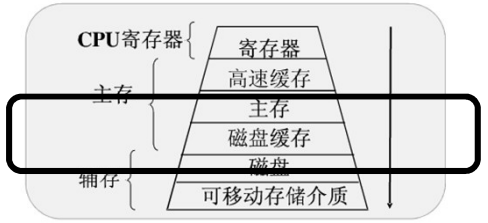
第五章　虚拟存储器



- 容量愈来愈大
- 访问数据的速度愈来愈慢
- 价格愈来愈便宜

连续分配➡离散分配 {（基本）分页；（基本）分段} 段页式

方便程序装入
提高内存利用率

虚拟存储器的引入

- 程序装入内存时可能会出现如下问题
  - 程序太大，要求的空间超出了内存总容量
  - 有大量作业要求运行，但内存不能容下所有作业

虚拟存储器的引入

- 常规存储器管理方式或运行的特征
  - 一次性
    - 要求作业全部装入内存才能运行
  - 驻留性
    - 程序装入内存后便一直驻留内存，直至运行结束

许多不用或暂时不用的程序占用了大量内存空间，而其他程序却无法装入！是否必要？

程序执行的局部性

- 程序执行多数情况下是顺序执行（少部分转移和过程调用）
- 过程调用的深度在大多数情况下都不超过5（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）
- 程序中存在许多循环结构（少数指令，多次执行）
- 程序中许多对数据结构处理局限于很小的范围内

局部性表现

- 时间
  - 程序中某指令一旦执行，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
  - 典型原因是因在程序中存在着大量循环操作
- 空间
  - 程序访问某个存储单元后，其附近的存储单元也将被访问
  - 程序在一时间段内所访问地址可能集中在一定的范围之内
  - 典型情况便是程序的顺序执行

局部性原理

1968年， Denning.P

程序在执行时将呈现出局部性规律，即在一较短时间内，程序的执行仅限于某个部分；相应地，它所访问的存储空间也局限于某个区域

基于局部性原理，一个作业在运行之前，没有必要全部装入内存，而仅将那些当前要运行的那部分页面或段，先装入内存便可启动运行，其余部分暂时留在磁盘上。

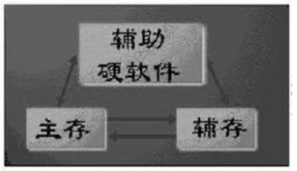
虚拟存储器的基本思想

程序、数据、堆栈的大小可以超过内存的大小，操作系统把程序当前使用的部分保留在内存，而把其它部分保存在磁盘上，并在需要时在内存和磁盘之间动态**交换**

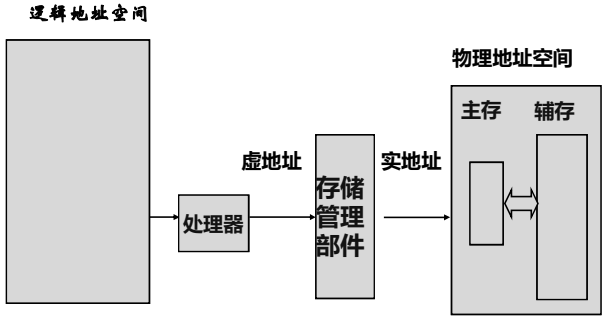
虚拟存储器定义

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，其运行速度接近于内存速度，而其成本却又接近于外存



虚拟存储器定义



虚拟存储器的容量

- 虚拟存储器的最大容量由计算机的地址结构确定

  如：若CPU的有效地址长度为32位，则程序可以寻址范围是0～$(2^{32})-1$ ，即虚存容量为 4GB。

- 虚拟存储器的容量与主存的实际大小没有直接的关系，而是由主存与辅存的容量之和确定

虚拟存储器的特征

- ❑多次性
  - ❖一个作业被分成多次调入内存运行
- ❑对换性
  - ❖允许在作业的运行过程中进行换进、换出
- ❑虚拟性
  - ❖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

资源转换技术:

以CPU时间和外存空间换取昂贵内存空间

虚拟存储器的实现方法

- ❑ 离散分配
  - ❑请求分页系统
  - ❑请求分段系统

虚拟存储器的实现方法——请求分页系统

- ❖分页系统的基础上增加了请求调页功能和页面置换功能
- ❖硬件支持
  - ➢请求分页的页表机制

    在基本分页的页表机制上增加请求分页的数据结构
  - ➢缺页中断机构

    用户程序访问页面未调入内存时产生缺页中断请求调页
  - ➢地址变换机构

    基础分页地址变换机构的调整
- ❖软件支持
  - ➢请求调页软件
  - ➢页面置换软件

虚拟存储器的实现方法——请求分段系统

- ❖ 硬件支持：
  - ➢请求分段的段表机制
  - ➢缺段中断机构
  - ➢地址变换机构。
- ❖ 实现请求调段和置换功能也需要得到OS的支持

实现虚拟存储器须解决的问题

- 主存辅存统一管理问题
- 逻辑地址到物理地址的转换问题
- 部分装入和部分对换问题有一定的系统开销

请求分页系统需要解决的问题

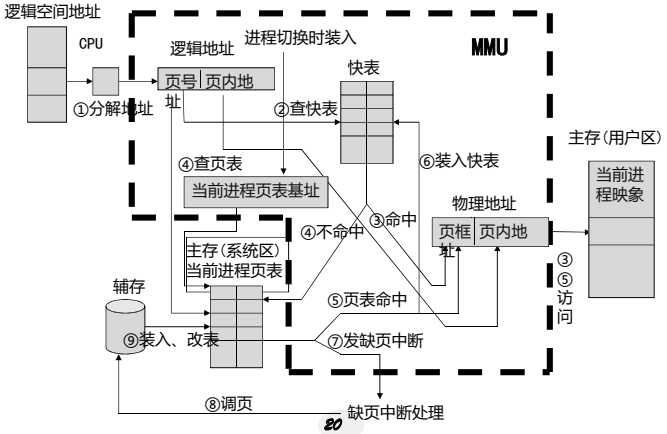
- 系统如何获知进程当前所需页面不在主存（页表机制）
- 发现缺页时如何把所缺页面调入主存（缺页中断机构）
- 主存中没有空闲的页框时，为了要接受一个新页，需要把老的一页淘汰出去，根据什么策略选择欲淘汰的页面（置换算法）

