第 6 章 进程调度

月出皓兮　苏曙光老师的课堂笔记

2022 年 4 月 14 日

目录

1 进程调度（schedule）

1.1 定义

在队列中按某种策略选择一个最合适的对象。

1.2 分类

1. 长程调度/宏观调度/作业调度【作业/磁盘】
2. 中程调度/交换调度【进程/内存/磁盘】
3. 短程调度/进程调度【进程/内存】
4. I/O 调度/设备调度【进程/设备】

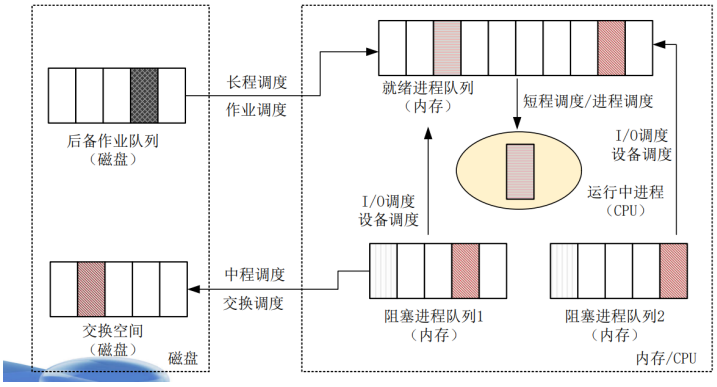

图 1: 进程调度分类

在这里我们主要讨论的是短程调度（一直将其称为进程调度）。
在**合适的时候**以**一定策略**选择**一个就绪进程运行**.

1.3 目标

1. 响应速度尽可能快：**交互程序：CPU 切换频繁**
2. 进程处理的时间尽可能短：**后台程序：特定进程优先**
3. 系统吞吐量尽可能大：**CPU 有效工作时间多**
4. 资源利用率尽可能高
5. 对所有进程要公平：**部分进程饥饿**
6. 避免饥饿

7. 避免死锁

上述部分原则之间存在自相矛盾!

1.3.1 周转时间/平均周转时间

周转时间，即进程**提交**给计算机到**完成**所花费的时间（t），周转时间说明了进程在系统中停留时间的长短。

t = tc - ts

ts——进程的提交时间（Start）

tc——进程的完成时间（Complete ）

平均周转时间，所有进程的周转时间的平均。平均周转时间越短，意味着这些进程在系统内停留的时间越短，因而**系统吞吐量**也就越大，**资源利用率**也越高。

t = （t1 + t2 + ···+ tn）/ n

1.3.2 带权周转时间/平均带权周转时间

带权周转时间 w，说明了进程在系统中的**相对**停留时间。

w = **周转时间 / 进程运行时间** = t / tr

t：进程的周转时间

tr ： 进程的运行时间（run）

平均带权周转时间。

w = （w1 + w2 + ···+ wn）/ n

1.4 进程调度算法

1.4.1 先来先服务调度（First Come First Serve）

按照作业进入系统的**时间先后次序**来挑选作业。先进入系统的作业优先被运行。

其特点有：

晚来的作业会等待较长时间。

只考虑作业的**等候时间**，而没考虑**运行时间**的长短。因此一个晚来但是很短的作业可能需要等待很长时间才能被运行，因而本算法**不利于短作业**。

1.4.2 短作业优先调度算法（Short Job First Serve）

参考**运行时间**，选取**时间最短**的作业投入运行。

其特点有：

早来的长作业会长时间等待

忽视了作业等待时间，一**个早来但是很长的作业**将会在很长时间得不到调度，易出现资源**“饥饿”**的现象。

1.4.3 响应比高者优先调度算法

调度作业时计算作业列表中每个作业的响应比，选择响应比**最高**的作业优先投入运行。

响应比定义：作业的**响应时间**和与**运行时间**的比值

响应比 = 响应时间/运行时间 =（等待时间 + 运行时间）/运行时间 = 1 + 等待时间 / 运行时间 = 加权周转时间。

特点：

有利于短作业

有利于等候已久的作业。

兼顾长作业

1.4.4 优先数调度算法

根据进程优先数，把 CPU 分配给最高的进程。

进程优先数 = 静态优先数 + 动态优先数

静态优先数进程创建时确定，在整个进程运行期间不再改变。静态优先数的确定基于

1. 进程所需的资源多少（不一定）

2. 基于程序运行时间的长短（长的进程静态优先数应该越小）

3. 基于进程的类型（IO/CPU 中偏向 IO 的进程交互性强，优先数高；前台/后台中偏向前台的进程对于用户体验比较重要，优先数高，核心/用户中用户态进程优先数更高）

