5 虚拟存储器

5.1 虚拟存储器概述

虚拟存储管理

- 作业全部装入内存运行导致
  - 超过内存容量的作业无法运行
  - 内存限制，大量作业在外存等待无法运行
- 增加物理内存
- 逻辑上扩充内存——虚拟存储管理

虚拟存储器的引入

- **1. 常规存储器管理方式的特征**
  - **(1)一次性**
    - 一次性装入内存
    - 有些数据和程序段未必用到
  - **(2) 驻留性**
    - 作业装入后，一直驻留内存，直至结束
    - 即使阻塞也在内存；用过后不再使用的程序和数据仍然驻留内存
    - 内存使用浪费

虚拟存储器的引入

- **2.** 局部性原理，
  - **1968**年， **Denning.P**就指出：
  - **(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， 但经研究看出，过程调用的深度在大多数情况下都不超过**5**。
  - **(3)** 程序中存在许多循环结构， 这些虽然只由少数指令构成， 但是它们将多次执行。
  - **(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。

虚拟存储器的引入

- **2.** 局部性原理
  - **(1)** 时间局限性。　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)** 空间局限性。　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。

虚拟存储器的引入

- **3. 虚拟存储器定义**
- 基本原理
  - 基于局部性原理，不是把作业或进程的程序和数据全部装入内存后才开始执行。只装入核心和反复调用执行的部分；其它部分在执行过程中动态调入。
  - 执行过程中，如内存不足，可将内存中暂时不用的部分调至辅存。
  - 大的作业可以在较小的内存空间运行；内存中可以装入更多的并发进程。
- 概念
  - 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。
  - 其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，而每位的成本却又接近于外存。

虚拟存储器的特征

- **1** 多次性
  - 一个作业分多次调入内存运行
  - 为基本特征
- **2** 对换性
  - 允许作业运行过程过换进换出
- **3** 虚拟性
  - 从逻辑上扩充内存容量，用户看到的内存容量大于实际物理内存容量。
  - 为虚拟存储的目标

虚拟存储器的实现方法

- **1请求分页系统**
  - 硬件支持
    页表机制, 缺页中断机构, 地址变换机构
  - 请求调页和页面置换软件

- **2 请求分段系统**
  - 硬件支持
    段表机制, 缺段中断机构,地址变换机构
  - 请求调段和段置换软件

5.2 请求分页存储管理方式

5.2 请求分页存储管理方式

- 基本原理
  - 不是把作业或进程的程序和数据全部装入内存后才开始执行。
  - 只装入核心和反复调用执行的部分；其它部分在执行过程中动态调入。
- 主要问题
  - 地址变换同静态页式管理
  - 虚页不在内存？
    - 找到对应页面
  - 从外存调入内存，放在何处
    - 可能需要淘汰内存中的页，发生页面置换。

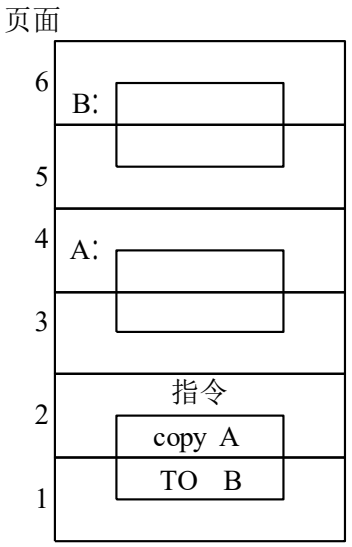
5.2.1 请求分页中的硬件支持

- **1. 页表机制**
  - 页号
  - 物理块号
  - 状态位：该页是否在内存
  - 访问字段：访问情况，如次数，时间等信息
  - 修改位：是否被修改过
  - 外存地址


逻辑地址空间
页表
驻留位为0
16位的逻辑地址，逻辑地址空间64K。物理内存只有32K。页面大小为4K。
MOV REG, [8192]
MOV REG, [32780]
驻留位为1
物理地址空间
物理页面
15 60K-64K X
14 56K-60K X
13 52K-56K X
12 48K-52K X
11 44K-48K 7
10 40K-44K X
9 36K-40K 5
8 32K-36K X
7 28K-32K X
6 24K-28K X
5 20K-24K 3
4 16K-20K 4
3 12K-16K 0
2 8K-12K 6
1 4K-8K 1
0 0K-4K 2
28K-32K 7
24K-28K 6
20K-24K 5
16K-20K 4
12K-16K 3
8K-12K 2
4K-8K 1
0K-4K 0