页表机制

页号	物理块号	状态位P	访问位A	修改位M	外存地址

- 状态位P（中断位、存在位）指示页面在内存还是在外如果是0，表示该页面不在内存中，会引起一个缺页中断。
- 访问位A 记录本页在一段时间内被访问的次数或记录本页在最近多长时间未被访问
- 修改位M 表示该页在内存中是否被修改过
- 外存地址 该页在外存上的地址，通常是物理块号

页表机制



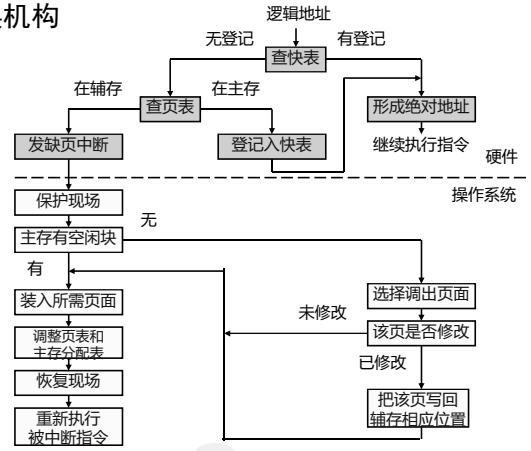
缺页中断机构

- 在请求分页系统中，每当所要访问的页面不在内存时，便产生一缺页中断
- 相应的中断处理程序把控制转向缺页中断子程序
- 中断子程序把所缺页面装入主存
- 处理机重新执行缺页时打断的指令
- 形成物理地址

地址变换机构

- 如果在快表中未找到该页的页表项，则应再到内存中去查找页表，再从找到的页表项中的状态位P，该页是否调入内存。
  - 该页已经调入内存，这是应将此页的页表项写入快表，当快表已满时，应先调出按某种算法所确定的页的页表项，然后再写入该页的页表项。
  - 该页尚未调入内存，这时便应产生缺页中断，请求OS从外存中把该页调入内存。

地址变换机构



请求分页中的内存分配

最小物理块数的确定

- 指保证进程正常运行所需的最小物理块数。当系统分配的物理块数少于此值时，进程将无法运行
- 进程应获得的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
- 对于单地址指令且采用直接寻址方式的机器，则所需最少2个物理块：一块存放指令页面，另一块则存放数据页面
- 允许间接寻址的机器，至少要求有3个物理块
- 对于长度是两个或多于两个字节指令的机器，其指令本身可能跨两个页面，且源和目标地址所涉及的区域也可能跨两个页面，至少需要6个物理块

物理块分配算法——平均分配算法

将系统中所有可供分配的物理块，平均分配给各个进程

例如，当系统中有100个物理块，有5个进程在运行时，每个进程可分得20个物理块。

问题：未考虑到各进程本身的大小

例如：进程A大小为200页，只分配20个块会有高**缺页率**

进程B有10页，却有10个物理块闲置未用

物理块分配算法——按比例分配算法

根据进程的大小按比例分配物理块

如果系统中共有n个进程，每个进程的页面数为S_i，则系统中各进程页面数的总和为：

$$S=\sum_{i=1}^{n}S_i$$

设系统中可用物理块总数为m，每个进程所分物理块数为：

$$b_i=\frac{S_i}{S}\times m$$

b应该取整，它必须大于最小物理块数

物理块分配算法——考虑优先权的分配算法

- 为重要的、紧迫的用户程序分配较多的内存空间
- 通常采取的方法
  - 把内存中可供分配的所有物理块分成两部分
  - 一部分按比例地分配给各进程
  - 另一部分根据进程优先权增加份额
- 实时控制系统，可按完全按优先权为各进程分配其物理块的

页面调入策略

- 系统应在何时调入所需页面？
- 系统应从何处调入所需页面？
- 如何调入所需页面？

页面调入策略——调入时机

- 预调页策略 （50%）
  - 以预测为基础的预调页策略
  - 预先调入预计在不久之后便会被访问的页面
- 请求调页策略
  - 访问程序和数据时发现缺页提出请求
  - OS将需页面调入内存
  - 虚拟存储中大多采用策略

页面调入策略——调入地点

- 请求分页系统外存：
- **1. 用于存放文件的文件区**
- **2. 用于存放对换页面的对换区**

- 对换区是采用连续分配方式，文件区采用离散分配方式
- 对换区的磁盘I/O速度比文件区高

页面调入策略——调入地点

- 系统拥有足够的对换区空间
  - 从对换区调入所需页面，以提高调页速度
  - 在进程运行前将与有关文件从文件区拷贝到对换区
- 系统缺少足够的对换区空间
  - 直接从文件区调入不会被修改的文件
  - 将可能被修改的部分从对换区调入
- UNIX方式
  - 未运行页面从文件区调入
  - 运行被换出的页面从对换区调入
  - 页面共享被其它进程调入内存页无须再从对换区调入

