上海交通大学
SHANGHAI JIAO TONG UNIVERSITY

SJTU

L4-1. CPU调度

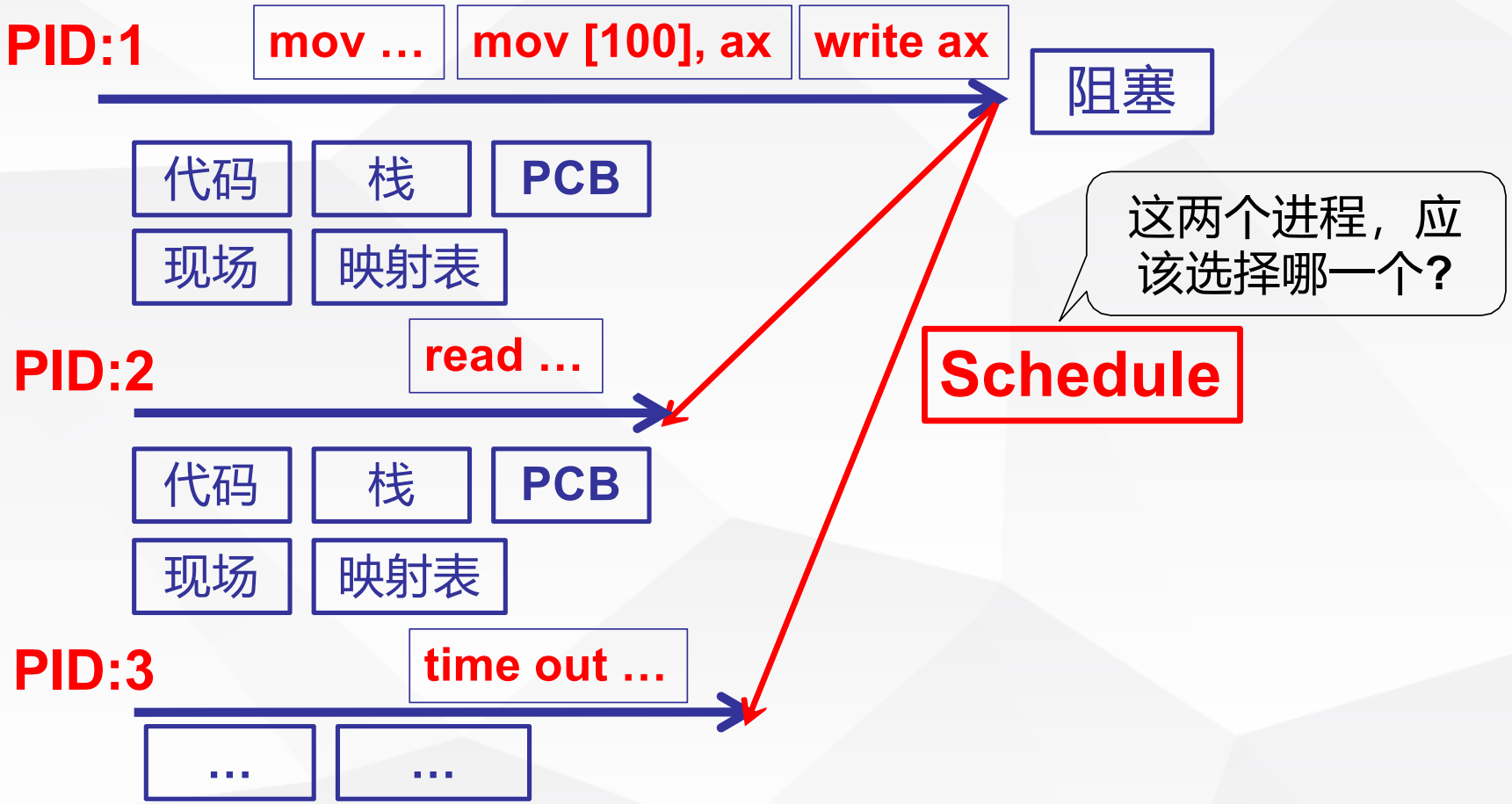
宋卓然

上海交通大学计算机系

songzhuoran@sjtu.edu.cn

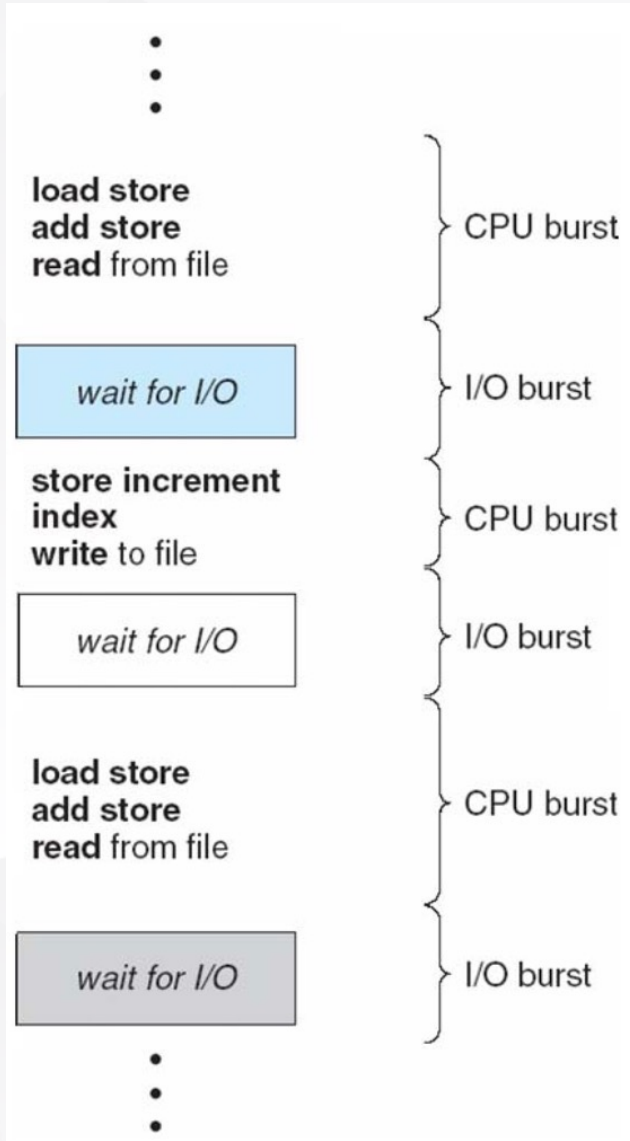
饮 水 思 源 • 爱 国 荣 校

多进程图像与CPU调度



基本概念

- 最大化CPU利用率
- 进程执行包括周期（cycle）进行CPU执行和I/O等待，两种状态来回切换



CPU调度器

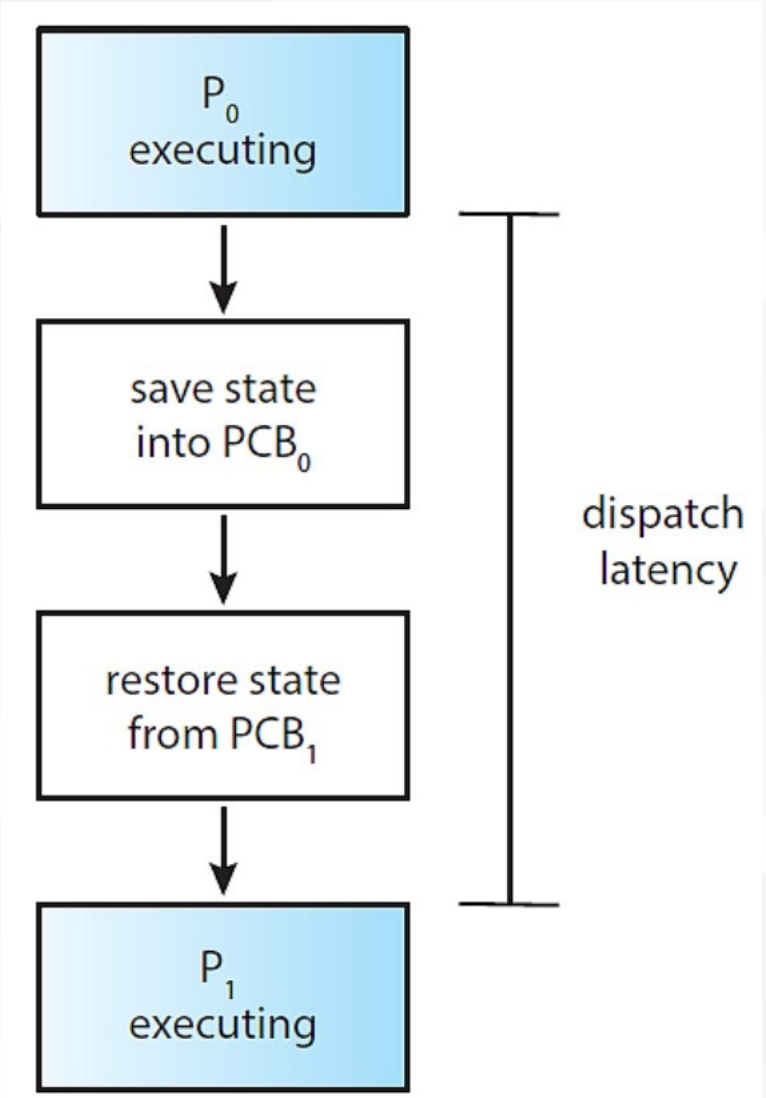
- 从就绪队列中的进程中进行选择，并将CPU分配给其中一个进程
- 需要进行CPU调度的情况可分为以下四种：
  - 当一个进程从运行状态切换到等待状态时(例如，I/O请求，或wait()调用以便等待一个子进程的终止)
  - 当一个进程从运行状态切换到就绪状态时(例如，当出现中断时)
  - 当一个进程从等待状态切换到就绪状态时(例如，I/O完成)
  - 当一个进程终止时
- 如果调度只能发生在第1、4两种情况，调度为非抢占的调度（主动调度）
- 否则为抢占的调度

CPU调度器

- 非抢占调度
  - 一旦某个进程分配到CPU，该进程就会一直使用CPU，直到它终止或切换到等待状态（Windows 3.x）
- 抢占调度
  - 时间片到了则强制调度另一进程使用CPU（包括Windows 95之后的系统）
- 抢占调度可能导致数据竞争（共享数据、更新，修改内核中的数据时被抢占）
  - 等待系统调用完成后切换进程
  - 对受中断影响的代码段加以保护

调度程序

