OS中的算法
- CPU调度
  - 三种调度方式
    - 高级调度
    - 中级调度
    - 低级调度
  - 调度的时机
- 进程调度
  - 有关进程调度的基本概念
  - 算法
    - FCFS先到先来服务
    - SJF短作业优先
    - 优先级调度算法
    - HRRN高响应比优先
    - RR(Round-Robin)时间片轮转
    - 多级反馈队列
- 内存调度
  - 连续分配
  - 非连续分配
  - 缺页置换算法
    - OPT最佳置换算法
    - FIFO先进先出置换算法
    - LRU最近最久未使用置换算法
    - CLOCK时钟算法，又称为NRU最近未用算法
    - CLOCK的改进型
- 磁盘调度
  - 基本概念
  - 算法
    - FCFS,先来先服务算法
    - SSTF 最短寻找时间优先
    - SCAN扫描算法，又称电梯调度算法
    - C-SCAN，循环扫描
    - LOOK,C-LOOK

OS中的算法

讨论顺序按CPU<->内存<->磁盘的顺序展开，

CPU<->内存：主要对象是作业和进程，发生调度的场景是CPU和内存，偶尔涉及到外存。

内存<->磁盘：主要对象是数据，发生调度的场景是内存和磁盘

CPU调度

三种调度方式

面向作业，进程，CPU和内存分为三种

高级调度

又称**作业调度**，是将作业从外存中调入内存，将作业处理完后再把作业从内存调至外存，作业的调入调出只会发生一次

中级调度

又称**内存调度**，面向进程，当内存空间不足时，OS会把暂时不能运行的进程调至外存等待，此时的进程称为挂起态，注意这个时候进程映像被调至外存，但是进程的PCB仍然存在于内存中

低级调度

又称**进程调度**，是在内存中的调度，按照进程调度算法从就绪队列中选择一个进程将CPU的资源分配给它。

三种调度	发生频率	调度对象	涉及位置
作业调度	低	作业	内存——外存
内存调度	中	进程	内存——外存
进程调度	高	进程	内存

调度的时机

不能进行调度的情况：

- 处理中断
- 进程在OS内核程序**临界区**
- 其它需要完全屏蔽中断的原子操作

可以进行进程调度的情况：

- 进程时间片用完
- 进程发出中断指令切换到阻塞态
- 进程无法继续执行时

补充说明：

临界资源是指在同一时间只能有一个进程访问的资源，比如说输入机，打印机等等，**临界区**是访问**临界资源的代码段**，进程准备进入临界区之前需要检查是否有其它进程在使用这个资源，如果有则阻塞自己，如果没有则进入给临界资源加锁并且进入临界区，退出临界区时进程会解锁以便其它进程访问。

进程调度

前面提到，CPU的进程调度会根据一定算法从就绪队列中选择一个进程来执行，那么这个算法就涉及到进程调度

有关进程调度的基本概念

- **长作业，短作业**，运行时间长的称为长作业，运行时间短的成为短作业。一般**CPU繁忙型进程**由于需要使用长时间的CPU资源，所以它是**长作业**，而**IO繁忙型进程**只需要得到CPU然后发出I/O请求就可以做别的事了，索引它是短作业。、从性能上来说，CPU应该优先满足短作业，再满足长作业。
- **优先级**

  优先级别高的进程会更先使用CPU，进程可以按照优先级分为

  系统进程>非系统进程

  交互型>非交互型

  I/O型>非I/O型
- **抢占式**，**非抢占式**，当一个进程使用CPU资源的时候，其它进程只能等待它使用完成才轮到自己使用，这种方式就要非抢占式，与之相对应的是抢占式，当其它进程需要使用CPU的时候OS可以剥夺当前正在运行的进程的CPU资源提供给新的进程使用。
- **衡量调度算法的指标**
  - 周转时间=完成时间-到达时间
  - 平均周转时间=总的周转时间/作业数
  - 带权周转时间=周转时间/实际运行时间
  - 平均带权周转时间=总的带权周转时间/作业数
  - 等待时间，指进程等待处理机资源的时间
  - 响应时间，指用户提交请求到系统首次产生响应所用的时间
  - CPU利用率=CPU运行时间/总时间
  - 系统吞吐量，指单位时间内CPU完成作业的数量

