时产生缺页中断，缺页中断是属于 I/O 中断。 ✓
4. 设主存容量为 1GB，辅存容量为 500GB，那么虚拟内存的最大容量是 501GB。 ×
5. 在请求分页存储管理中，若采用 FIFO 页面淘汰算法，则当分配的页面数增加时，缺页中断的次数减少。 ×

课堂练习

- 在请求分页存储管理系统中，存取一次内存的时间是 8ns，访问一次快表的时间是 1ns。假设要访问的页面不在快表中时会出现缺页中断，缺页中断处理的时间是 20ns。一个作业最多可保留 3 个页面在内存。系统连续对作业的 2，4，5，2，7，6，4，5 页面的数据进行一次存取，求分别采用 FIFO 算法和最优页面置换算法时存取这些数据需要的总时间。

课堂练习

- 在请求分页存储管理系统中，存取一次内存的时间是 8ns，访问一次快表的时间是 1ns。假设要访问的页面不在快表中时会出现缺页中断，缺页中断处理的时间是 20ns。一个作业最多可保留 3 个页面在内存。系统连续对作业的 2，4，5，2，7，6，4，5 页面的数据进行一次存取，求分别采用 FIFO 算法和最优页面置换算法时存取这些数据需要的总时间。

FIFO：$(1 + 8\times2 + 20)\times7 + (1 + 8) = 268$ ns

OPT：$(1 + 8\times2 + 20)\times5 + (1 + 8)\times3 = 212$ ns

课堂练习

- 有一分页系统，物理块的大小为 1KB，其页表如下：

页号	块号	修改位	访问位	状态位
0	5	1	1	1
1	8	0	0	0
2	4	0	1	1
3	2	0	0	1
4	9	0	1	0

如果系统采用请求分页存储管理并采用局部替换策略，分配该进程的物理块数为 3，当访问的页不在内存时，采用改进型 Clock 置换算法选择淘汰页面，请问逻辑地址 1200 对应的物理地址是多少，给出计算过程。

课堂练习

- 逻辑地址 1200，页号为 1，页内位移为 176
- 查找页表状态位为 0，该页不在内存
- 采用改进型 Clock 置换算法，首先选择访问位和修改位为 0 的页面，即逻辑页 3，状态位为 1，块号为 2
- 物理地址：2*1024+176=2224

本章作业

THE END

School of Computer & Information Engineering

Henan University

Kaifeng, Henan Province

475001

China