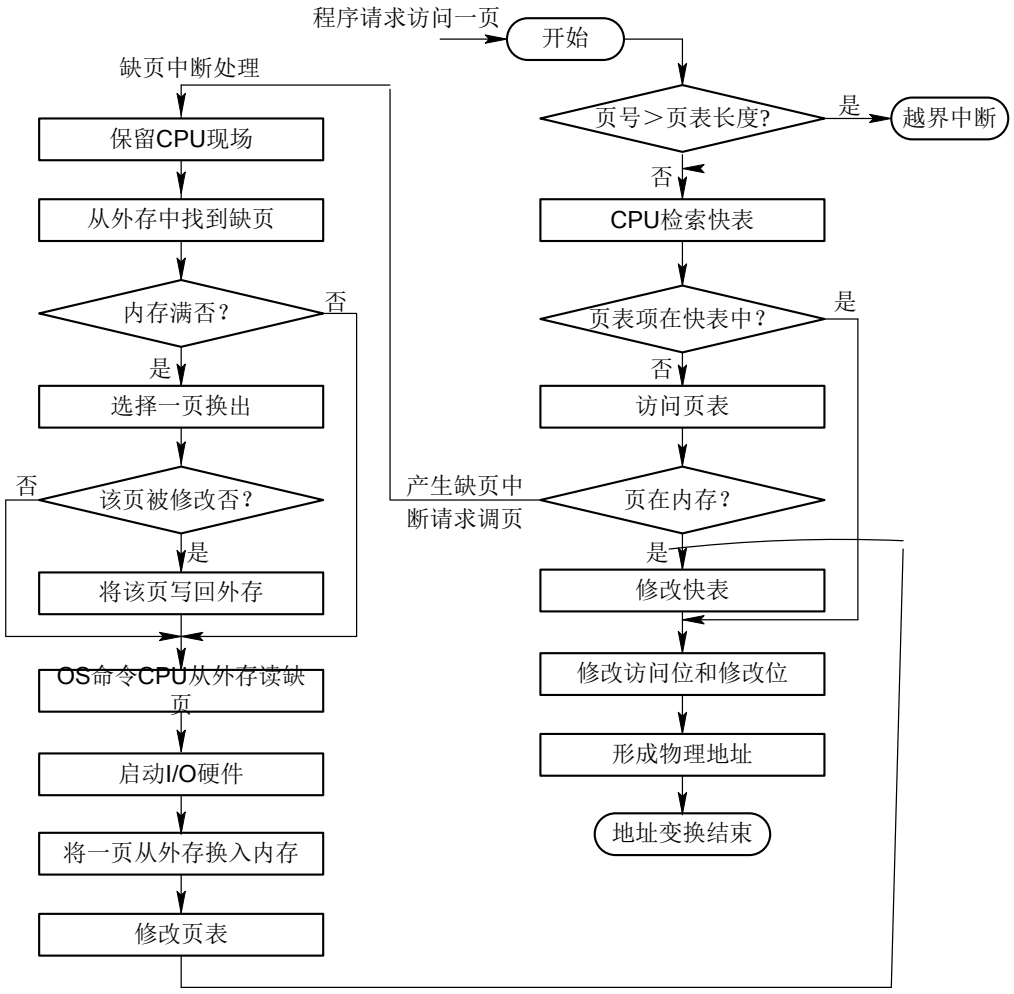
5.2.1 请求分页中的硬件支持

- **2 缺页中断机构**
  - 访问的页不在内存时，发生缺页中断
  - 在指令执行期间产生和处理中断信号
  - 一条指令执行期间，可能产生多次缺页中断
    - 共6次，指令跨页，数据A，B跨两页。

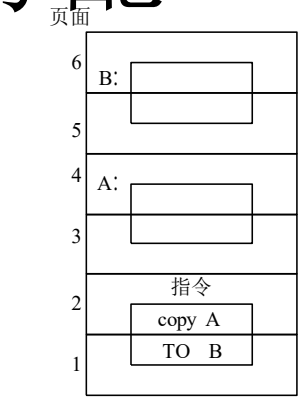


5.2.1 请求分页中的硬件支持

- **3. 地址变换机构**
  - 产生和处理缺页中断
  - 从内存中置换页
  - 写回



5.2.2 请求分页系统中的内存分配



- 为进程分配内存时的问题
- **1. 最小物理块数的确定**
  - 保证进程正常运行所需的最小物理块数。
  - 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。
  - 单地址指令且采用直接寻址方式（地址在指令中，单字节指令），则所需的最少物理块数为**2**：存放指令的页面、存放数据的页面。
  - 间接寻址：至少**3**个物理块。
  - 指令长度可能是两个或多于两个字节，因而其指令本身有可能跨两个页面，且源地址和目标地址所涉及的区域也都可能跨两个页面。共**6**个页面。

5.2.2 请求分页系统中的内存分配

- **2. 物理块的分配策略**
  - **1) 固定分配局部置换**
    - 为进程分配固定数目的物理块，运行期间不变
    - 缺页置换只能在该进程的页面中选取
    - 物理块数确定困难
  - **2) 可变分配全局置换**
    - 运行中间可以变化。易于实现。缺页获得新物理块，需要置换考虑所有物理块情况进行置换；
  - **3) 可变分配局部置换**
    - 只在进程中换页；如缺页频繁，增加物理块

5.2.2 请求分页系统中的内存分配

- **3. 物理块分配算法**
  - **1) 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。
  - **2) 按比例分配算法**
    - 根据进程的大小按比例分配物理块
    - 系统中共有**n**个进程，每个进程的页面数为**Si**，系统中各进程页面数的总和为**S**；
    - 假定系统中可用的物理块总数为**m**，则每个进程所能分到的物理块数为**bi**：

$$b_i = \frac{S_i}{S} \times m \qquad S = \sum_{i=1}^{n} S_i$$

  - **3) 考虑优先权的分配算法**

5.2.3 页面调入策略

- **1. 何时调入页面**
  - 进程运行时将所缺页调入内存的时机
  - **1)** 预调页策略
    - 预测不久以后被访问的页调入内存
    - 主要用于首次调入
  - **2)** 请求调页策略
    - 运行中，访问的页不在内存时，由**OS**调入
    - 容易实现，但每次调入一页，开销大

