页帧分配和系统抖动

页帧分配

- 一个进程起码需要最小数量的页面，才能运行
- 例如： PC 的汇编指令 MOV – 4 个页面
  - 指令本身占 6 字节，可能横跨 2 个页面
  - 含一个源数据块，1 个页面
  - 含一个目标数据快，1 个页面
- 常用 2 种页帧分配策略
  - fixed allocation
  - priority allocation

Fixed Allocation

- Equal allocation – 例如，假设 100 个页帧， 5 个进程，那么，每个进程分得 20 个页帧
- Proportional allocation – 按照进程映像占用的逻辑空间大小，按比例分配页帧

Fixed Allocation

- Proportional allocation

  - s_i = size of process p_i
  - $S = \sum s_i$
  - m = total number of frames
  - a_i = allocation for $p_i = \frac{s_i}{S} \times m$

$$m = 64$$

$$s_i = 10$$

$$s_2 = 127$$

$$a_1 = \frac{10}{137} \times 64 \approx 5$$

$$a_2 = \frac{127}{137} \times 64 \approx 59$$

Priority Allocation

- 依照进程的优先权，按比例分配页帧给进程

- 如果进程 P_i 产生了缺页
  - 从进程原本占用的页帧（储存了进程的页面）中，选取一个页面换出
  - 从较低优先权的进程中，选取一个页帧，换出其储存的页面

全局分配 vs. 局部分配

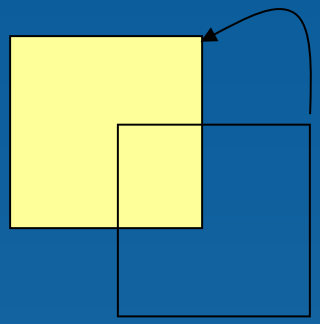
- **全局置换**
  - 进程从所有页帧中选取一个，进行置换
  - 进程可以选取其它进程占用的页帧
- **局部置换**
  - 进程只能从它自己占用的页帧中选取一个，进行置换

抖动 (Thrashing)

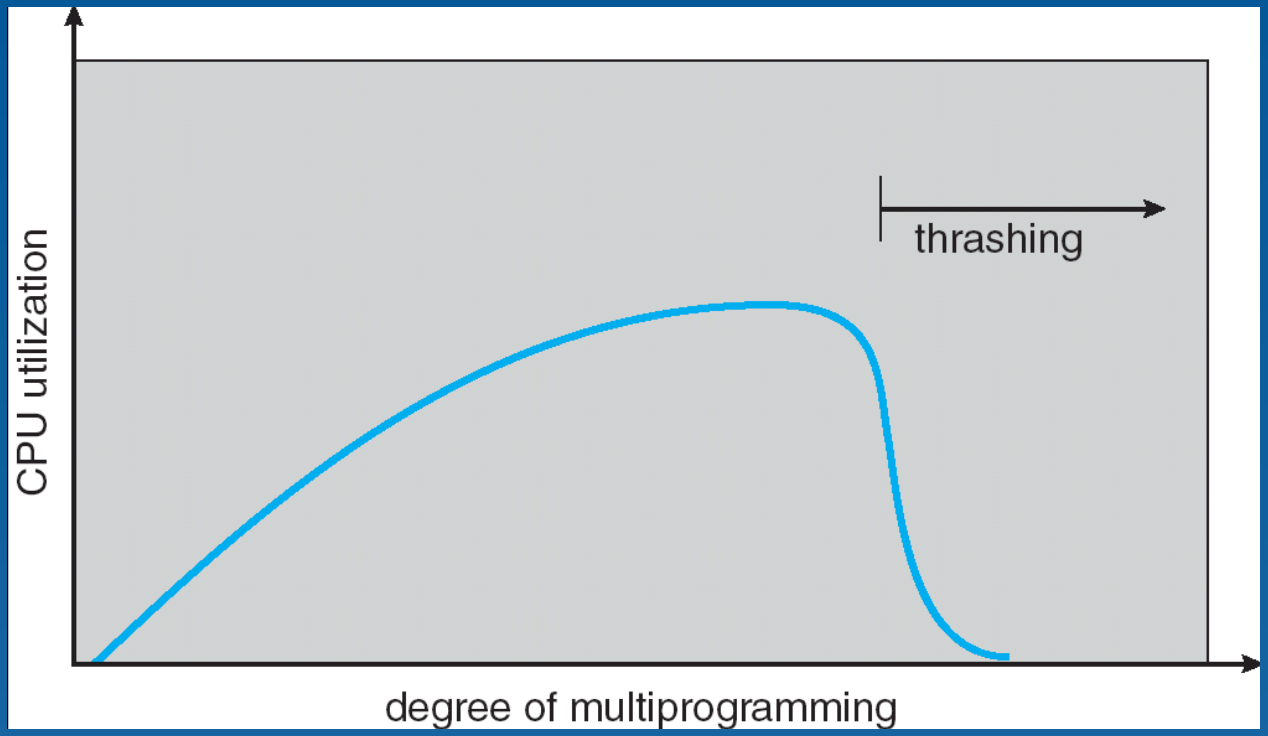
- 当进程没能拥有足够的页帧，会导致很高的缺页次数。其后果:
  - 很低的 CPU 利用率
  - 误导 OS 以为有必要提高多任务的程度
  - 误导 OS 装入更多作业，内存中驻留更多进程
  - 于是，每个进程拥有的页帧数更少
  - 如此恶性循环，会怎样？
- **抖动** ≡ 进程忙于换入、换出页面