- 调度程序是一个模块， 用来将CPU控制交给由短期调度程序选择的进程
  - 切换上下文
  - 切换到用户模式
  - 跳转到用户程序的合适位置，以便重新启动程序
- 调度程序停止一个进程而启动另一个所需的时间称为调度延迟（希望尽可能快）



CPU调度(进程调度)的直观想法

- **FIFO?**
  - 谁先进入，先调度谁: 简单有效　　银行、食堂
  - 一个只简单询问业务的人该怎么办?
- **Priority?**
  - 任务短可以适当优先　　但你怎么知道这个任务将来会执行多长时间呢?
  - 这人的询问越来越长怎么办?
  - 那如果一个银行业务很长是因为客户需要填写一个很长的表，该怎么办?

应该还有很多这样的询问...

调度准则

- CPU利用率： 使CPU尽可能忙碌
- 吞吐量：一个时间单元内进程完成的数量
- 周转时间：从进程提交到进程完成的时间段称为周转时间。周转时间为所有时间段之和，包括等待进入内存、在就绪队列中等待、在CPU上执行和I/O执行
- 等待时间：在就绪队列中等待所花时间之和
- 响应时间：从提交请求到产生第一次响应的时间

最大化CPU利用率、吞吐量，最小化周转时间、等待时间和响应时间

面对诸多场景，如何设计调度算法？

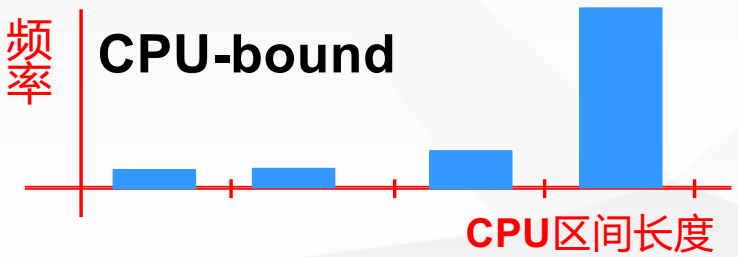
- 我们的算法应该让什么更好？
  - 面对客户：银行调度算法的设计目标应该是　用户满意
  - 面对进程：CPU调度的目标应该是　进程满意
- 那怎么才能让进程满意呢？　时间…
  - 尽快结束任务：周转时间(从任务进入到任务结束)短
  - 用户操作尽快响应：响应时间(从操作发生到响应)短
  - 系统内耗时间少：吞吐量(完成的任务量)
- 总原则：系统专注于任务执行，又能合理调配任务…