5.2.3 页面调入策略

- **2** 何处调入页面
  - 外存组织
    - 请求分页系统中的外存分为两部分：用于存放文件的文件区和用于存放对换页面的对换区。
    - 对换区采用连续分配方式，而文件区采用离散分配方式，故对换区的磁盘**I/O**速度比文件区的高。
  - **(1)**系统拥有足够的对换区空间
    - 全部从对换区调入所需页面，以提高调页速度
    - 为此，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。
  - **(2)** 系统缺少足够的对换区空间
    - 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。
    - 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。
  - **(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。
    - 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，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。

5.2.3 页面调入策略

- **3.** 页面调入过程
  - 访问的页面未在内存 - 缺页中断
  - 中断处理程序首先保留CPU环境，分析中断原因后， 转入缺页中断处理程序
  - 查找页表，得到该页在外存的物理块
  - 如果此时内存能容纳新页，则启动磁盘I/O将所缺之页调入内存，修改页表
  - 如果内存已满，则须先按照某种置换算法从内存中选出一页准备换出；
  - 如果该换出页未被修改过，可不必将该页写回磁盘；如果此页已被修改，则必须将它写回磁盘，
  - 然后再把所缺的页调入内存， 并修改页表中的相应表项，置其存在位为“1”
  - 将此页表项写入快表中
  - 缺页调入内存后，利用修改后的页表， 去形成所要访问数据的物理地址，再去访问内存数据。

5.3 页面置换算法

页面置换算法

- 如访问页不在内存，且内存无空闲空间
- 需从内存调出一页至硬盘对换区

- 理论上将以后不再访问的页面换出
- 实际上淘汰被访问概率最低的页

5.3.1 最佳置换算法和先进先出置换算法

- **1. 最佳(Optimal)置换算法**
- **基本原理**
  - **淘汰以后永不使用的， 或许是在最长(未来)时间内不再被访问的页面。通常可保证获得最低的缺页率**
  - **实际上无法预知将来的访问次序**
- **例**
  - **进程有8页，分配其3个页面，访问内存的顺序：7012030432123020 1。**
  - **各页面变换情况：5次置换。**

OPT	
7	7
0	70
1	701
2	201
0	
3	203
0	
4	243
2	
3	
0	203
3	
2	
1	201
2	
0	
1	

- 2. 先进先出(FIFO)页面置换算法
  - 淘汰在内存最久的页面
  - 实现：替换指针
  - 9次置换
  - 与实际运行规律不符

	FIFO
7	7
0	70
1	701
2	201
0	
3	231
0	230
4	430
2	420
3	423
0	023
3	
2	
1	013
2	012
0	
1	

5.3.2 最近最久未使用及最少使用置换算法

- LRU(Least Recently Used)置换算法
- 原理
  - 利用“过去”预测“未来”
  - 记录每个页面距离上次被访问的时间 t
  - 淘汰 t 最大的页面

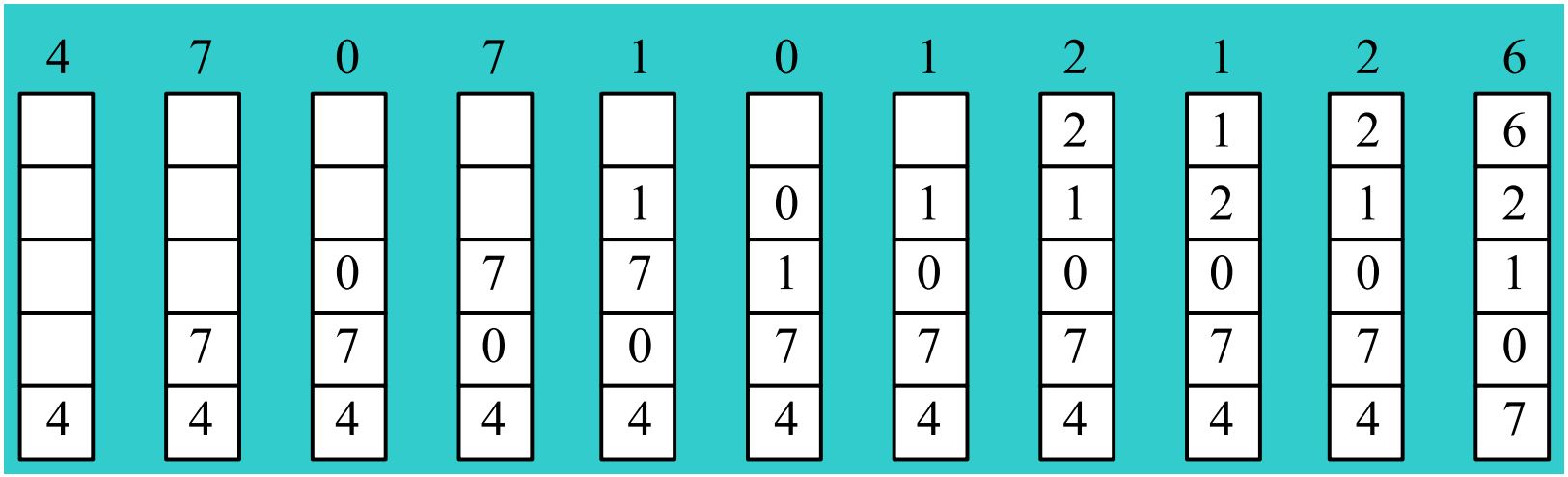
	LRU
7	7
0	70
1	701
2	201
0	
3	203
0	
4	403
2	402
3	432
0	032
3	
2	
1	132
2	
0	102
1	