页面调入策略——调入过程

① 缺页中断

② 保留CPU环境

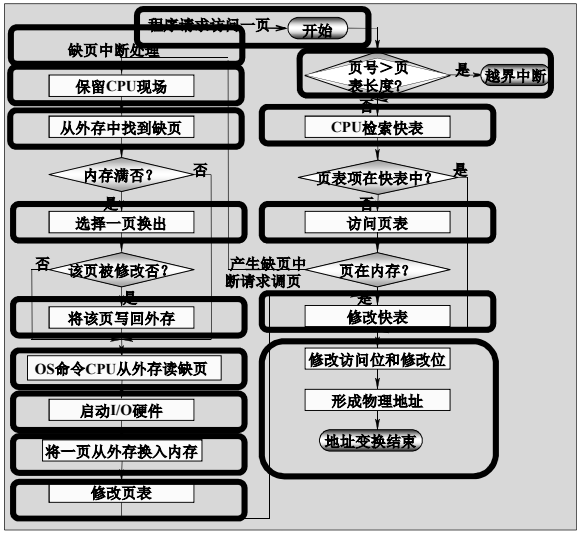
③ 缺页中断处理程序

④ 查找页表得到外存物理块地址

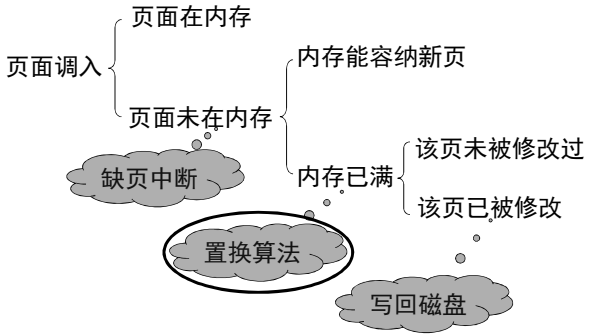
a. 若内存能容纳新页
   a. 启动磁盘I/O将调入缺页
   b. 修改页表

a. 若内存已满，按照置换算法选出一页准备换出
b. 缺页调入内存，修改页表项，状态位置 “1”
c. 将页表项写入快表中
d. 形成所访问数据的物理地址，访问内存数据

5.2 地址变换过程



页面调入策略——调入过程

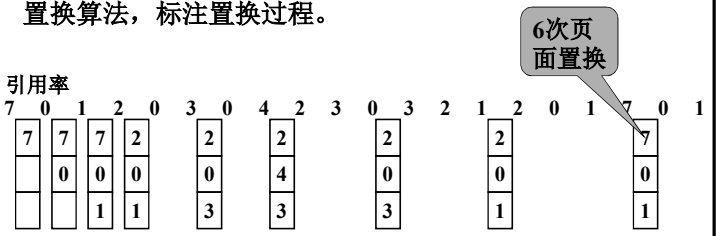


最佳置换算法

- 1966年由Belady提出的一种理论上的算法
- 所选择的被淘汰页面，将是以后永不使用的，或是在最长(未来)时间内不再被访问的页面
- 保证获得最低的缺页率

最佳置换算法

例：假定系统为某进程分配了三个物理块， 并考虑有以下的页面号引用串：7，0，1，2，0，3，0，4，2，3，0，3，2，1，2，0，1，7，0，1进程运行时，根据最佳置换算法，标注置换过程。

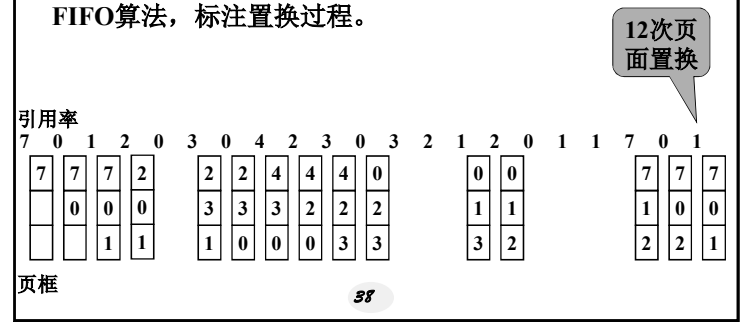


先进先出置换算法

- 思想：淘汰最先进入内存的页面
- 选择在内存中的驻留时间最久的页面予以淘汰
- 算法实现
  - 把进程已调入页面，按先后次序链接成一个队列
  - 设置替换指针指向最老页面
- FIFO不能保证经常被访问的含有全局变量、常用函数、例程等的页面不被淘汰

先进先出置换算法

例：假定系统为某进程分配了三个物理块， 并考虑有以下的页面号引用串：7，0，1，2，0，3，0，4，2，3，0，3，2，1，2，0，1，7，0，1进程运行时， 根据FIFO算法，标注置换过程。



先进先出置换算法

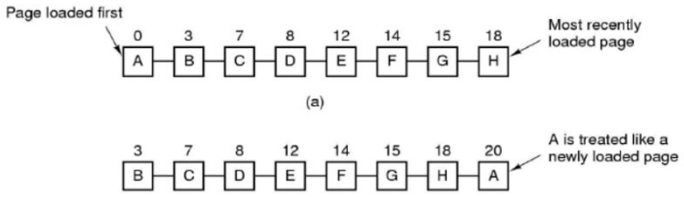
例子：系统给某进程分配 m个页框，初始为空

页面访问顺序为

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

采用FIFO算法，计算当 m=3 和 m=4 时的缺页中断次数

第二次机会页面替换算法



第二次机会页面替换算法

结合FIFO与页表的访问位：

- 检查FIFO中的队首页面(最早进入主存的页面),如果它的访问位是0,这个页面既老又没有用，选择该页面
- 如果访问位是1,说明它进入主存较早，但最近仍在使用。把它的访问位清0,并把这个页面移到队尾，把它看作是一个新调入的页。
- 算法含义：最先进入主存的页面，如果最近还在被使用的话,仍有机会作为像一个新调入页面一样留在主存中