算法

FCFS先到先来服务

按照进程到达时间先后顺序执行，先到达的进程先服务

SJF短作业优先

顾名思义，运行时间短的进程优先执行

优先级调度算法

其中可以分为抢占式和非抢占式，按照一定规则使得优先级别高的进程先运行，这种调度算法会饥饿

HRRN高响应比优先

比较优秀的一个算法，兼顾了短作业优先并且让长作业不会饥饿

响应比计算方法是：

$$响应比 = \frac{等待时间+要求服务时间}{要求服务时间}$$

RR(Round-Robin)时间片轮转

不会饥饿，适用于多用户系统

时间片过长会退化成FCFS，时间片过短，性能会下降

多级反馈队列

同样是解决了短作业，长作业算法的缺点的算法，进程不会饥饿

总结

算法	思想	优点	缺点	是否饥饿	适用于	特点
FCFS	按序服务	不会饥饿	对短作业（I/O型）不好	否	CPU型作业频繁	有利于长作业，不利于短作业；算法简单，效率低
SJF	所花时间短的先服务	会饥饿	对长作业（CPU型）不好	是	I/O型作业频繁	对长作业不利。**平均等待时间和平均周转时间最少（不考虑饥饿的话性能最优）**
优先级	优先级别高的先服务	紧急任务可以优先使用CPU	饥饿	是	任务有紧急性，需要较快响应的系统，终端型作业和短作业	衍生算法多，有抢占式和非抢占式两种，同时可以动态调整优先级避免产生饥饿
HRRN	响应比高的进程先服务	兼顾了长作业和短作业	无明显缺点	否		短作业响应比天然高，长作业由于等待时间长响应比也会升高从而不会饥饿
RR	按照时间片来分配进程资源	分时系统最适合的算法	无明显缺点	否	多用户系统，分时系统	时间片的选取很关键，时间片长则退化成FCFS，时间片短则频繁切换进程开销过大
多级反馈队列	优先级队列+时间片轮转	终端型：短作业优先。短作业：周转时间短。长作业不会饥饿	无明显缺点	否		可以衍生出很多算法

内存调度

内存中引入**非连续分配**时，由于发生缺页的时候要根据一定算法选择换入，换出的页，因此有了专门的调度算法，在讲算法之前，先说一下操作系统划分内存的分配方式，重点关注的是调页算法

操作系统内存的分配方式有两种，一种是**连续分配**，另一种是**非连续分配**

连续分配

连续分配是指一个进程的数据存放位置必须是连续的

- 单一连续分配

  整个内存只分配给一个进程，用于早期的操作系统。注意只有这种分配方式是给一道进程的，其它都是给多道进程的

- 固定分区分配

  将内存划分为固定分区，每个进程装入适合的分区中

  因为一开始就划分好了分区，所以称为固定分区

  分区方式有两种：

  - 分区大小不变
  - 分区大小可变

- 动态分区分配

  一开始并不划分分区，而是根据进程运行时动态的调整分区大小，当一个进程装入时根据算法装入合适的位置

  分区分配算法主要有这几种

  - BF（best fit）最佳适应

    算法选择一个最合适的内存空间装入，此算法需要维护一个存储内存空间大小的链表，从小到大排序，因此这个算法需要额外的时间和空间去维护链表

  - WF（worst fit）最坏适应

    总是选择一个最大的内存空间装入，此算法同样要维护一个从大到小排序的链表，需要额外的时间和空间

  - FF（first fit）首次适应

    选择一个第一次能满足进程所需空间大小的内存块装入。由于装入发生在低地址段的频率较高，所以低地址处会有许多碎片

  - NF（next fit）邻近适应

    FF的改进版，从上一次比较的位置开始寻找内存块，而不是从头开始寻找内存块。NF会造成大的内存空间分裂成小碎片。

    以上四种算法FF性能最好，BF和WF性能都很差

    算法之间的比较

名词	性能	额外空间	特点
BF	差	有序链表	留下很多小碎片（外部碎片）
WF	差	有序链表	大块的内存被分割了
FF	最优	无	低地址处留下很多碎片，增加了查找时间
NF	还可以	无	把大空间分割了

    大块的空间被分割是很可惜的事情，因为可能找不到空间装入某些大进程

    很多的小碎片也很麻烦

上述三种分区方法的比较（注意这里始终是连续分配方式下的不同分区方法）

名称	碎片类型	支持进程	特点
单一连续分配	内	一个	唯一采用覆盖和交换的分配方式，用于早期的操作系统

名称	碎片类型	支持进程	特点
固定分区分配	内	多个	不能实现多进程共享存储区，内存利用率低
动态分区分配	外	多个	会产生很多小的碎片

非连续分配

特别重要的一个知识点

- 页式

  将内存分成固定大小的分区，每一个分区称之为页框/页帧/内存块/物理块（内存块的说法贴切）

  需要一个索引表来指向分区，索引表的内容就是块号，索引表称为页表，索引表中的项称为页表项

  用户程序也按页分，这里的每一页对应内存中的一个内存块
- 段式

  将内存分为长度不等的段，此时索引表的内容是（段长，起始地址）

  段式比页式更方便的实现数据和信息共享
- 段页式

  先分段，再分页

名词	碎片类型	索引表内容	特点
页式	内部	块号	最常用
段式	外部	段长，内存中起始地址	更方便的实现数据和信息共享
段页式	内部	段长，页表中起始地址；块号	结合两者优点