LRU（Least Recently Used）

- 实现
  - 1 寄存器
    - 每个页面设一个移位寄存器 $R=R_{n-1}R_{n-2}R_{n-3}\ldots R_2R_1R_0$，访问该页面时，$R_{n-1}$置1；
    - 定时信号控制寄存器右移一位；
    - R值最小的那个就是最久未使用的

LRU(Least Recently Used)

- 实现
  - 2 栈
    - 用栈保存当前各页面号
    - 当进程访问某页面，将该页面号从栈中移出到栈顶
    - 这样，栈顶是当前访问的页面号，而栈底是最近最久未使用的页面号
    - 访问6时，发生缺页，淘汰4。

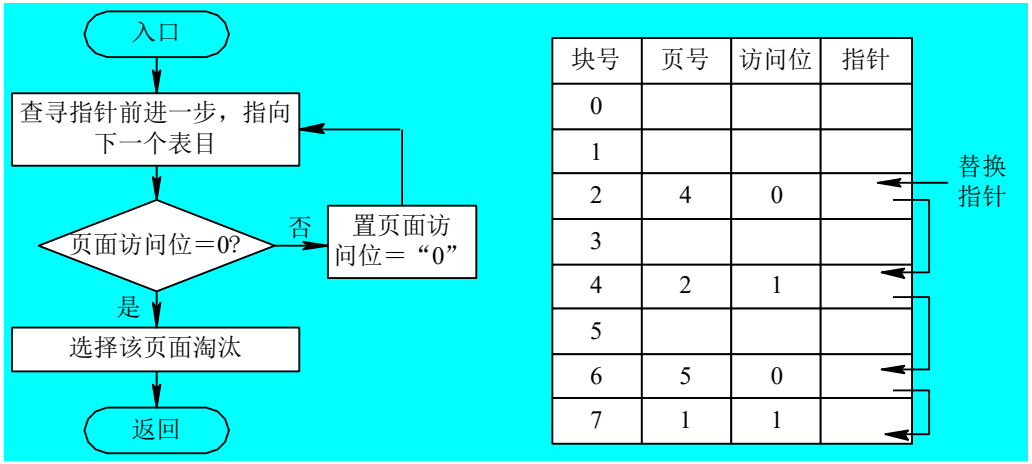


- 最少使用(LFU： Least Frequently Used)置换算法
  - 原理：记录各页面的访问频率，选择最近时期使用最少的页面淘汰
  - 实现：移位寄存器，访问该页最高位置1；定期右移该寄存器。

5.3.3 Clock置换算法

- 一种近似LRU方法，LRU需要硬件支持

- 1 简单Clock算法
  - 内存中所有页面链接成循环队列；
  - 页面设访问位，访问时置1；
  - 需要淘汰页面时，顺序检查各页面，访问位为0则淘汰；若为1则置0，暂不淘汰。
  - 即通过访问位指名该页面是否被访问过，最近未使用算法



5.3.3 Clock置换算法

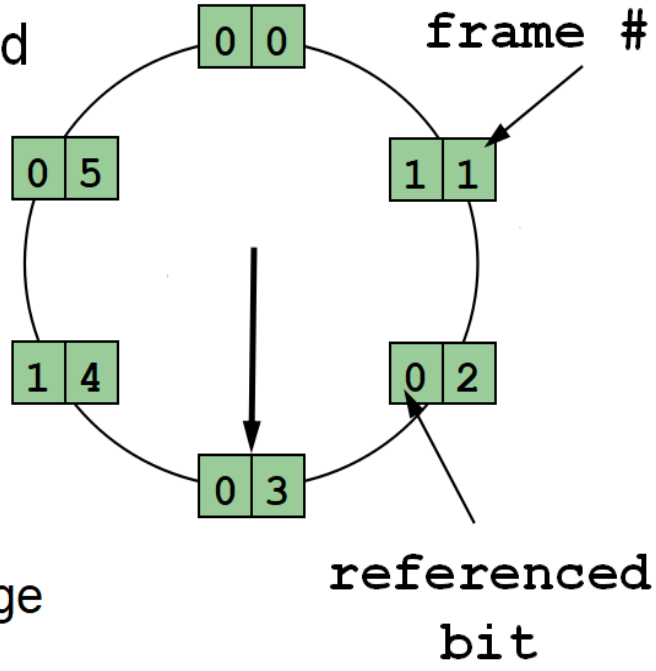
- 2 改进Clock算法
  - 考虑置换代价，页面是否被修改
    - 1类(A=0, M=0): 表示该页最近既未被访问， 又未被修改， 是最佳淘汰页。
    - 2类(A=0, M=1)： 表示该页最近未被访问， 但已被修改， 并不是很好的淘汰页。
    - 3类(A=1, M=0)： 最近已被访问， 但未被修改， 该页有可能再被访问。
    - 4类(A=1, M=1): 最近已被访问且被修改， 该页可能再被访问。

- 2 改进Clock算法
  - 淘汰过程
  - (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，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都置0。
  - (3) 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。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。

5.3.3 Clock置换算法(另一版本）

- Keep circular list of all page frames in memory
- Maintain a clock hand, from where frames are evicted
- New frames are added behind the clock hand

