











内容导航:

-  3.1 处理机调度概述
-  3.2 调度算法
-  **3.3 实时调度**
-  3.4 Linux进程调度
-  3.5 死锁概述
-  3.6 预防死锁
-  3.7 避免死锁
-  3.8 死锁的检测与解除

第3章 处理机调度与死锁



实时调度是针对实时任务的调度

实时任务，都联系着一个截止时间

- 硬实时HRT任务
- 软实时SRT任务

实时调度应具备一定的条件



1. 提供必要的信息（向调度程序提供）

- (1) 就绪时间
- (2) 开始截止时间和完成截止时间
- (3) 处理时间
- (4) 资源要求
- (5) 优先级

2. 系统处理能力强

- 实时系统中通常都有着多个实时任务
- 若处理机的处理能力不够强，则有可能因处理机忙不过来而使某些实时任务不能得到及时处理，从而导致发生难以预料的后果
- 假定系统中有 m 个周期性的硬实时任务，它们的处理时间可表示为 C_i ，周期时间表示为 P_i ，则在单处理机情况下，必须满足下面的限制条件：

$$\sum_{i=1}^m \frac{C_i}{P_i} \leq 1$$

3.采用抢占式调度机制

- 当一个优先权更高的任务到达时，允许将当前任务暂时挂起，而令高优先权任务立即投入运行，这样便可满足该硬实时任务对截止时间的要求。这种调度机制比较复杂

- 对于一些小的实时系统，如果能预知任务的开始截止时间，则对实时任务的调度可采用非抢占调度机制，简化调度程序和对任务调度时所花费的系统开销。

但在设计这种调度机制时，应使所有的实时任务都比较小，并在执行完关键性程序和临界区后，能及时地将自己阻塞起来，以便释放出处理机，供调度程序去调度那种开始截止时间即将到达的任务

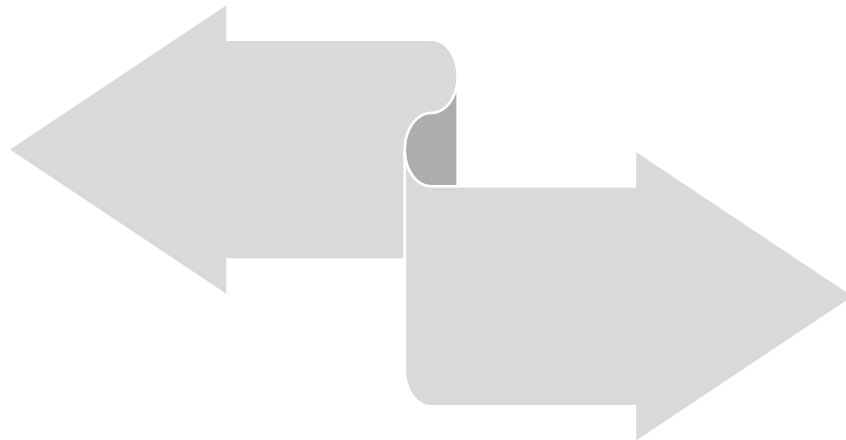
4.具有快速切换机制

- **对中断的快速响应能力**。对紧迫的外部事件请求中断能及时响应，要求系统具有快速硬件中断机构，还应使禁止中断的时间间隔尽量短，以免耽误时机(其它紧迫任务)
- **快速的任務分派能力**。为了提高分派程序进行任务切换时的速度，应使系统中的每个运行功能单位适当的小，以减少任务切换的时间开销