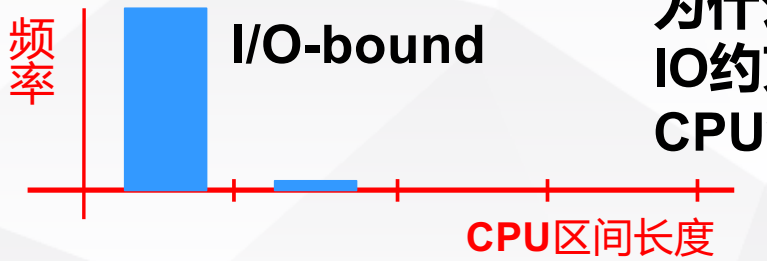
如何做到合理？需要折中，需要综合…

- 吞吐量和响应时间之间有矛盾…
  - 响应时间小⇒切换次数多⇒系统内耗大⇒吞吐量小

这样的矛盾还有很多…

- 前台任务和后台任务的关注点不同…
  - 前台任务关注响应时间，后台任务关注周转时间
- **IO约束型任务和CPU约束型任务有各自的特点**





IO约束的优先级往往高于CPU约束任务
为什么?
IO约束：前台任务
CPU约束：后台任务

折中和综合让操作系统变得复杂，
但有效的系统又要求尽量简单…

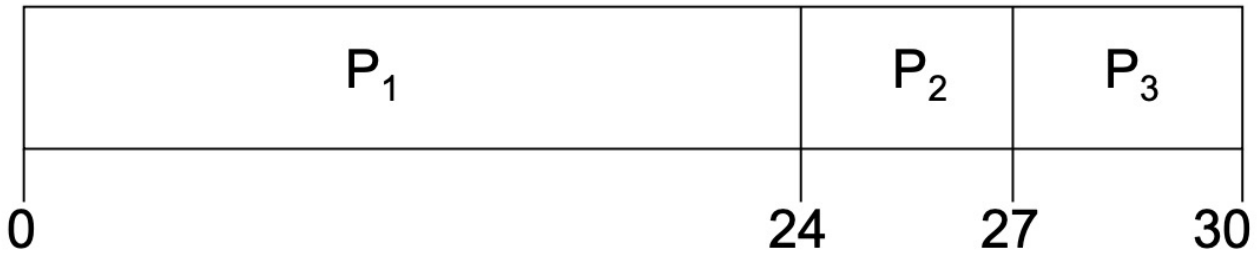
调度算法

- 先到先服务调度 First-Come, First-Served (FCFS) Scheduling
- 最短作业优先调度 Shortest-Job-First (SJF) Scheduling
- 优先级调度 Priority Scheduling
- 轮转调度 Round-Robin Scheduling
- 多级队列调度 Multilevel Queue Scheduling
- 多级反馈队列调度 Multilevel Feedback Queue Scheduling

先到先服务调度 (FCFS)

Process	CPU Burst	*Arrival* Time
P_1	24	0
P_2	3	1
P_3	3	2

- 甘特图：

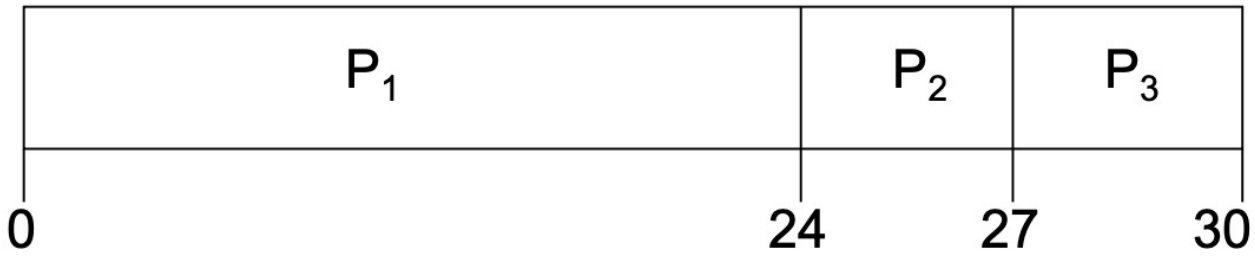


- 等待时间：P1 = 0; P2 = 24-1=23; P3 = 27-2=25
- 平均等待时间：(0 + 23 + 25)/3 = 16

先到先服务调度 (FCFS)

Process	CPU Burst	*Arrival* Time
P_1	24	0
P_2	3	1
P_3	3	2

- 甘特图：

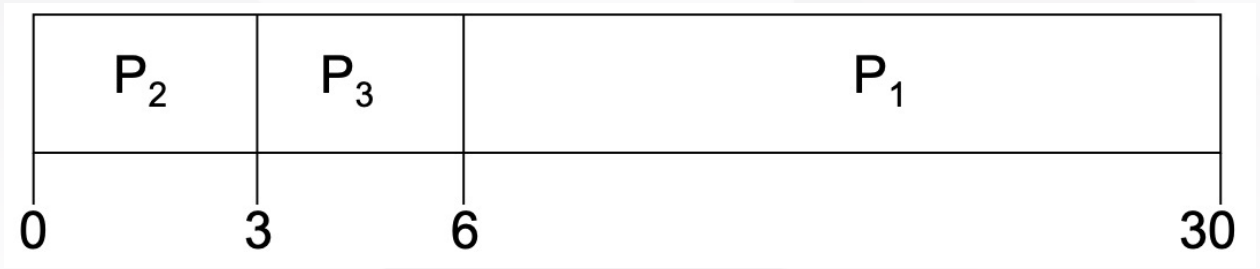


- 周转时间：P1 = 24; P2 = 27-1=26; P3 = 30-2=28
- 平均周转时间：(24 + 26 + 28)/3 = 26

先到先服务调度 (FCFS)

Process	CPU Burst	*Arrival* Time
P_1	24	2
P_2	3	0
P_3	3	1

- 甘特图：



- 等待时间：P1=4;P2 =0;P3=2
- 平均等待时间：(4 + 0 + 2)/3 = 2
- 非抢占的

最短作业优先调度 (SJF)