- On a page fault, when we need to evict a page frame:
  - Choose page starting from clock hand
    - If referenced bit is set
      - Unset referenced bit, go to next
    - Else if referenced bit is not set
      - If page is dirty
        - Schedule page write, go to next
      - Else page is clean
        - Select it for replacement, done
    - Next: advance clock hand to next page



5.3.4页面缓冲算法(PBA： Page Buffering Algorithm)

- 影响页面换进换出效率的若干因素
  - 页面置换算法、写回磁盘的频率、读入内存的频率
- 可变分配和局部置换
  - 将淘汰的页面根据其是否被修改放入空闲链表或已修改页面链表
  - 即无论页面是否被修改，淘汰时均放在内存
  - 修改页面的数量达到一定值时，再将他们写回磁盘

5.3.5 访问内存的有效时间

- 在内存、快表命中(λ访问快表时间，t 访问内存时间）

  EAT=λ+t

- 在内存、快表不命中

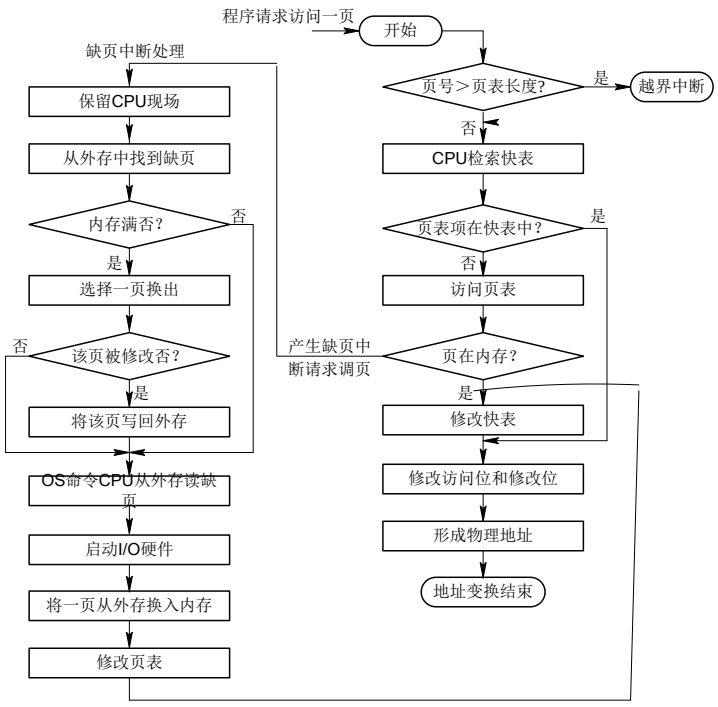
  EAT= λ+t+ λ+t

- 不在内存中 （缺页中断处理时间）

  EAT = λ+t+ ε+ λ+t

- 通用公式（缺页中断处理时间ε, 快表命中率a, 缺页率f）

  EAT= λ+at+(1-a)(t+f(ε+λ+t)+(1-f)(λ+t))



5.4 抖动与工作集模型

前面介绍的各种页面置换算法，都是基于一个前提，即程序的局部性原理。但是此原理是否成立？

- 若局部性原理不成立，则各种页面置换算法就没有区别。例如：若进程对逻辑页面的访问顺序是1、2、3、4、5、6、7、8、9…，即单调递增，则在物理页面数有限的前提下，不管采用何种置换算法，每次的页面访问都必然导致缺页中断。
- 如果局部性原理是成立的，那么如何来证明它的存在，如何来对它进行定量地分析？

1 抖动问题(thrashing)

如果分配给一个进程的物理页面太少，不能包含整个工作集，即驻留集 ⊂ 工作集，则进程将会造成很多缺页中断，需要频繁地在内存与外存之间替换页面，从而使进程的运行速度变得很慢，这种状态称为“抖动”(进程总被阻塞，I/O繁忙)。

2 工作集

工作集：一个进程当前正在使用的逻辑页面集合，可以用一个二元函数W(t, Δ)来表示：

- t是当前的执行时刻；
- Δ 称为工作集窗口（working-set window ），即一个定长的页面访问窗口；
- W(t, Δ)＝在当前时刻 t 之前的 Δ 窗口当中的所有页面所组成的集合（随着 t 的变化，该集合也在不断地变化）；
- | W(t, Δ) | 指工作集的大小，即页面数目。