根据实时任务性质

- HRT调度算法
- SRT调度算法



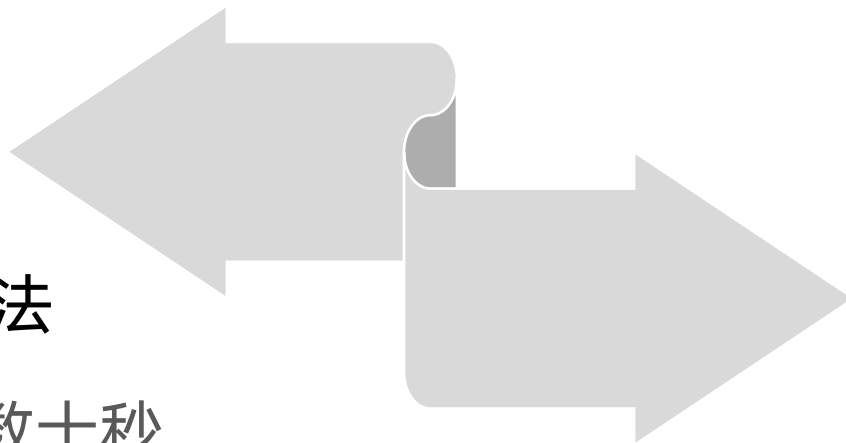
根据调度方式

- 非抢占式调度算法
- 抢占式调度算法



非抢占式轮转调度算法

- 响应时间：数秒至数十秒
- 可用于要求不太严格的实时控制系统



非抢占式优先调度算法

- 响应时间：数秒至数百毫秒
- 可用于有一定要求的实时控制系统



基于时钟中断的抢占式优先级调度

- 响应时间：几十毫秒至几毫秒
- 可用于大多数实时系统



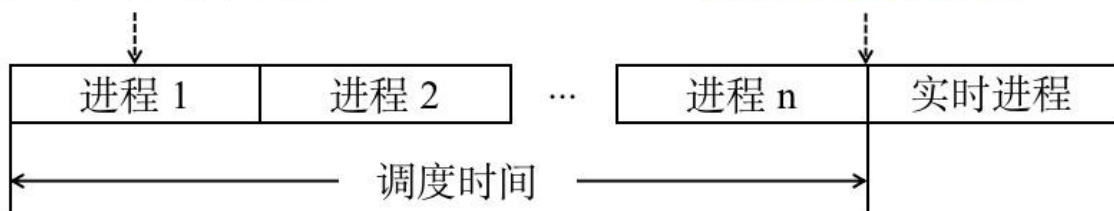
立即抢占的优先级调度

- 响应时间：几毫秒至几百微秒
- 可用于有严格时间要求的实时系统



实时进程请求调度

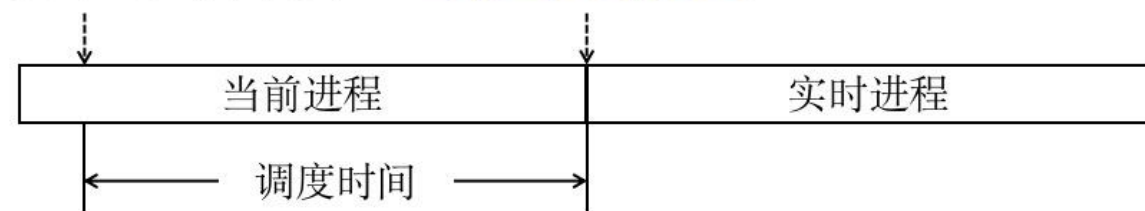
调度实时进程运行



(a) 非抢占式轮转调度

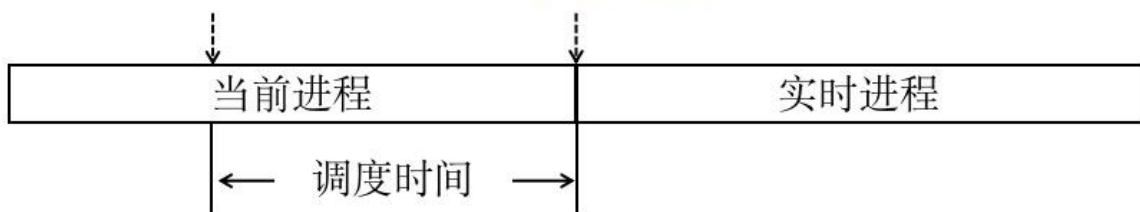