- 将进程与其下次CPU执行的长度关联起来。当CPU为空闲时，它会被赋给具有最短CPU执行的进程
- SJF是最优的，可以获得最短的平均周转时间
- 如果调度结果为p1、p2、…、pn，则平均周转时间为

  p1+p1+p2+p1+p2+p3+…=$\sum(n+1-i)pi$=np1+(n-1)p2+(n-2)p3+…
- 显然，p1<p2<…，所以周转时间最小
- 难点在于知道每个进程使用CPU的时间

预测下次CPU执行时间

- 下次CPU执行时间与之前的执行时间相似
- 以前CPU执行的测量长度的指数平均：

$$\tau_{n+1} = \alpha t_n + (1 - \alpha)\tau_n$$

- t_n为最近的第n个时刻CPU的执行时间； τ_n存储了过去历史；参数α控制最近和过去历史在预测中的权重，通常被设置为$\frac{1}{2}$

预测下次CPU执行时间

$$\tau_{n+1} = \alpha t_n + (1 - \alpha)\tau_n$$

- 展开公示：

$$\tau_{n+1} = \alpha t_n + (1 - \alpha)\alpha t_n - 1 + \ldots$$
$$+ (1 - \alpha)^j \alpha t_{n-j} + \ldots$$
$$+ (1 - \alpha)^{n+1} \tau_0$$

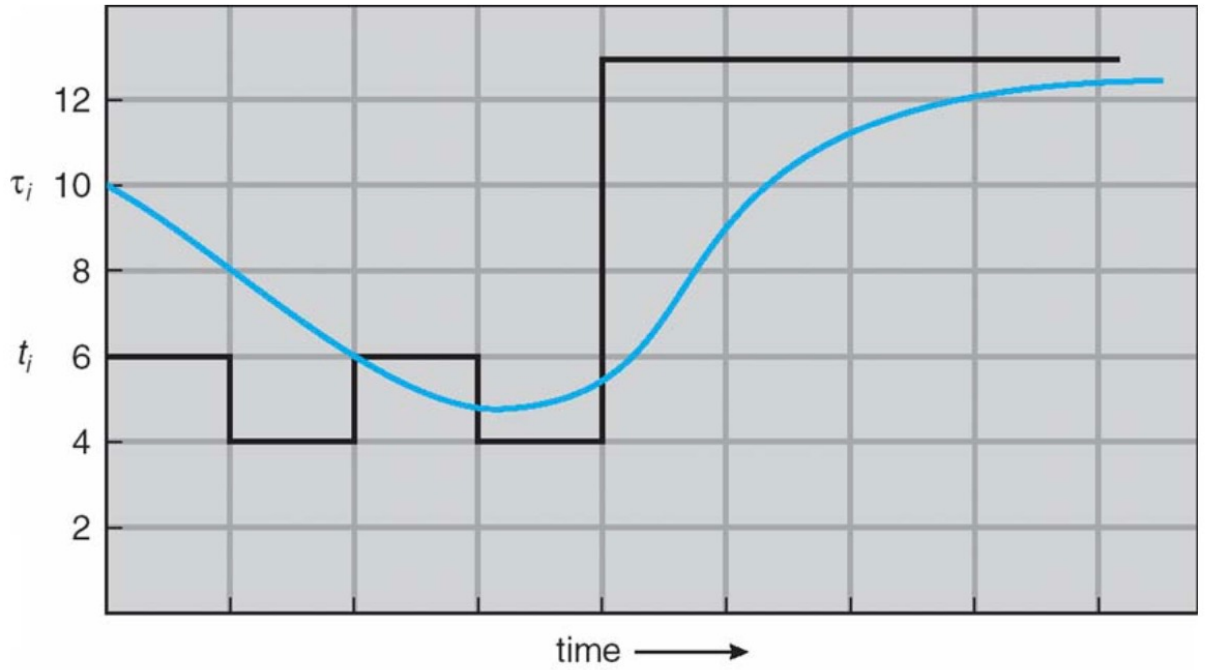
- 由于α和$1 - \alpha$都小于等于1，说明每个连续项的权重都比其前一个项小，越临近的CPU执行越重要

预测下次CPU执行时间

- $\alpha = 0$
  - 表示最近历史没有影响(当前情形为瞬态)
- $\alpha = 1$
  - 只有量近CPU执行才重要(过去历史被认为是陈旧的、无关的)
- $\alpha = \frac{1}{2}$
  - 最近的和历史的都重要

预测下次CPU执行时间

- 例子，$\alpha = \frac{1}{2}$、$\tau_0 = 10$:

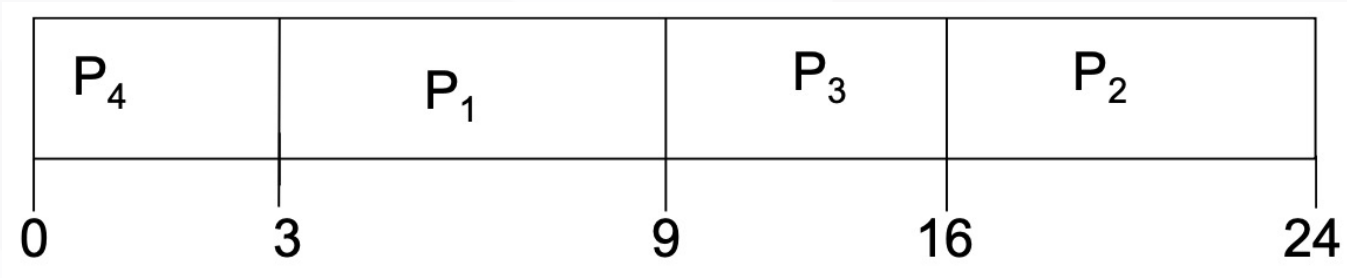


CPU burst (t_i)	6	4	6	4	13	13	13	...
"guess" (τ_i)	10	8	6	6	5	9	11	12	...