缺页置换算法

OPT最佳置换算法

思想：每次发生缺页时，从当前时刻向后看，找到当前内存中最后使用的页，然后置换

例如

块号\访问页号	2	3	1	4	1	4	3	2
1	2	2	2	4				2
2		3	3	3				3
3			1	1				1
是否缺页	✓	✓	✓	✓				✓

9次访问，共缺页5次，最后一次缺页随机选择一个页换出即可

请注意：OPT算法由于需要缺页之后的访问顺序，而os中无法知道进程执行过程中页的访问顺序，所以OPT算法只是理论上的算法，无法实现

FIFO先进先出置换算法

思想：维护一个页号队列，每次发生缺页时，置换队首页号

FIFO很直观，但是会产生**Belady异常**，即：增加物理块反而会增加缺页次数，这是由于FIFO要使用队列，而基于堆栈类的算法（LRU）不会产生Belady异常

LRU最近最久未使用置换算法

思想：每次发生缺页时，从当前时刻往前看，找到当前内存中最久未使用的页，然后置换

为了模仿OPT而产生的算法，性能比较好，但是需要**寄存器和栈的硬件支持**，因此开销比较大

CLOCK时钟算法，又称为NRU最近未用算法

思想：增加修改位，按照顺/逆时间遍历所有的页号，将符合要求的页换出，下一次缺页时从上次换入的页开始遍历

流程：

第一轮扫描，选择未修改的页置换，此时将扫描过的页的修改位置为0

重复第一轮扫描，总能找到一页置换

CLOCK的改进型

思想：增加更多的标志位，（访问位，修改位）

假设(1,1)表示访问，且有修改，(1,0)表示访问但未修改

因为修改过的页要花更多的时间写回磁盘，我们优先选择未修改过的页

又因为根据局部性原理，访问过的页很有可能不久再被访问，所以我们优先换出未访问的页。

流程：

第一轮扫描，选择（0，0）的页置换

第二轮扫描，选择**（0，1）**的页置换，也就是未被访问但是被修改过的页。此轮将扫描过的修改位改为1

重复上述两轮扫描，总是能找到一页并置换

此算法换页的淘汰顺序是：（0，0）>（0，1）>（1，0）>（1，1）

请注意实际上不存在（0，1）的页，也就是不存在未被访问但是修改过的页，这是算法执行过程中修改标志位导致的

磁盘调度

为什么要磁盘调度?

因为CPU要读入数据，数据不在内存中，OS就从磁盘中读取数据到内存中，磁盘中如何调度，按照什么顺序调度，就涉及到下面的算法

基本概念

磁盘的组成：磁头，磁臂，盘片

解释：磁头读取数据，磁臂负责移动磁头，盘片存储数据

盘片的组成：盘面，磁道，扇区（盘块）

一个盘片有上下两个盘面组成，一个盘面有多条磁道，磁道越多磁盘可以存储的数据就越多，一个磁道可以等分为多个扇区

所有盘片的同一磁道组成柱面

磁盘访问时按照（柱面，盘面，扇区）的顺序访问

因为移动磁臂的开销很大，所以优先确保这个柱面上所有需要读入的磁道都读取入完了才 移动磁臂。然后保证同一个盘面没有要读入的内容了，再切换扇区

算法

FCFS,先来先服务算法

思想：按照请求序列，先来的先读

SSTF 最短寻找时间优先

思想：总是优先满足离磁头最近的请求

特点：会饥饿

SCAN扫描算法，又称电梯调度算法

规定磁头移动方向，只有磁头移动到**一端**后才**反向移动**

不会饥饿，但是由于靠近一端的请求序列会在短时间内扫过两次所以对请求序列不公平

C-SCAN，循环扫描

规定磁头移动方向，只有磁头移动到**一端**后，才从**另一端**开始移动

LOOK,C-LOOK

LOOK对应SCAN

C-LOOK对应C-SCAN

带有LOOK标识的，表示以上两个算法的磁头并不是严格的执行到另一端再返回，而是遇到**该方向最后一个请求**就开始反转移动方向。