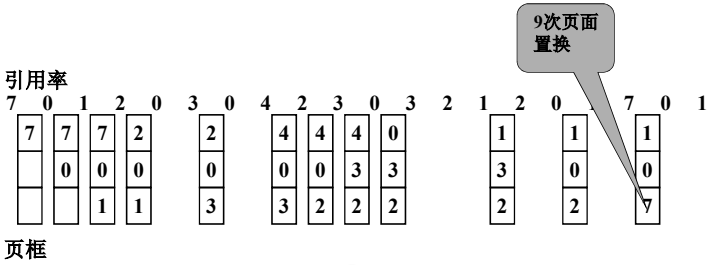
最近最久未使用(LRU)置换算法

LRU(Least Recently Used)

- FIFO置换算法之所以性能较差，是因为它所依据的条件是各个页面调入内存的时间，而页面调入的先后并不能反映页面的使用情况。
- LRU置换算法根据页面调入内存后的使用情况,利用“**最近的过去**”作为“**最近的将来**”的近似，选择**最近最久未使用**的页面予以淘汰

最近最久未使用(LRU)置换算法

例：假定系统为某进程分配了三个物理块， 并考虑有以下的页面号引用串：7，0，1，2，0，3，0，4，2，3，0，3，2，1，2，0，1，7，0，1进程运行时， 根据LRU算法，标注置换过程。



例子：

- 系统给某进程分配3个页框(固定分配策略)，初始为空
- 进程执行时，页面访问顺序为：
  2 3 2 1 5 2 4 5 3 2 5 2

要求：

计算应用FIFO、LRU、OPT算法时的缺页次数

最近最久未使用(LRU)置换算法

问题：

- 一个进程在内存中的各个页面各有多久时间未被进程访问
- 如何快速地知道哪一页最近最久未使用的页面。

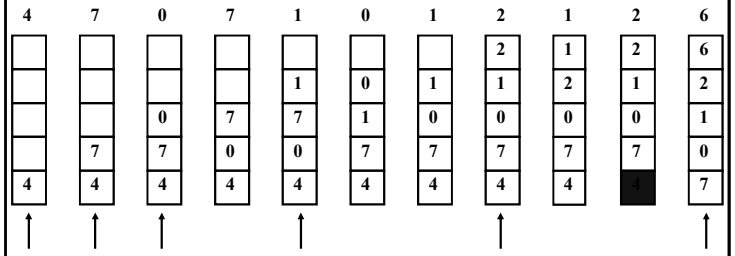
支持软件和硬件：

- 栈：访问某页时压入“栈顶”，“栈底”换出
- 矩阵
- 移位寄存器： 定时右移
- 时间戳

最近最久未使用(LRU)置换算法——硬件支持

❑栈

❖被进程访问页面页号从栈中移出，再压入栈顶



最近最久未使用(LRU)置换算法——移位寄存器

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

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

- 访问时将R_{n-1}位置成1
- 定时信号每隔一时间间隔右移一位
- 具有最小数值的寄存器所对应的页面，就是最近最久未使用的页面

LRU移位寄存器

t0

页	R_7	R_6	R_5	R_4	R_3	R_2	R_1	R_0
1	0	1	0	1	0	0	1	0
2	1	0	1	0	1	1	0	0

t1

页	R_7	R_6	R_5	R_4	R_3	R_2	R_1	R_0	
1	0	0	1	0	1	0	0	1	0
2	0	1	0	1	0	1	1	0	0

t2

页	R_7	R_6	R_5	R_4	R_3	R_2	R_1	R_0	
1	0	0	0	1	0	1	0	0	1
2	0	0	1	0	1	0	1	1	0

访问1

页	R_7	R_6	R_5	R_4	R_3	R_2	R_1	R_0
1	1	0	0	1	0	1	0	0
2	0	0	1	0	1	0	1	1

t3

页	R_7	R_6	R_5	R_4	R_3	R_2	R_1	R_0	
1	0	1	0	0	1	0	1	0	0
2	0	0	0	1	0	1	0	1	1

最近最久未使用(LRU)置换算法——移位寄存器

实页 \ R	R_7	R_6	R_5	R_4	R_3	R_2	R_1	R_0
1	0	1	0	1	0	0	1	0
2	1	0	1	0	1	1	0	0
3	0	0	0	0	0	1	0	0
4	0	1	1	0	1	0	1	1
5	1	1	0	1	0	1	1	0
6	0	0	1	0	1	0	1	1
7	0	0	0	0	0	1	1	1
8	0	1	1	0	1	1	0	1

最少使用置换算法（LFU）

- 每页设置访问计数器
- 每当页面被访问时，该页面的访问计数器加1；
- 发生缺页中断，淘汰计数值最小页面，将所有计数清零

实现：

- 每个页面设立移位寄存器
- 被访问时左边最高位置1
- 定期右移并且最高位补0
- 寄存器各位之和最小的是最少使用页面

最少使用置换算法（LFU）

页面淘汰标准：最近时期、使用最少

硬件支持（寄存器）

缺点：页面访问频率不准确

特点：内存中每个页面设置移位寄存器，记录该页面被访的频率

LFU&LRU：

- LRU比较寄存器内数字大小
- LFU比较寄存器内R_1 +…+ R_n

备注：LFU页面访问图与LRU页面访问图完全一样

最少使用置换算法（LFU）

实页 \ R	R_7	R_6	R_5	R_4	R_3	R_2	R_1	R_0
1	0	1	0	1	0	0	1	0
2	1	0	1	0	1	1	0	0
3	0	0	0	0	0	1	0	0
4	0	1	1	0	1	0	1	1
5	1	1	0	1	0	1	1	0
6	0	0	1	0	1	0	1	1
7	0	0	0	0	0	1	1	1
8	0	1	1	0	1	1	0	1