最短作业优先调度 (SJF) 例子

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

- 甘特图



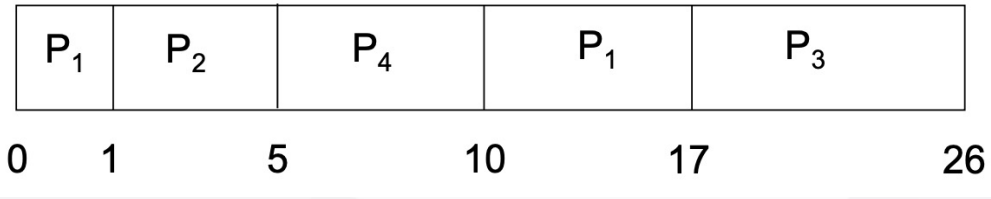
- 平均等待时间：(3+16+9+0)/4=7

最短作业优先调度 (SJF) 例子

- 最短作业优先调度可以是抢占的或非抢占的
- 抢占式：也被称为最短剩余时间优先调度（shortest-remaining-time-first）

Process	*Arrival* Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

- 甘特图



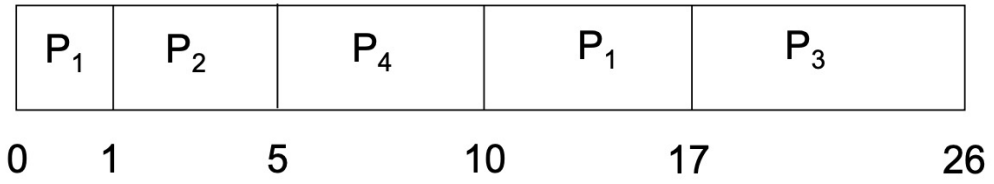
- 平均等待时间?

最短作业优先调度 (SJF) 例子

- 最短作业优先调度可以是抢占的或非抢占的
- 抢占式：也被称为最短剩余时间优先调度（shortest-remaining-time-first）

Process	*Arrival* Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

- 甘特图



- 平均等待时间：[(10-1)+(1-1)+(17-2)+(5-3)]/4 = 26/4 = 6.5

优先级调度

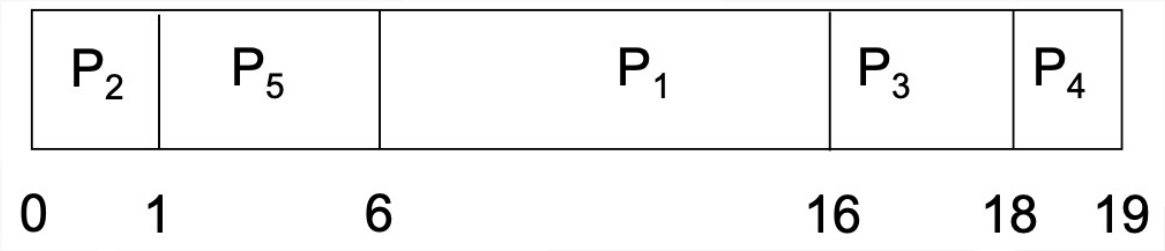
- 每个进程都有一个优先级与之关联
- 具有最高优先级的进程会分配到CPU，具有相同优先级的进程按FCFS顺序调度（用低数字表示高优先级）
  - 抢占
  - 非抢占
- SJF是一个简单的优先级调度算法， 其优先级是下次（预测的）CPU执行的倒数
- 主要问题：饥饿，低优先级的进程可能无限等待
- 解决方法：老化，增加等待时间长的进程的优先级

在1973年关闭MIT的IBM7094时，发现有一个低优先级进程早在1967年就已提交，但是一直未能运行

优先级调度 例子

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

- 甘特图



- 平均等待时间：(1+6+16+18)/5=8.2

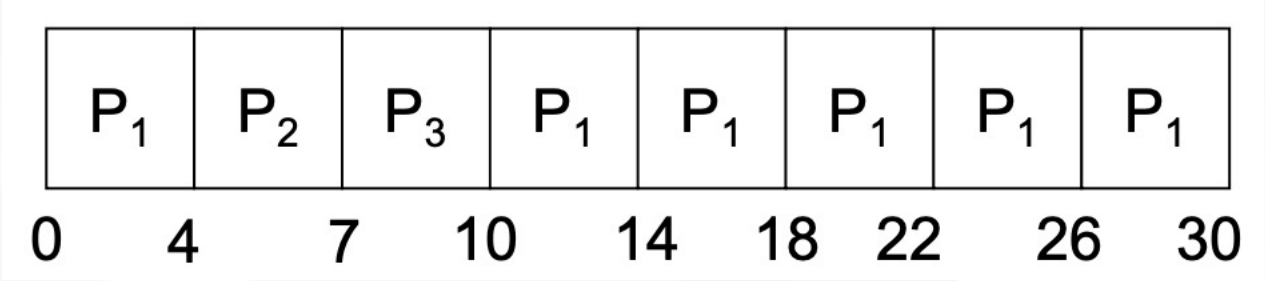
轮转调度（round-robin,RR）

- 类似于FCFS调度， 但是增加了抢占以切换进程，保证了响应时间
- 将一个较小时间单元定义为时间量(time quantum)或时间片(timeslice)
- 时间片的大小通常为10~100ms，经过此时间后，进程将被抢占并添加到就绪队列的末尾
- 有两种情况可能发生。进程可能只需少于时间片的CPU执行。对于这种情况，进程本身会自动释放CPU。调度程序接着处理就绪队列的下一个进程。否则，如果当前运行进程的CPU 执行大于一个时间片，那么定时器会中断，进而中断操作系统。然后，进行上下文切换， 再将进程加到就绪队列的尾部，接着CPU调度程序会选择就绪队列内的下一个进程。