动态优先数在进程运行期间可以改变。

1. 当使用 CPU 超过一定时长时：适当降低

2. 当进程等待时间超过一定时长时：适当提高

3. 当进行 I/O 操作后：适当提高

1.4.5 循环轮转调度法（ROUND-ROBIN）

把所有就绪进程按**先进先出**的原则排成队列。**新来进程**加到**队列末尾**。进程以**时间片** q 为单位轮流使用 CPU。刚刚运行了一个时间片的进程排到队列**末尾**，等候下一轮调度。**队列逻辑上是环形的**。

优点：

1. 公平性：每个就绪进程有平等机会获得 CPU

2. 交互性：每个进程等待 (N-1)* q 的时间就可以重新获得 CPU

时间片 q 的大小，如果 q 太大，交互性差。甚至退化为 FCFS 调度算法。如果 q 太小，进程切换频繁，系统开销增加。

改进：时间片的大小可变（**可变时间片轮转调度法**），组织多个就绪队列（**多重时间片循环轮转**）

1.5 进程调度方式

当一进程正在 CPU 上运行时，若有更高优先级的进程需要运行，系统**如何分配 CPU**。

1.5.1 非抢占方式

让正在运行的进程继续执行，直到该进程**完成或发生某事件**而进入**“完成”或“阻塞”**状态时，才把 CPU 分配给新来的更高优先级的进程。

1.5.2 抢占方式

当更高优先级的进程来到时，便**暂停**正在运行的进程，**立即**把 CPU 分配给新来的优先级更高的进程。

1.6 Linux 调度机制

1.6.1 宏观评价

基于**优先级调度**。支持普通进程，也支持实时进程；实时进程优先于普通进程；普通进程公平使用 CPU 时间。

1.6.2 关键参量 priority, counter, rt_priority, policy

以下变量均定义在 task_struck 结构体中。（以 Linux0.11 为例）

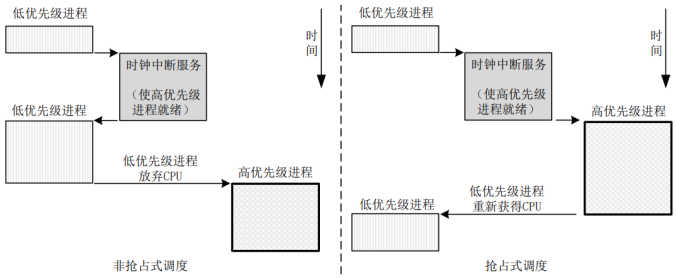

图 2: 非抢占方式与抢占方式的对比

优先级 = priority + counter

priority：进程（包括实时和普通）的静态优先级

counter：进程在当前一轮调度中还能连续运行的时间片数量。较高优先级的进程有更大的 counter，counter 初值 = priority。counter 的改变：时钟中断服务程序：counter，所有进程的 counter 都减到 0 后，重新开始新一轮调度：counter 初值 = priority。

policy 指明进程使用何种调度策略，用来区分实时进程和普通进程。

```
/*
 * Scheduling policies
 */
#define SCHED_OTHER     0   普通的分时进程
#define SCHED_FIFO      1   先进先出的实时进程
#define SCHED_RR        2   基于优先级轮转的实时进程
```

图 3: 进程使用何种调度策略

rt_priority 实时进程特有的优先级: rt_priority+1000

1.6.3 nice 指令

Linux nice 命令以更改过的优先序来执行程序，如果未指定程序，则会印出目前的排程优先序。

priority = priority - nice

nice -n 数字进程

nice 范围：-20（最高）~ 19（最低）

默认 nice = 0

普通用户：可以调整自己进程，而 nice 范围 [0, 19]。

Root 用户：可以调整任何进程，而 nice 范围 [-20, 19]。

1.6.4 调度函数的实现

调度函数 schedule 在可运行队列中找到一个进程给它分配 CPU。

调用时机：

直接调度：时钟中断，即 do_timer()，当资源无法满足被阻塞时，sleep_on()。

间接调度/松散调度：进程从内核态返回到用户态前。