Clock置换算法——简单时钟置换

- 为每页设置一位访问位
- 将内存中的所有页面链成一个循环队列
- 某页被访问时，其访问位被置1
- 淘汰访问位为0页面

页号	物理块号	状态位P	访问字段A	修改位M	外存地址

Clock置换算法——简单时钟置换

页号	物理块号	状态位P	访问字段A	修改位M	外存地址

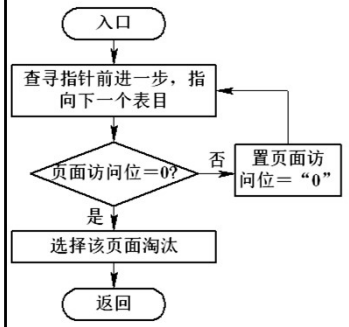
- 状态位(存在位)P：指示该页是否调入内存，程序访问时参考
- 访问字段A：记录本页在一段时间内被访问的次数或最近已有多长时间未被访问，置换算法选择换出页面时参考
- 修改位M：在调入内存后是否被修改过。若未被修改，置换时不须写回外存，减少系统的开销和启动磁盘的次数；若被修改，则必须重写到外存，保证外存中保留的是最新副本
- 外存地址：指出该页在外存上的地址，通常是物理块号，供调入该页时使用

Clock置换算法——简单时钟置换

（近似的LRU算法）

- 为每页设置一位访问位，将内存中的所有页面都通过链接指针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
  - 检查页的访问位，0换出，1重新置0暂不换出
  - 按FIFO检查下一个页面
  - 检查到最后一个页面，则再返回队首检查
- 循环地检查各页面的访问情况，称为CLOCK算法

Clock置换算法——简单时钟置换

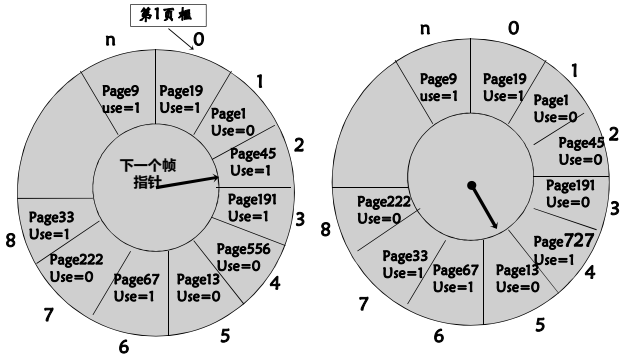


块号	页号	访问位	指针
0			
1			
2	4	0	
3			
4	2	1	
5			
6	5	0	
7	1	1	

替换指针

注意：是循环队列！！

Clock置换算法——简单时钟置换



一个页替换前的缓冲区状态　　下一页替换后的缓冲区状态

Clock置换算法——改进时钟置换

- 页面换出时，如果已被修改，须重新写到磁盘
- 该页未被修改过，则不必将它拷回磁盘

同时满足两条件的页面作为首选淘汰的页

置换未使用过的页，最近未用算法

Not Recently Used (NRU)

页号	物理块号	状态位P	访问字段A	修改位M	外存地址

Clock置换算法——改进时钟置换

- 考虑使用情况和置换代价
- 由访问位A和修改位M组合：
  - 1类(A=0, M=0)：表示该页最近既未被访问，又未被修改，是最佳淘汰页
  - 2类(A=0, M=1)：表示该页最近未被访问，但已被修改，并不是很好的淘汰页
  - 3类(A=1, M=0)：最近已被访问，但未被修改，该页有可能再被访问
  - 4类(A=1, M=1)：最近已被访问且被修改，该页可能再被访问

Clock置换算法——改进时钟置换

- 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
- 如果第一步失败，即查找一周后未遇到第一类页面， 则开始第二轮扫描，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A都置0
- 如果第二步也失败，亦即未找到第二类页面，则将指针返回到开始的位置，并将所有的访问位A复0。 然后重复第一步，如果仍失败，必要时再重复第二步，此时就一定能找到被淘汰的页

Clock置换算法——改进时钟置换

页号	块号	A	M	
0	5	0	0	← 最佳淘汰页
1	3	0	1	← 次佳淘汰页
2	10	~~1~~ 0	0	
3	11	~~1~~ 0	1	
4	14	~~1~~ 0	1	

页面缓冲算法

- 页面缓冲算法(Page Buffering Algorithm)
- 对FIFO算法的发展
- 通过被置换页面的缓冲，找回刚被置换的页面
- 页面分配采用可变分配和局部置换的方式。
- 被置换页面的选择和处理：用FIFO算法选择被置换页
- 被置换的页面放入两个链表之一:
  - 如果页面未被修改，就将其归入到空闲页面链表的末尾
  - 否则将其归入到已修改页面链表

页面缓冲算法

- 需要调入新的物理页面时，新页面内容读入到空闲页面链表的第一项所指的页面，将第一项删除
- 空闲页面和已修改页面停留在内存中一段时间，如果这些页面被再次访问，只需较小开销，而被访问的页面可以返还作为进程的内存页
- 已修改页面达到一定数目后，再将它们一起调出到外存，然后将它们归入空闲页面链表，大大减少I/O操作的次数

