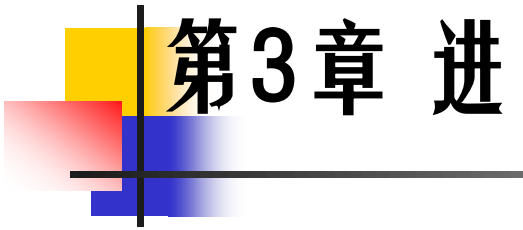


第3章 进程调度（Process Scheduling）

3.1 进程调度的功能与目标

3.2 进程调度的方式与时机

3.3 进程调度算法

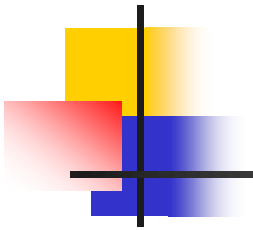
3.1 进程调度的功能与目标

一、进程调度的功能

1. 进程调度要做什么？

当多个进程就绪时，OS的调度程序（Scheduler）决定先运行哪一个。

- ✓ 记录进程的状态 - 借助于PCB;
- ✓ 选择投入运行的进程 - 依赖于调度策略（算法）；
- ✓ 进行进程的上下文切换。



进程调度的功能

2. 进程调度要考虑的问题 - 处理机调度

- ✓ 按什么原则分配CPU：需要确定调度方式和算法
- ✓ 何时分配CPU：需要确定调度时机
- ✓ 如何分配CPU：进程的上下文切换（CPU调度过程）

3.1 进程调度的功能与目标

二、进程调度程序要达到的目标（选择调度算法的准则）

（1）公平：确保每个进程获得合理的CPU份额。

（2）CPU利用率高：尽可能使CPU 100%忙碌。

（3）响应时间短。

响应时间指的是交互式用户从输入到有结果输出之间的时间。

（4）周转时间短。

周转时间 = 作业完成时间 – 作业提交时间

（5）吞吐量大。

吞吐量指的是单位时间内完成的作业数。

（6）调度算法不宜太复杂，以免占用太多的CPU时间。

3.2 进程调度的方式与时机

一、进程切换

暂停一个进程，运行另一个进程。

进程切换的时机：OS靠中断获得CPU的控制权

（1）硬件中断

- ✓ 时钟中断：OS检查时间片是否到。若时间片到，则转入Ready，否则当前进程继续运行；
- ✓ I/O中断：若是某个进程等待的事件，则将其由阻塞->就绪。然后，决定是否继续运行当前进程还是让更高优先级的就绪进程剥夺之。

（2）进程异常

由OS处理，决定是否终止之（若是致命错误），还是继续运行或切换。

（3）请求OS服务（陷入）

如请求I/O，在启动I/O后通常置当前进程为阻塞。

3.2 进程调度的方式与时机

二、调度方式

（1）非抢占方式（Non-preemptive mode）:非剥夺

一旦某进程被调度，直到其完成或等待某事件而阻塞，才会切换到其他进程。

（2）抢占方式（Preemptive mode）:可剥夺

允许暂停正在运行的进程，切换到其他进程。

抢占的原则：

- ✓时间片原则：时间片到时抢占
- ✓优先级原则：优先级高者到时抢占

3.2 进程调度的方式与时机

三、引起进程调度的时机

（1）当前进程终止：执行完毕或出现错误而结束

（2）当前进程阻塞：等待I/O完成，执行了阻塞原语

（3）当前进程执行的时间片到

（4）出现了一个更高优先级的就绪进程

进程中断/异常/系统调用返回到用户态时

3.3 进程调度算法

- 引入多道程序系统的直接目的就是想让处理机“忙” 。所以当处理机空闲时，系统需要从多个就绪进程中挑选一个使其投入运行。选择哪一个呢？这需要一种算法。调度算法的实质就是一种**资源分配**。
- 从资源的角度来看，该算法确定了处理机的分配策略，故称其为处理机调度算法；
- 而从资源使用者的角度看，该算法确定了进程运行的次序，故也称进程调度算法。

3.3 进程调度算法

一、先来先服务（First Come First Serve, FCFS）

或称先进先出进程调度算法（FIFO）

按照进程就绪的先后次序来调度进程

优点：实现简单

缺点:

（1）未考虑进程的优先级

（2）有利于CPU繁忙型进程，不利于I/O繁忙型进程

（3）有利于长进程，不利于短进程

3.3 进程调度算法

二、时间片轮转（Round Robin, RR）

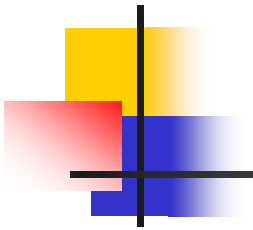
实现方法：

将所有的就绪进程按FCFS原则排成一个队列，

规定一个时间片为进程每次使用CPU的最长时间，

每次选择队首进程运行，

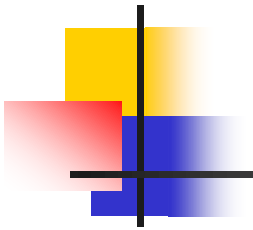
当时间片到时，剥夺该进程的运行，将其排在队尾。



时间片轮转

选择多大的时间片合适？大一些好还是小一些好？

- ✓ 如果太小，则导致频繁的进程切换，消耗CPU时间
- ✓ 如果太大，则响应时间长，大到一个进程足以完成其全部运行工作所需的时间时，就退化为FCFS模式。



时间片轮转

选择时间片大小要考虑的因素：

（1）对响应时间的要求

（2）就绪进程数

（3）系统的处理能力

基本要求：

保证用户键入的常用命令能在1个时间片内处理完。

响应时间 = 进程数 * 时间片

3.3 进程调度算法

三、短进程（作业）优先

选择估计运行时间最短的进程运行。

缺点：

对长进程（作业）不利。

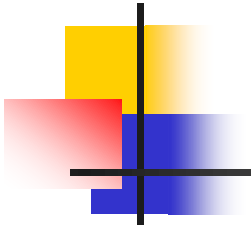
极端情况下，会使长进程（作业）得不到调度。

如何知道哪个是最短的？

3.3 进程调度算法

四、基于优先级（优先数）的调度

- ✓ 每个进程一个优先级；
- ✓ 总是选择就绪队列中优先级最高的进程投入运行；
- ✓ 可以是抢占式，或非抢占式。



基于优先级的调度

优先级的确定：

（1）静态优先级

在进程创建时确定，在进程的运行期间保持不变。

很可能出现优先级低者永远得不到调度的情况。

（2）动态优先级

在进程创建时指定一个基础优先级，

每隔一定时间（如一个时钟中断），重新计算优先级。例如：

等待时间越长，优先级越高；

已占用CPU时间越长，优先级越低。

3.3 进程调度算法

五、多级反馈队列

(1) 按优先级设置n(n>1)个就绪队列，每个队列赋予不同优先级。如第一级队列到最后一级队列，优先级依次降低。

(2) 优先级越高队列，分配的时间片越小。为什么？

(3) 每个队列按照先来先服务原则排队。

(4) 一个新进程就绪后进入第一级（或相应优先级）队尾。

(5) **调度方法**

- ✓ 每次选择优先级最高队列的队首进程运行；
- ✓ 若被调度进程的时间片到，则放入下一级队列的队尾；
- ✓ 最后一级队列采用时间片轮转；
- ✓ 当有一个优先级更高的进程就绪时，可以抢占CPU，被抢占进程回到原来队列的末尾。

多级反馈队列

该算法综合了前面几种算法的优点。

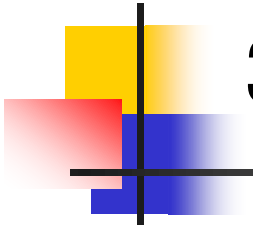
既考虑了先来先服务，又照顾了长进程；

既考虑了优先级，又讲求公平。

说明：

在实际系统中，可能会采取更复杂的动态优先级调度策略。

比如，当一个进程等待时间较长时，提升到上一级队列。

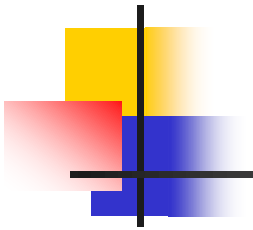


3.3 进程调度算法

六、实时OS的调度

对时间要求严格。

一般基于开始截止时间、完成截止时间。



本章作业

教材《计算机操作系统教程（第4版）》

p100：

4.6