示例：抖动

- 进程只有 2 个页帧，执行 MOVSB 指令搬迁 1 个页面的数据



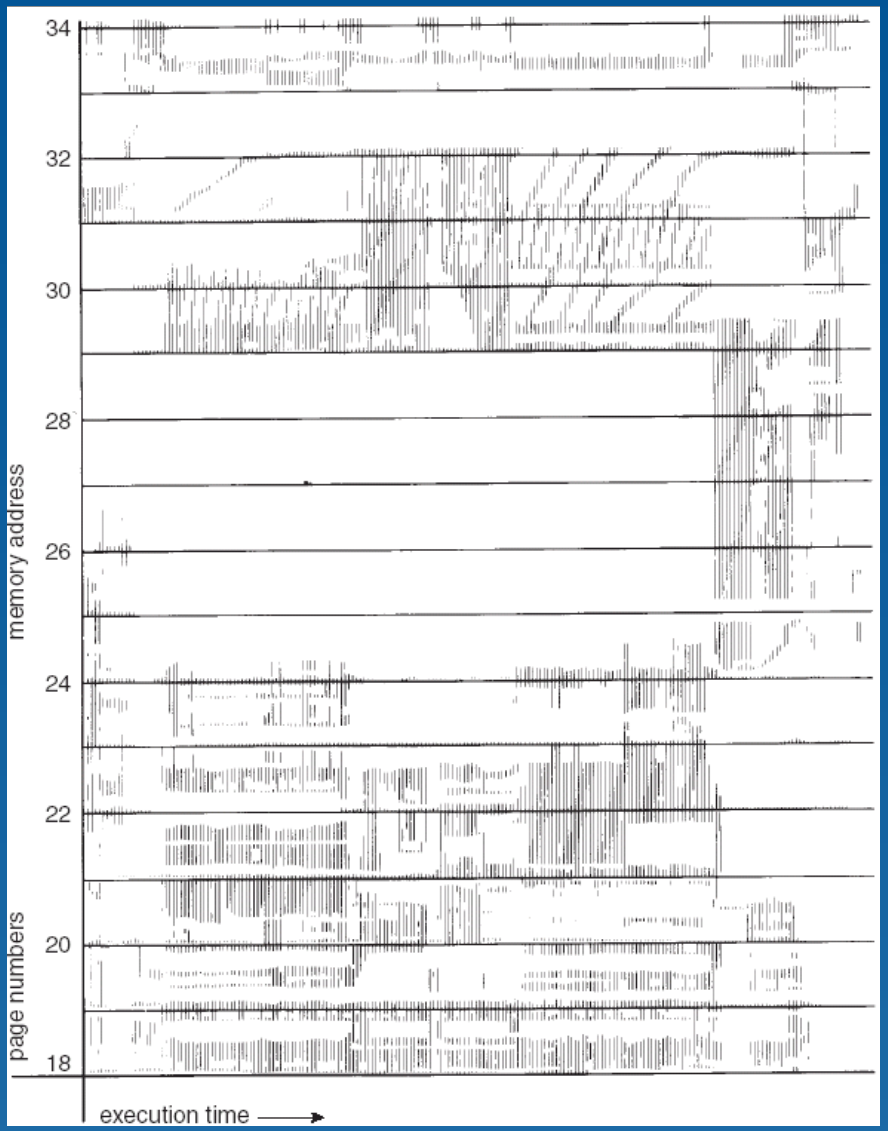
抖动（续）



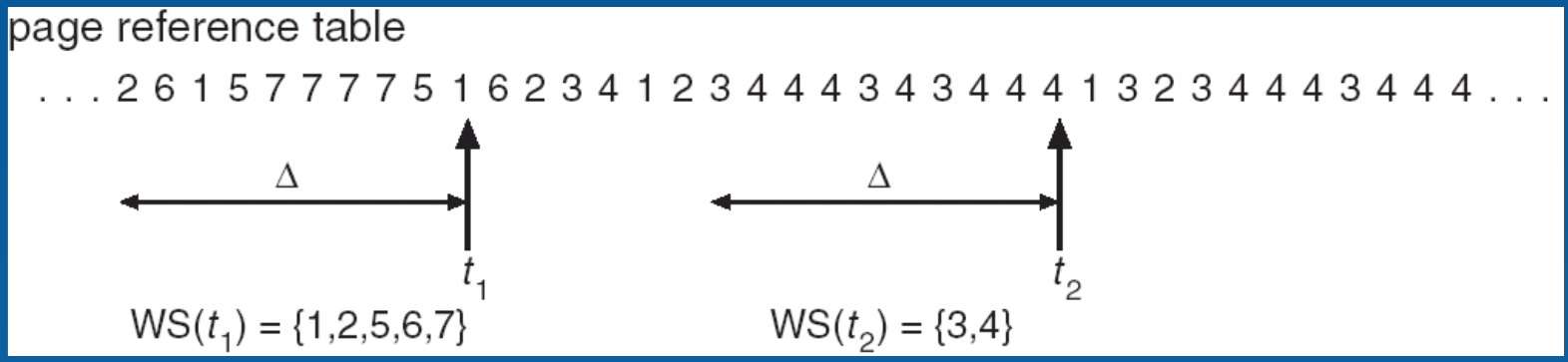
按需调页，抖动

- 为什么按需调页策略十分有效 ？
  因为局部性 (Locality model)
  - 进程访问内存，呈现从这个 locality 到那个 locality 的规律
  - Localities 可能有重叠，但仍然可以区分出 locality
- 为什么会产生抖动现象 ？
  Σ localities > 内存容量

内存访问的局部性 (Locality) 规律



示例：工作集 (Working-Set)



工作集 (Working-Set) 模型

- Δ ≡ 工作集窗口 ≡ 固定数目的页面引用