缺页分析

- 请求分页式虚拟存储器系统在正常运行情况下能有效地减少内存碎片，提高处理机的利用率和吞吐量
- 如果在系统中运行的进程太多，进程在运行中会**频繁地发生缺页**情况，会对系统的性能产生很大的影响

缺页分析

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

抖动定义

- 颠簸（Thrashing）

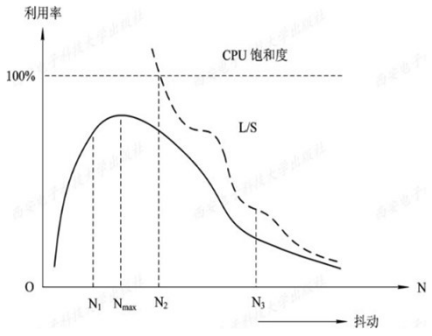
  在虚存中，页面在内存与外存之间频繁调度，以至于调度页面所需时间比进程实际运行的时间还多，此时系统效率急剧下降，甚至导致系统崩溃

“抖动”的产生

- 同时在系统中运行的进程太多
- 分配给每一个进程的物理块太少
- 进程正常运行的基本要求不能满足，运行时频繁缺页
- 系统中排队等待页面调进/调出的进程数目增加
- 对磁盘的有效访问时间也随之急剧增加
- 每个进程的大部分时间都用于页面的换进/换出
- 进程几乎不能再去做任何有效的工作
- 处理机的利用率急剧下降并趋于0

多道程序度与“抖动”

- 虚拟存储器系统逻辑上扩大内存
- 装入部分程序和数据可开始运行
- 系统中能运行更多的进程
- 增加多道程序度



工作集定义

指在某段时间间隔Δ里，进程实际所要访问页面的集合

思想：

- 为减少缺页，应将程序的全部工作集装入内存
- 无法事先预知程序将访问哪些页面
- 用程序的过去某段时间内的行为作为程序在将来某段时间内行为的近似

工作集例示

引用页序列	窗口大小 3	4	5
24	24	24	24
15	15 24	15 24	15 24
18	18 15 24	18 15 24	18 15 24
23	23 18 15	23 18 15 24	23 18 15 24
24	24 23 18	—	—
17	17 24 23	17 24 23 18	17 24 23 18 15
18	18 17 24	—	—
24	—	—	—
18	—	—	—
17	—	—	—
17	—	—	—
15	15 17 18	15 17 18 24	—
24	24 15 17	—	—
17	—	—	—
24	—	—	—
18	18 24 17	—	—

“抖动”的预防

1. 采取局部置换策略

- 在页面分配和置换策略中，如果采取的是可变分配方式，采取局部置换策略。
- 某进程发生缺页时，只能在分配给自己的内存空间内进行置换，不允许从其它进程去获得新的物理块，把进程抖动所造成的影响限制在较小的范围内。

“抖动”的预防

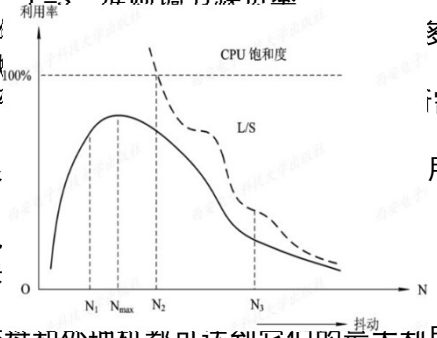
2. 把工作集算法融入到处理机调度中

- 当调度程序发现处理机利用率低下时，试图从外存调入一个新作业改善处理机的利用率
- 调度中融入了工作集算法，调入作业之前检查进程在内存的驻留页面足够多时调入
- 避免新作业调入导致缺页率增加
- 为缺页率高的进程增加物理块，不调入新作业

“抖动”的预防

3. 利用“L=S”准则调节缺页率

- 1980
- L是缺
- S是平
- L>>S
  - 很
- L<S
  - 说
  - 缺
- L=S
  - 磁盘和处理机都可达到它们的最大利用率



多道程序度

需的时间

用

“抖动”的预防

4. 选择暂停的进程

- 当多道程序度偏高影响处理机利用率
- 为防止 “抖动”，减少多道程序的数目
- 基于某种原则选择暂停某些当前活动的进程
- 将它们调出到磁盘上，以便把腾出的内存空间分配给缺页率发生偏高的进程
- 系统通常优先选择暂停优先级最低的进程调出

请求分页虚拟存储管理的几个设计问题

1) 页面大小

. 从页表大小考虑

. 从主存利用率考虑

. 从读写一个页面所需时间考虑

. 最佳页面尺寸页面

请求分页虚拟存储管理的几个设计问题

2) 页面交换区

替换算法要挑选页面淘汰出主存，但被淘汰出去的页面可能很快使用，又要被重新装入主存。操作系统必须保存被淘汰的页面,例如UNIX/Linux使用交换区临时保存页面，系统初始化时,保留一定盘空间作交换区。

请求分页虚拟存储管理的几个设计问题

3) 写时复制

写时复制(copy-on-write)是存储管理节省物理主存(页框)的一种页面级优化技术，已被UNIX和Windows等采用，能减少主存页面内容的复制操作，减少相同内容页面在主存的副本数目。