轮转调度（round-robin,RR）例子

Process	Burst Time
P_1	24
P_2	3
P_3	3

- 采用4ms的时间片，甘特图为：

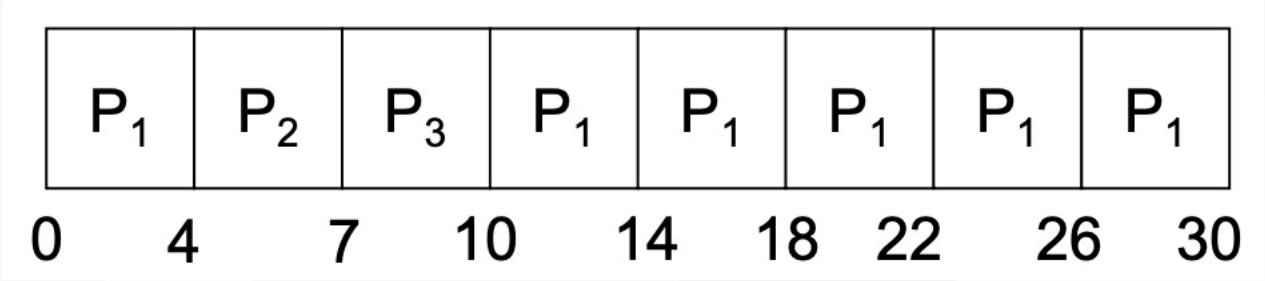


- 响应时间=(0+4+7)/3=3.67ms

轮转调度（round-robin,RR）例子

Process	Burst Time
P_1	24
P_2	3
P_3	3

- 采用4ms的时间片，甘特图为：

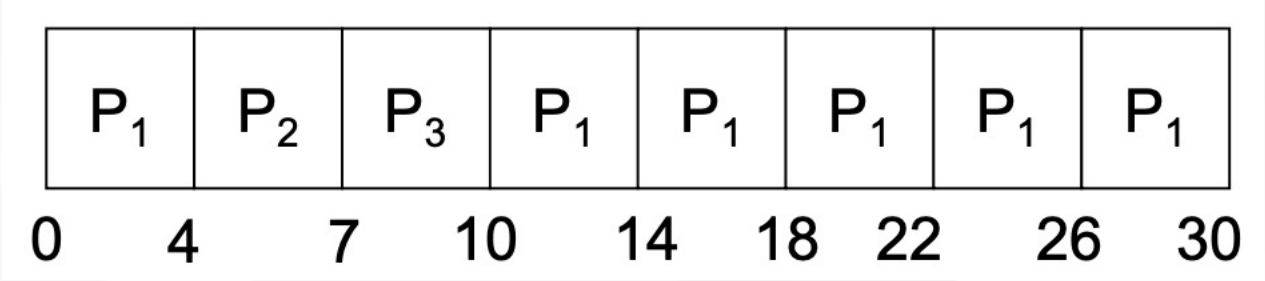


- 等待时间?

轮转调度（round-robin,RR）例子

Process	Burst Time
P_1	24
P_2	3
P_3	3

- 采用4ms的时间片，甘特图为：



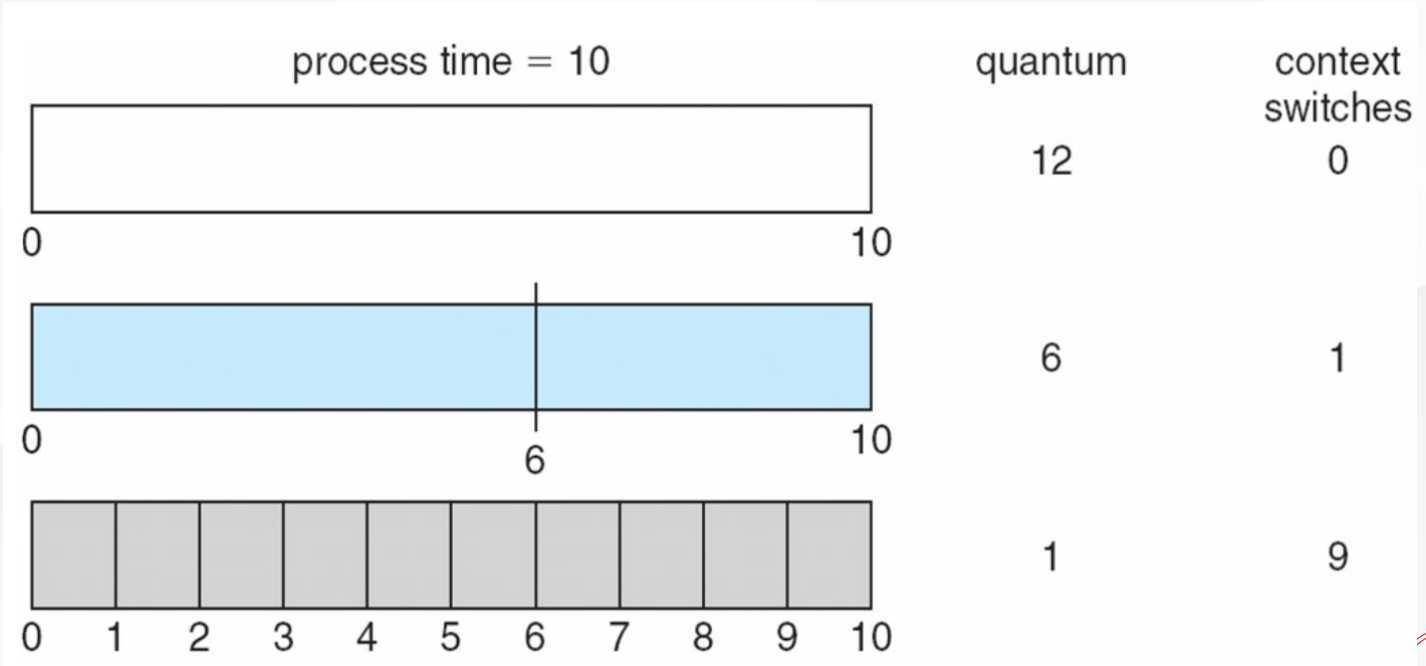
- 等待时间：P1等待10-4=6ms，P2等待4ms，P3等待7ms；平均等待时间为17/3=5.66ms

轮转调度（round-robin,RR）

- 如果就绪队列有n个进程，并且时间片为q，那么每个进程会得到1/n的CPU时间， 而且每次分得的时间不超过q个时间单元。每个进程等待获得下一个CPU时间片的时间不会超过(n-1)q个时间单元。例如，如果有5个进程，并且时间片为20ms，那么每个进程每 100ms 会得到不超过20ms 的时间
- 相较于SJF，RR的周转时间更长，但响应时间更短

轮转调度（round-robin,RR）

- 时间片q越大：演变为FCFS调度
- 时间片q越小：上下文切换越频繁，开销过大
- 时间片应远大于上下文切换时间，通常在10ms～100ms，而上下文切换一般小于10ms



多级队列调度

- 就绪队列可被分为多个单独队列
  - 前台队列（负责交互进程）
  - 后台队列（负责批处理进程）
- 每个队列拥有自己的调度算法
  - 前台队列，轮转调度
  - 后台队列，先到先服务调度
- 队列之间拥有调度策略
  - 固定优先级抢占调度，可能导致饥饿
  - 时间片，对队列设置时间片，例如80%给前台队列，20%给后台队列