- 例如： 10,000 条指令

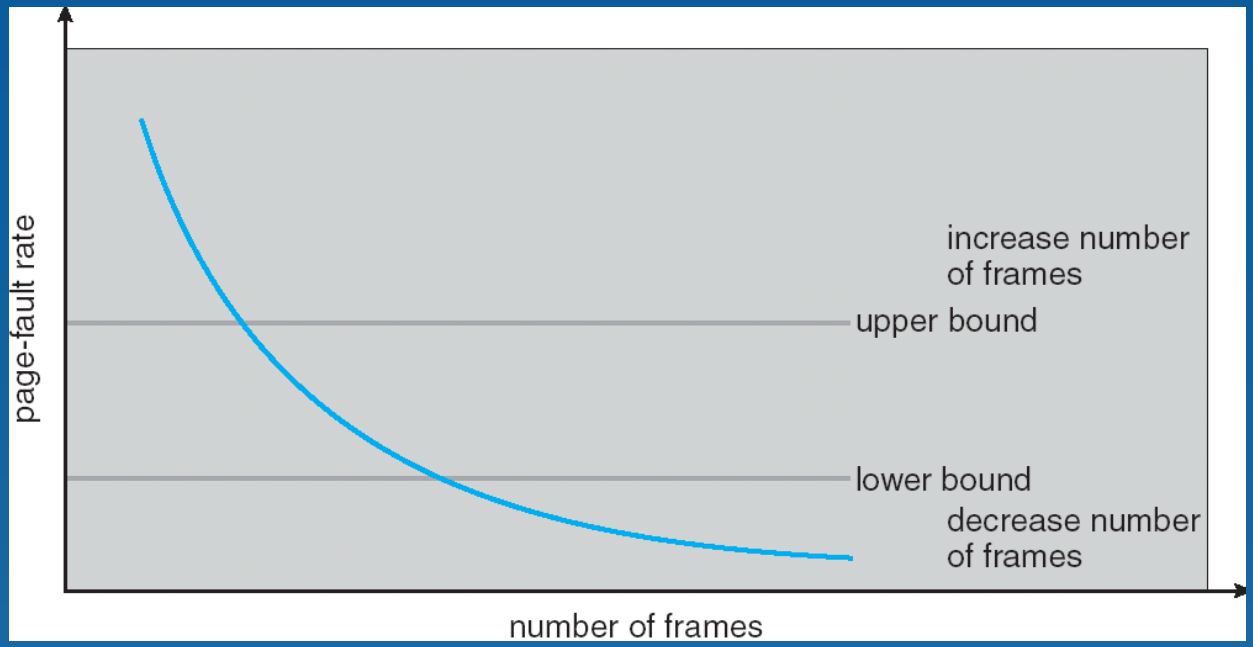
- 进程 Pi 的工作集 WSSi =
  最近一次 Δ (随时间变更) 的页面引用总数
  - Δ 太小，无法覆盖完整的 locality
  - Δ 太大，跨越若干 localities
  - Δ = ∞ ⇒ 覆盖整个程序，因此没有意义

工作集 (Working-Set) 模型

- $D = \Sigma\ WSS_i \equiv$ 就是页帧总需求数
- 当 $D > m \Rightarrow$ 抖动
- 可以采纳策略
- 如果 $D >$ m， 那么挂起一个进程

根据缺页频率调整页帧数

- OS 维持“可接受的”缺页率的范围
  - 如果缺页率太低，强迫进程释放一些页帧
  - 如果缺页率太高，给进程多分配一些页帧




End