页面访问顺序：

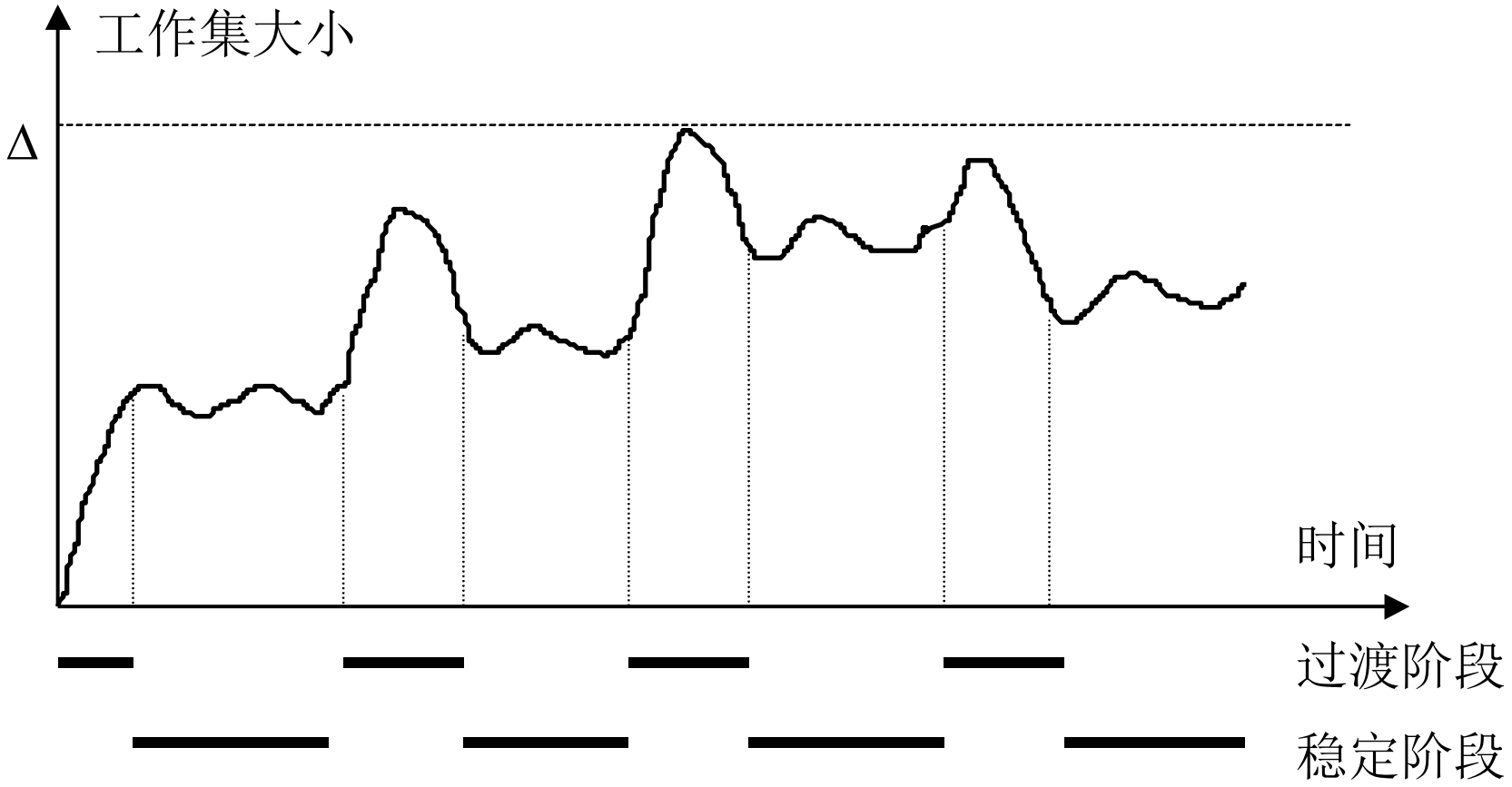
2 6 1 5 7 7 7 7 5 1 6 2 3 4 1 2 3 4 4 4 3 4 3 4 4 4 1 3 2 7

如果 Δ 窗口的长度为10，那么：

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

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

工作集大小的变化：进程开始执行后，随着访问新页面逐步建立较稳定的工作集。当内存访问的局部性区域的位置大致稳定时，工作集大小也大致稳定；局部性区域的位置改变时，工作集快速扩张和收缩过渡到下一个稳定值。



3 驻留集

驻留集是指在当前时刻，进程实际驻留在内存当中的页面集合。

- 工作集是进程在运行过程中固有的性质，而驻留集取决于系统分配给进程的物理页面数目，以及所采用的页面置换算法；
- 如果一个进程的整个工作集都在内存当中，即驻留集 $\supseteq$ 工作集，那么进程将很顺利地运行，而不会造成太多的缺页中断（直到工作集发生剧烈变动，从而过渡到另一个状态）；
- 当进程驻留集的大小达到某个数目之后，再给它分配更多的物理页面，缺页率也不会明显下降。

4 抖动的预防

- 采用局部置换算法。保证进程之间不互相影响，但不能消除抖动；
- 调入作业考虑是否满足了现有进程工作集要求；
- 暂停进程。依据一定原则暂停某个或某些进程，消除抖动
- L=S 准则。L :平均缺页间隔，S:平均页面置换时间。当 L=S时处理能力利用较充分。