多级队列调度 固定优先级的例子

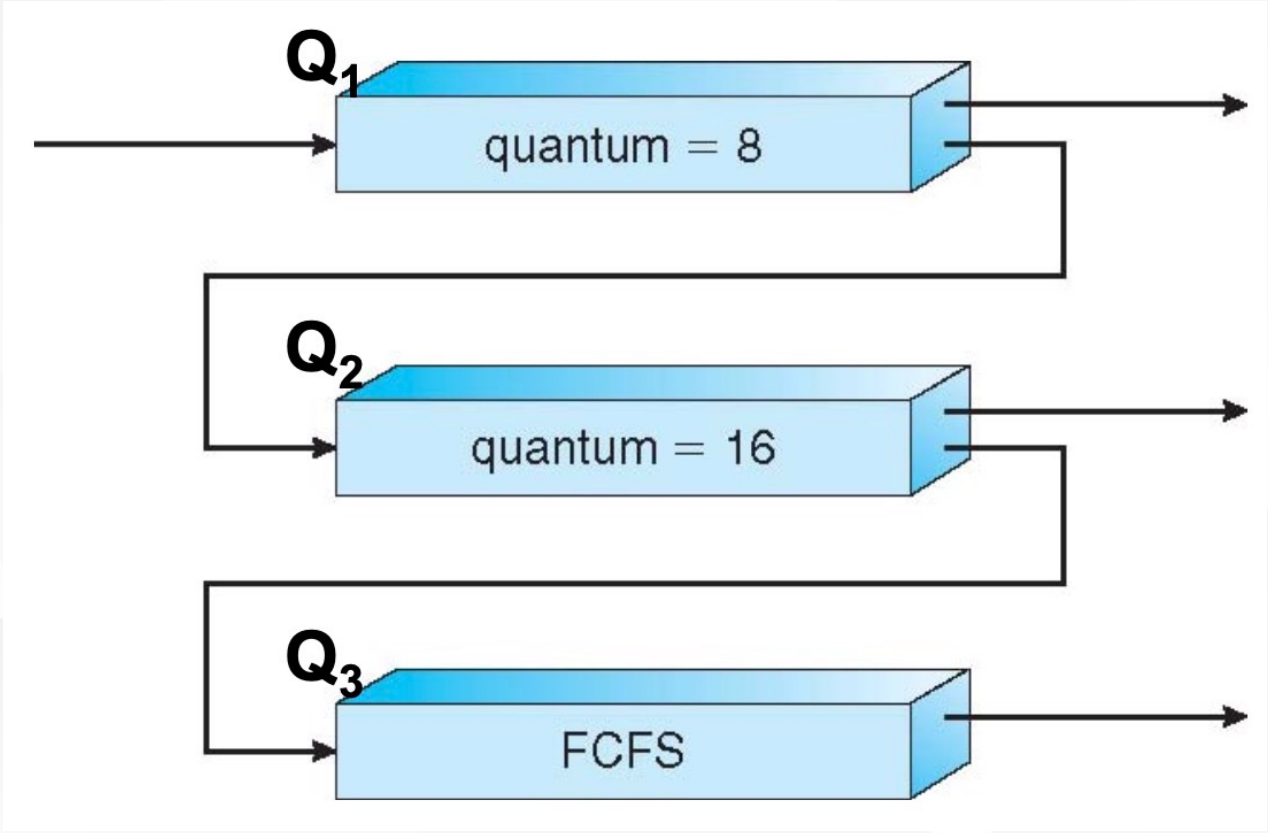
- 5个队列，优先级由高到低
  - 系统进程
  - 交互进程
  - 交互编辑进程
  - 拼处理进程
  - 学生进程
- 每个队列与更低层队列相比具有绝对的优先

多级反馈队列调度

- 多级队列调度的灵活度较低：进程被永久分配到某个队列
- 多级反馈队列调度：允许进程在队列间迁移，在较低优先级队列中等待过长的进程会被移到更高优先级队列，阻止饥饿
- 由以下参数定义：
  - 队列数量
  - 每个队列的调度算法
  - 用以确定何时升级到更高优先级队列的方法
  - 用以确定何时降级到更低优先级队列的方法
  - 用以确定进程在需要服务时将会进入哪个队列的方法

多级反馈队列调度 例子

- 三个队列，从0到2
  - Q1-RR；Q2-RR；Q3-FCFS



多级反馈队列调度 例子

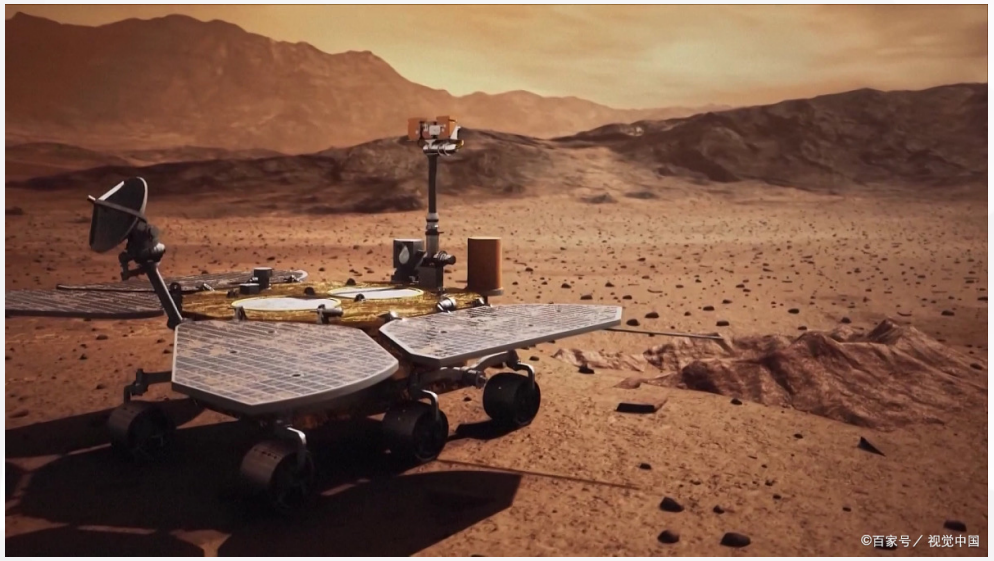
- Q1队列中的进程
  - 具有8ms的时间片
  - 若无法在8ms内完成，则被移动到Q2队列
- Q2队列中的进程
  - 当Q1空时执行，具有16ms的时间片
  - 若无法在16ms内完成，则被移动到Q3队列
- Q3队列中的进程
  - 当Q1、Q2均空时执行，采用FCFS调度

