调度算法

进程调度算法

先到先服务调度算法（First Come First Severd, FCFS）

- 「非抢占式」每次从就绪对列选择最先进入对列的进程，然后一直运行，直到进程退出或被阻塞，才会继续从队列中选择第一个进程接着运行
- FCFS对长作业有利，适用于CPU繁忙型作业系统，而不适用I/O繁忙型的系统

最短作业调度算法（Shortest Job First, SJF）

- 优先选择运行时间最短的进程来运行，有助于提高系统吞吐量
- SJF对长作业不利，很容易造成一种极端现象

高响应比优先调度算法（Highest Response Ratio Next, HRRN）

- 权衡了短作业和长作业，每次进行进程调度时，先计算「响应比优先级」，然后把「响应比优先级」最高的进程投入运行
- 优先级=（等待时间+要求服务时间）/要求服务时间
- 如果两个进程的「等待时间」相同，「要求的服务时间」越短，「响应比」就越高，短作业进行容易被选中运行
- 如果两个进程「要求的服务时间」相同时，「等待时间」越长，「响应比」就越高，兼顾到了长作业进程，等待时间足够长响应比提升

时间片轮转调度算法（Round Robin, RR）

- 最古老、最简单、最公平的调度算法
- 每个进程被分配一个时间段，称为时间片（Quantum），即允许该进程在该时间段中运行
- 如果时间片设置的太短会导致过多的进程上下文切换，降低了CPU效率；反之太长会对短作业的响应时间边长（通常20ms-50ms）

最高优先级调度算法（Highest Priority First，HPF）

- 从就绪队列中选择最高优先级的进程进行运行
- 静态优先级
  - 创建进程的时候，已经确定了优先级，整个运行时间优先级都不会变化
- 动态优先级
  - 根据进程的动态变化调成优先级
- 非抢占式
  - 就绪队列出现优先级高的进行，运行完当前进程再选择优先级高的进程
- 抢占式
  - 当就绪队列出现优先级高的进程，当前进程挂起，调度优先级高的进程运行
- 可能会导致优先级低的进程永远不会运行

多级反馈对列调度算法（Multilevel Feedback Queue）

- 「时间片轮转算法」和「最高优先级算法」综合发展
- 「多级」表示有多个队列，每个队列优先级从高到低名同事优先级越高时间片越短
- 「反馈」表示如果有新的进程加入优先级高的队列，立刻停止当前正在运行的进程，转而去运行优先级高的队列
- 工作流程
  - 设置多个对列，赋予每个队列不同的优先级，每个「对列优先级从高到低」，同时「优先级越高时间片越短」
  - 新的进程会被放到第一级队列的末尾，按先来先服务的原则排队等待被调度，如果在第一级队列规定的时间片没有运行完成，则转入第二级对列尾部，以此类推，直至完成
  - 当较高优先级的队列为空，才调度较低优先级的对列中的进程运行。如果进程运行时，有新进程进入较高优先级的队列，则停止当前运行的进程将其移入到原队列末尾，接着让较高优先级的进程运行
- 该算法很好的兼顾了长短作业，同时又较好的响应时间

页面置换算法

最佳页面置换算法（OPT）

- 置换在「未来」最长时间不访问的页面
- 算法实现需要计算内存中每个逻辑页面「下一次」访问时间，然后比较，选择未来最长时间不访问的页面
- 实际系统中无法实现，因为访问时动态的，无法预知的，所以页面置换算法是为了衡量你的算法效率，如果越接近说明效率越高

先进先出置换算法（FIFO）

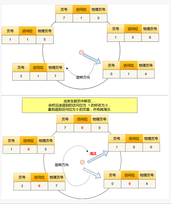
- 选择在内存驻留时间很长的页面进行置换

最近最久未使用置换算法（LRU）

- 选择最长时间没有被访问的页面进行置换

时钟页面置换算法（LOCK）

- 跟LRU近似，对FIFO的一种改进



- 将所有页面保存在类似钟面的「环形链表」，一个指针指向最老的页面，当发生中断的时候，先检查表指针的页面
  - 如果访问的位置是0就淘汰该页面，并把新页面插入到这个位置，然后指针后移
  - 如果访问位是1就清除访问位，把表针前移一个位置，重复这个过程找到访问位为0的页面位置

最不常用置换算法（LFU）

- 发生缺页中断时，选择「访问次数」最少的页面，将其淘汰
- 需要增加一个计数器实现，硬件成本较高，另外对这个计数器查找那个页面访问次数最小，查找链表本身，如果链表长度很大，是非常耗时，效率不高
- LFU只考虑了频率问题，没考虑时间问题，如果有些页面再过去的时间访问频率很高，但是现在已经没有访问了，而当前访问的页面由于没有这些页面访问次数高，发生缺页中断时，可能会误伤当前开始频繁访问，但访问次数还不高的页面

磁盘调度算法

先来先服务算法（First-Come，First-Served，FCFS）

- 先到来的请求先被服务

最短寻道时间优先算法（Shortest Seek First，SSF）

- 优先选择从当前磁头位置所需寻道最短的时间请求
- 可能出现「饥饿现象」

扫描算法（Scan）

- 解决最短寻道时间可能会出现的饥饿问题
- 磁头在一个方向上移动，访问所有未完成的请求，直到磁头到达该方向上的最后的磁道，才调换方向
- 中间部分磁道比较占便宜，响应频率比较高

循环扫描算法（Circular Scan, CSCAN）

- 优化扫描算法，按相同方向进行扫描，使得每个磁道的响应频率基本一致
- 只有磁头朝特定方向移动时，才处理磁道访问请求，而返回时直接快速移动至最靠边缘的磁道，也就是复位磁头（过程很快，中途不处理任何请求）

LOOK和C-LOOK算法

- 优化扫描算法，不需要移动到最始端或最末端，仅仅移动到最远请求位置然后反向移动，「LOOK 反向移动过程中会响应请求」
- 优化循环扫描算法，不需要移动到最始端或最末端，仅仅移动到最远请求位置然后反向移动，「C-LOOK 反向移动过程中不会响应请求」