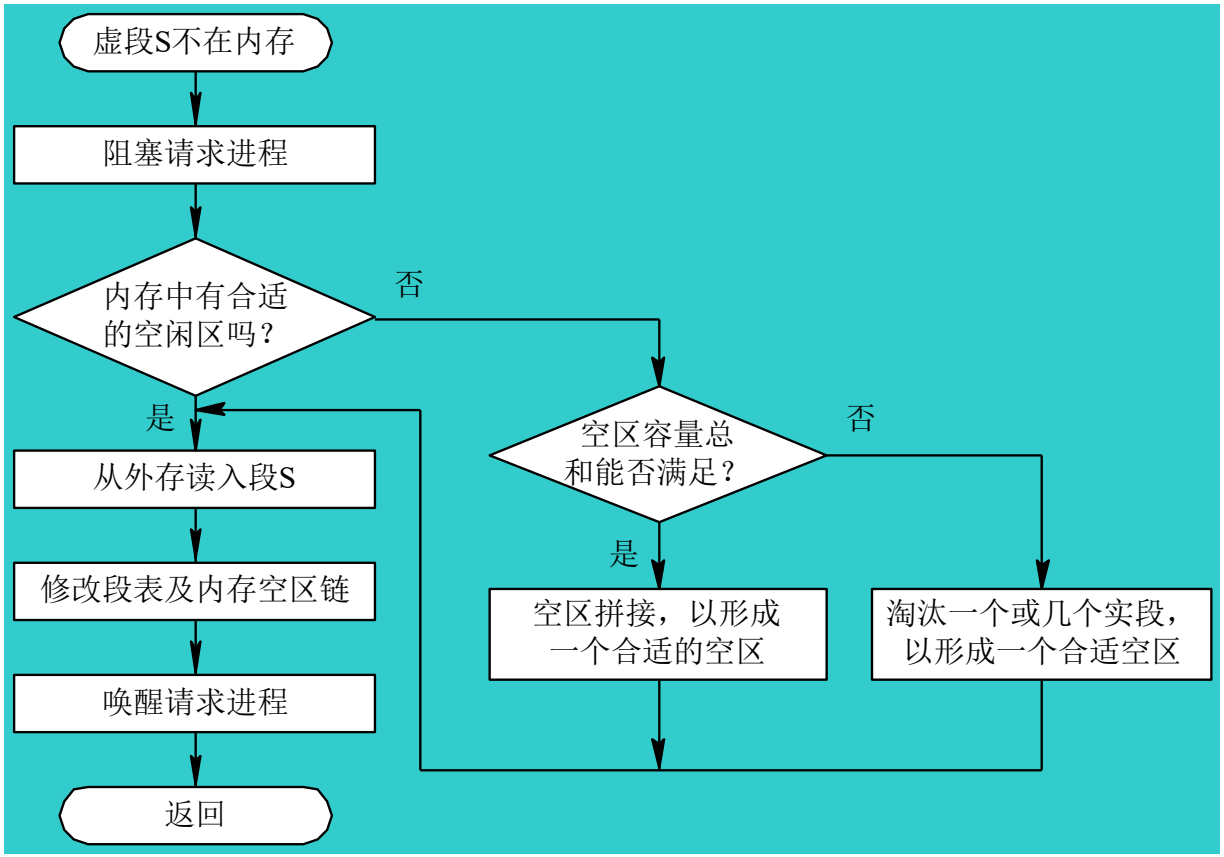
5.5 请求分段存储管理方式

程序运行前，调入部分段，启动运行后，OS将所缺段调入内存。

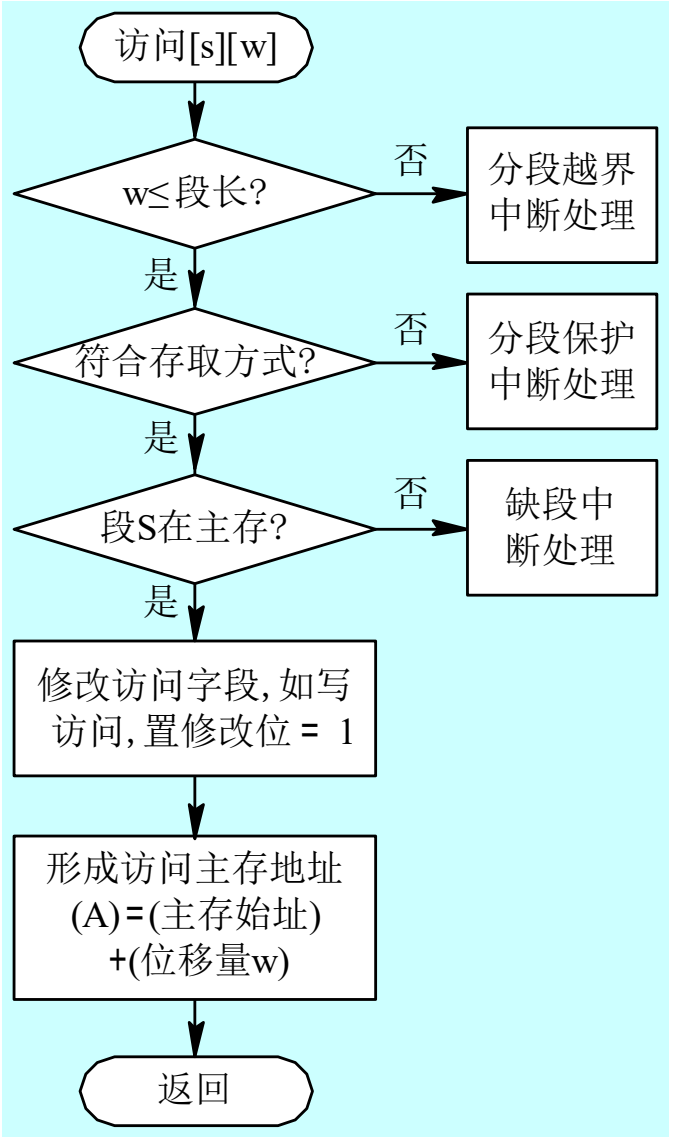
5.5.1 请求分段中的硬件支持

- 1 段表
  - 段名、段长、基址、存取方式
  - 访问字段：记录该段的访问频率
  - 修改位、存在位
  - 增补位：指明该段运行过程中是否增长
  - 外存地址