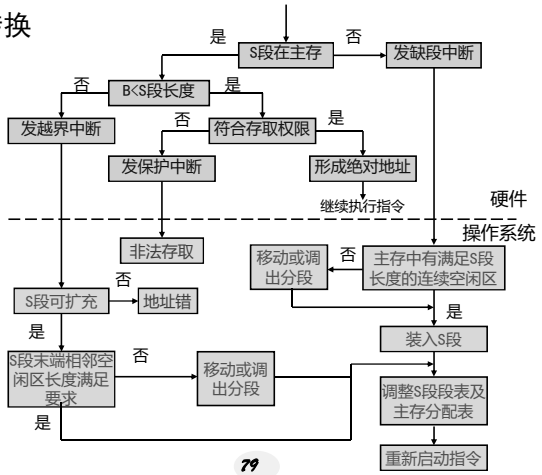
请求分段中的硬件支持

1. 段表机制

- 存取方式 用于标识本分段存取属性是只执行、只读还是允许读/写
- 存在位P 用于指示该段是否已调入内存
- 访问字段A 用于记录本页在一段时间内被访问的次数，或记录本页在最近多长时间未被访问
- 修改位M 表示该段在调入内存后是否被修改过
- 外存地址 本段在外存上的地址，盘块块号
- 增补位 本段在运行过程中是否做过动态增长

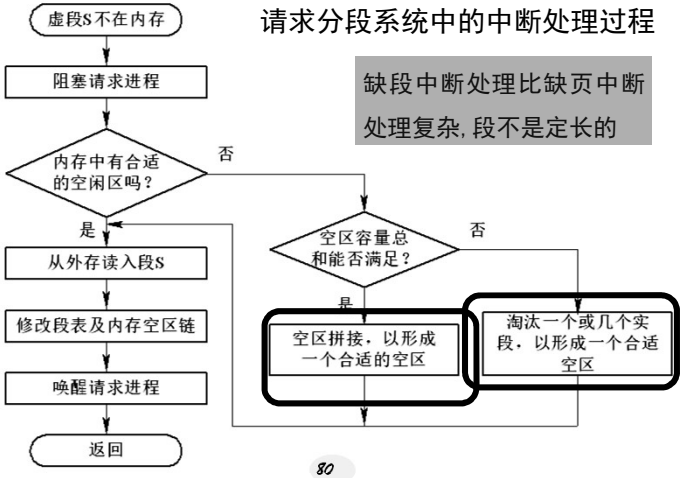
段名	段长	段的基址	存取方式	访问字段A	修改位M	存在位P	增补位	外存始址