优先级反转

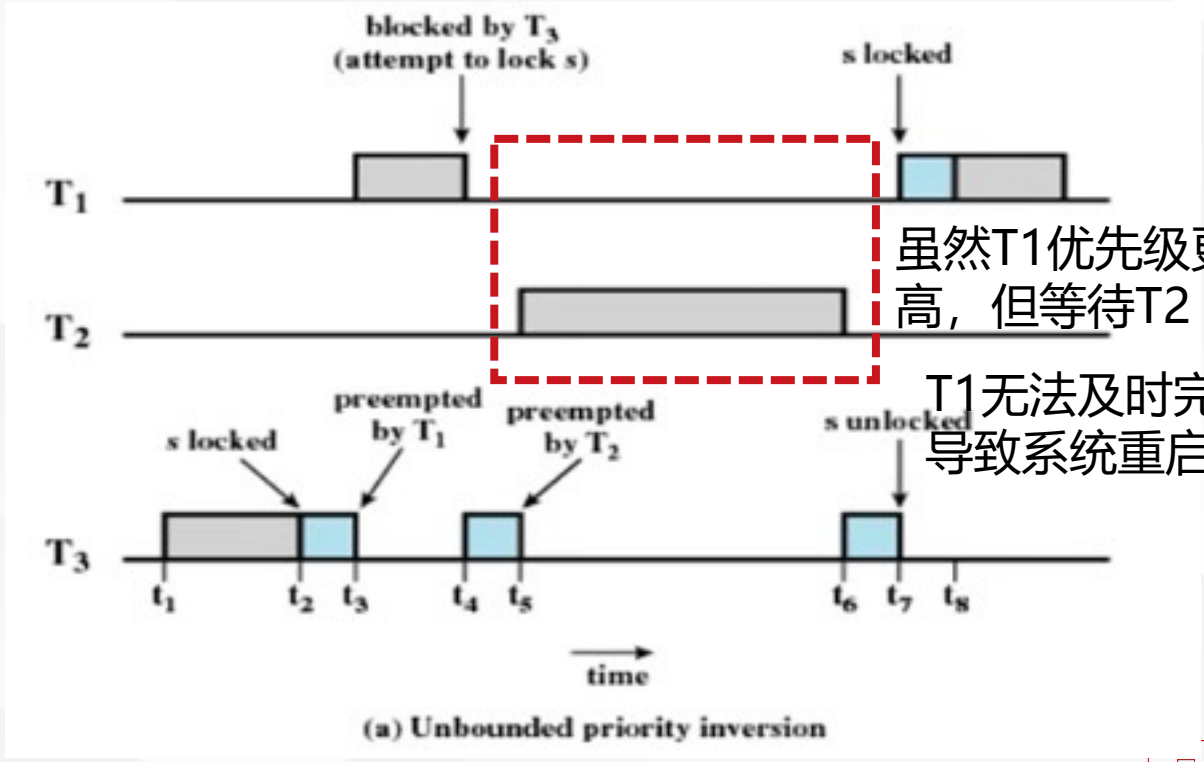
- 可以发生在任何基于优先级的可抢占调度机制中、
- 当系统内环境强制使高优先级等待低优先级时发生

出现优先级反转！

NASA的火星探路者：
操作系统经常重启



进程优先级：T1>T2>T3

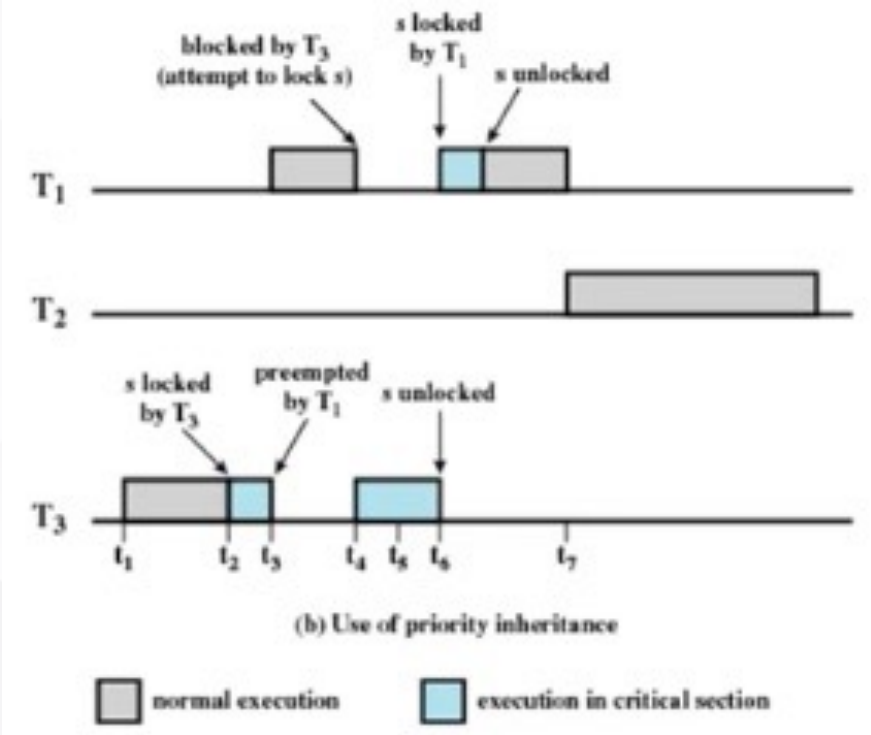


虽然T1优先级更高，但等待T2

T1无法及时完成，导致系统重启

优先级反转解决方案

- 优先级继承协议：正在访问共享资源的进程（T3）获得需要访问共享资源的更高优先级进程（T1）的优先级，直到它使用完成并恢复优先级



课堂习题

1.讨论下列几对调度准则在哪些情况下会冲突：
a.CPU 利用率和响应时间; b.平均周转时间和最大等待时间; c. I/O设备利用率和CPU利用率

2.假设采用指数平均公式来预测下个CPU执行的长度。 当采用如下参数数值时，该算法的含义是什么

$a.\ \ \alpha = 0$ 、$\tau_0 = 100$　　b. $\alpha = 0.99$ 、$\tau_0 = 10$

课堂习题

3.假设有如下一组进程，它们的CPU执行时间以毫秒来计算

进程	执行时间	优先级
P_1	2	2
P_2	1	1
P_3	8	4
P_4	4	2
P_5	5	3

假设进程按P1-P5顺序在时刻0到达。
a.画出4个甘特图，分别演示（FCFS、SJF、非抢占优先级、RR（时间片=2））的进程执行
b.每个进程在a里的每种调度算法下的周转时间是多少?
c.每个进程在a里的每种调度算法下的等待时间是多少?
d. 哪一种调度算法的平均等待时间(对所有进程)最小?