- 2. 缺段中断机构
  - 段长不固定

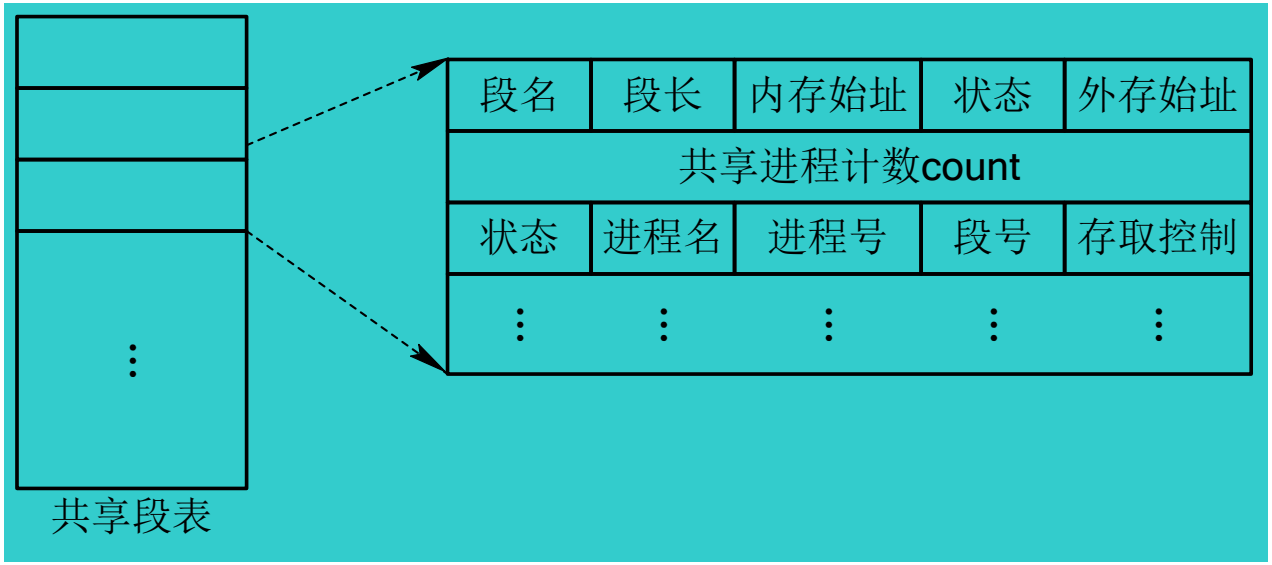


- **3. 地址变换机构**


访问[s][w]
w≤段长?
否
分段越界
中断处理
是
符合存取方式?
否
分段保护
中断处理
是
段S在主存?
否
缺段中
断处理
是
修改访问字段, 如写
访问, 置修改位 = 1
形成访问主存地址
(A)=(主存始址)
+(位移量w)
返回

5.5.2 分段的共享与保护

- **1. 共享段表**
  - 各共享段各占一个表目
  - 共享进程计数器
  - 存取控制字段



- **2. 共享段的分配与回收**
  - **1) 共享段的分配**
    - 第一个请求使用该共享段的进程，系统为该共享段分配一物理区，再把共享段调入，同时将该区的始址填入请求进程的段表，在共享段表中增加一表项，填写有关数据，把**count**置为**1**；
    - 其它进程需要调用该共享段时，而只需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，为此进程增加一表项，再执行**count=count+1**操作。
  - **2) 共享段的回收**
    - 当共享此段的某进程不再需要该段时，在进程段表中撤销相应表项，撤销共享段表中该段表项中这一进程的信息，以及执行**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；否则**(**减**1**结果不为**0)**， 则只是取消调用者进程在共享段表中的有关记录。

- **3. 分段保护**
  - **1)越界检查**
    - 段表记录段的始址和长度
  - **2) 存取控制检查**
    - 只读、只执行、读/写
  - **3) 环保护机构**
    - 低编号的环具有优先权
      - 操作系统核心位于0环
      - 重要的使用程序和操作系统服务占据中间环
      - 一般应用程序安排在外环

例题：

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(Page Frame)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为 10^8ns（已含更新 TLB 和页表的时间），进程的驻留集大小固定为 2，采用最近最少使用置换算法（LRU）和局部淘汰策略。假设①TLB 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。

（1）根据页式管理的工作原理，应先考虑页面大小，以便将页号和页内位移分解出来。页面大小为 4KB，即 2^{12}，则得到页内位移占虚地址的低 12 位，页号占剩余高位。可得三个虚地址的页号 P如下：

- 2362H：P=2，访问快表10ns，因初始为空，访问页表100ns 得到页框号，合成物理地址后访问主存100ns，共计10ns+100ns+100ns=210ns。
- 1565H：P=1，访问快表10ns，落空，访问页表100ns落空，进行缺页中断处理10^8ns，合成物理地址后访问主存100ns，10ns+100ns+10^8ns+10ns+100ns

- 25A5H：P=2，访问快表，因第一次访问已将该页号放入快表，因此花费10ns便可合成物理地址，访问主存100ns，共计10ns+100ns=110ns。

（2）当访问虚地址 1565H 时，产生缺页中断，合法驻留集为 2，必须从页表 中淘汰一个页面，根据题目的置换算法，应淘汰 0 号页面，因此 1565H 的对应页框号为 101H。由此可得 1565H 的物理地址为 101565H。