地址转换

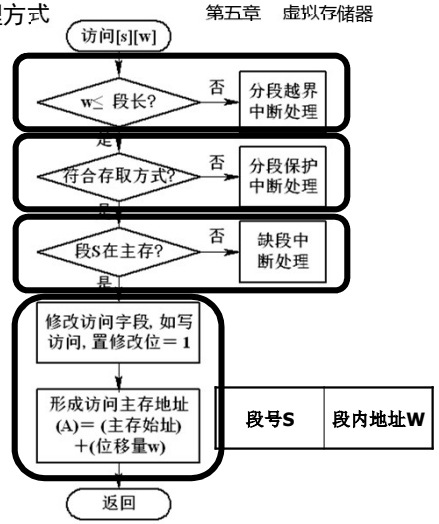


请求分段系统中的中断处理过程

缺段中断处理比缺页中断处理复杂，段不是定长的



请求分段系统的地址变换过程



分段的共享与保护

1. 共享段表

为了实现分段共享，可在系统中配置一张共享段表

所有各共享段都在共享段表中占有一表项

(1) 共享进程计数count

记录有多少个进程需要共享该分段

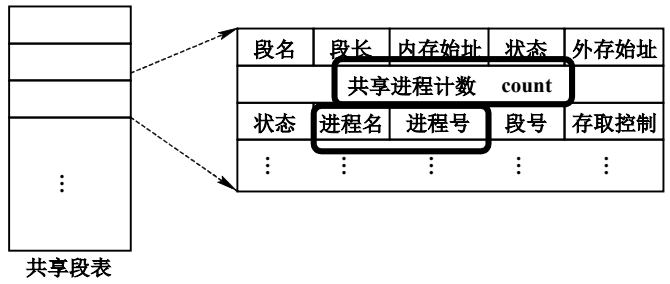
(2) 存取控制字段

给不同的进程以不同的权限

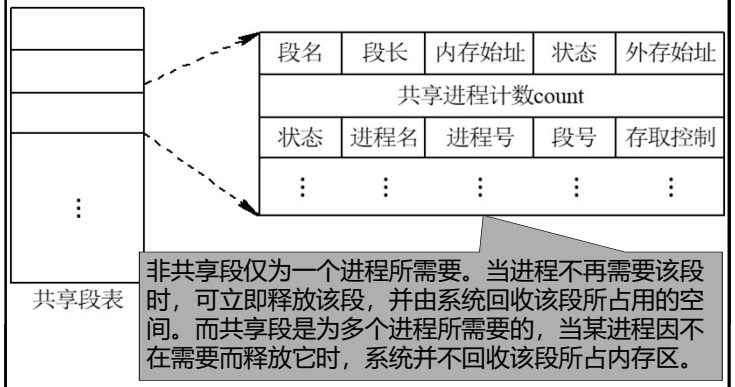
(3) 段号

不同的进程可以各用不同的段号去共享段

分段的共享与保护

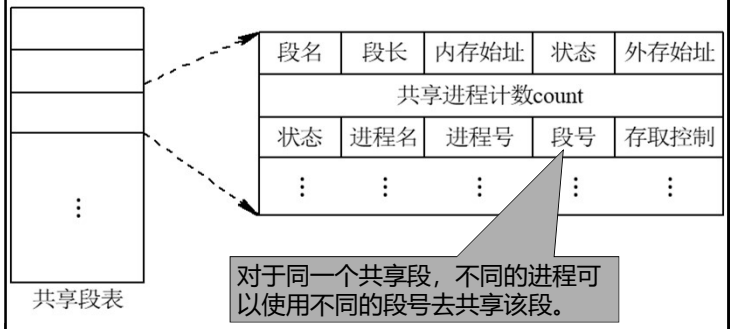


分段的共享与保护

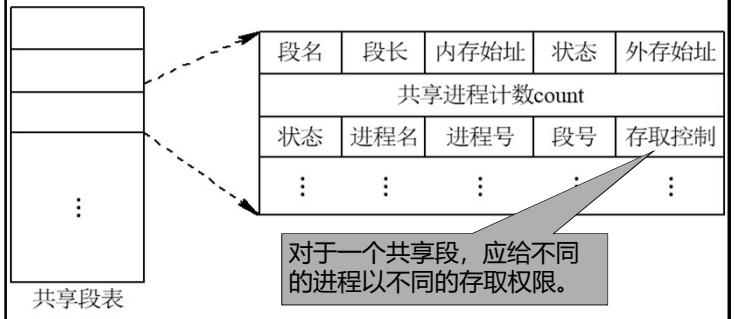


非共享段仅为一个进程所需要。当进程不再需要该段时，可立即释放该段，并由系统回收该段所占用的空间。而共享段是为多个进程所需要的，当某进程因不在需要而释放它时，系统并不回收该段所占内存区。

分段的共享与保护



分段的共享与保护



分段的共享与保护

2. 共享段的分配

- 第一个请求使用该共享段的进程，系统分配物理区，把共享段调入该区，填入请求进程的段表项，在共享段表中增加一表项，填写有关数据，count置为1；
- 其它进程需要调用该共享段时，无须分配内存，在调用进程的段表中，增加一表项，填写该共享段的物理地址；在共享段的段表中，填上调用进程的进程名、存取控制等，再执行count：=count+1操作，以表明有两个进程共享该段

分段的共享与保护

3. 共享段的回收

- 撤消进程段表中共享段所对应的表项
- 执行count：=count-1操作
- 若结果为0，则须由系统回收该共享段的物理内存，以及取消在共享段表中该段所对应的表项，表明此时已没有进程使用该段；
- 若减1结果不为0，则取消调用者进程在共享段表中的有关记录

分段的共享与保护

4. 分段保护

1) 越界检查

2) 存取控制检查

- 只读
- 只执行
- 读/写

3) 环保护机构

- 低编号的环具有高优先权，操作系统位于最核心环
- 一个程序可以访问驻留在相同环或较低特权环中的数据
- 一个程序可以调用驻留在相同环或较高特权环中的服务

分段的共享与保护

4. 分段保护——越界检查

- 段表寄存器中有段表长度信息
- 段表中为每个段设置有段长字段
- 存储访问时，将逻辑地址空间的段号与段表长度进行比较，如果段号等于或大于段表长度，将发出地址越界中断信号；
- 段内地址是否等于或大于段长，若大于段长，将产生地址越界中断信号

分段的共享与保护

4. 分段保护——存取控制检查

段表表项设置 “存取控制”字段规定对该段的访问方式

（1）只读:

只允许程序对该段中的程序或数据进行读访问

（2）只执行:

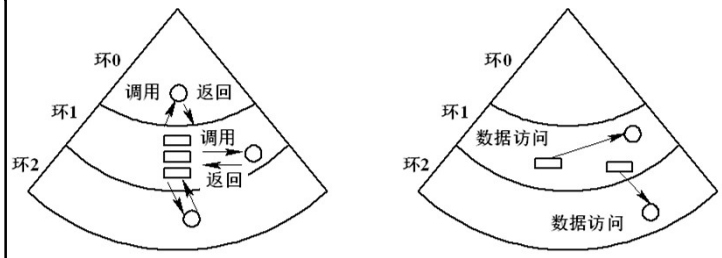
只允许程序调用该段去执行，但不准读写操作

（3）读/写。

允许程序对该段进行读写访问

分段的共享与保护

4. 分段保护——环保护机构



(a) 程序间的控制传输 (b) 数据访问

分段的共享与保护

4. 分段保护——环保护机构

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；

➢一般的应用程序，则被安排在外环上

环系统中，程序的访问和调用规则：

（1）一个程序可以访问驻留在相同环或较低特权环中的数据

（2）一个程序可以调用驻留在相同环或较高特权环中的服务

请求段页式虚拟存储管理

1、虚地址以程序的逻辑结构划分成段

（段页式存储管理的段式特征）

2、实地址划分成位置固定、大小相等的页框

（段页式存储管理的页式特征）

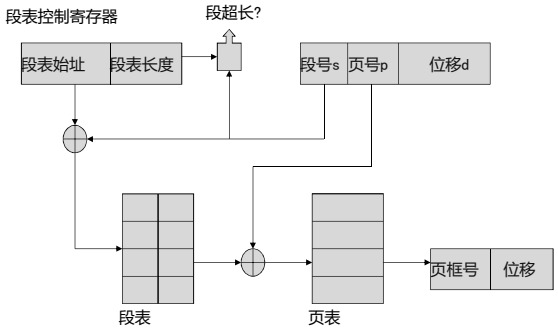
3、将每一段的线性地址空间划分成与页框大小相等的页面，

于是形成了段页式存储管理的特征。

4、逻辑地址形式为：

段号(s)	段内页号 (p)	页内位移(d)

请求段页式虚拟存储管理



请求段页式虚拟存储管理