实时进程请求调度

当前进程运行完成



(b) 非抢占式优先级调度

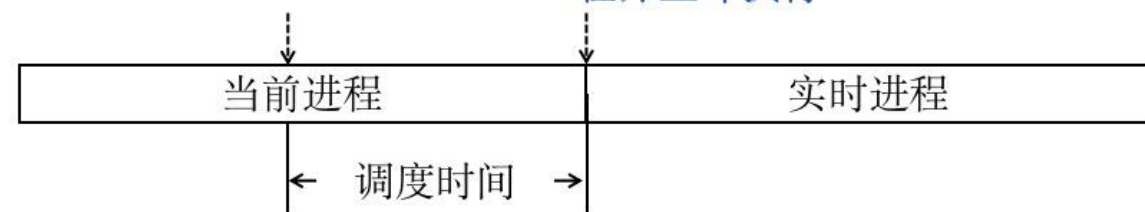
实时进程请求调度 时钟中断到来



(c) 基于时钟中断的抢占式优先级调度

实时进程请求调度

实时进程抢占当前进程并立即执行



(d) 立即抢占的优先级调度



最早截止时间优先(EDF)调度算法



EDF根据任务的截止时间确定优先级，截止时间越早，优先级越高



既可用于抢占式调度，也可用于非抢占式调度

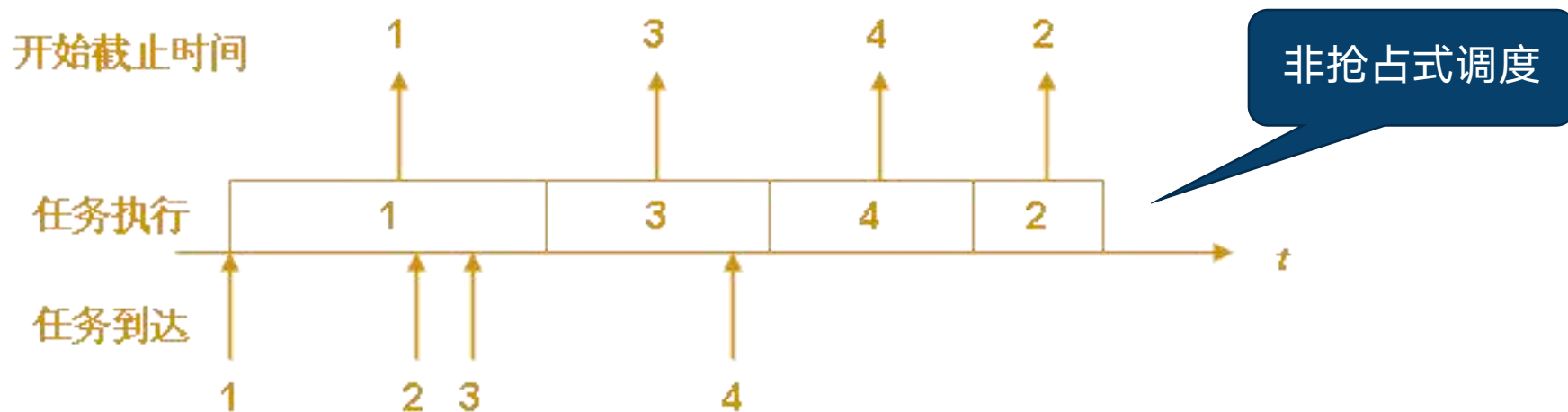


非抢占式调度用于非周期实时任务

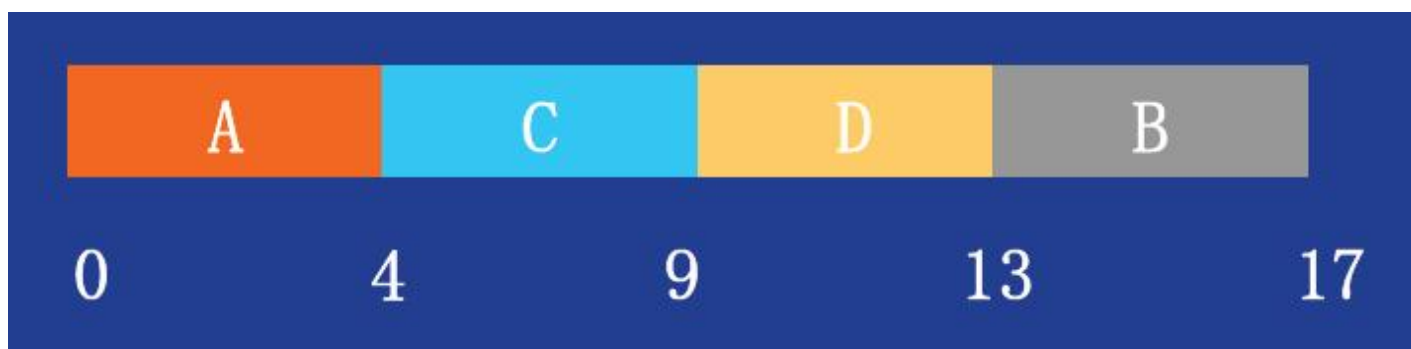


抢占式调度用于周期实时任务

- OS EDF根据任务的截止时间确定优先级，截止时间越早，优先级越高
- OS 既可用于抢占式调度，也可用于非抢占式调度
- OS 非抢占式调度用于非周期实时任务
- OS 抢占式调度用于周期实时任务



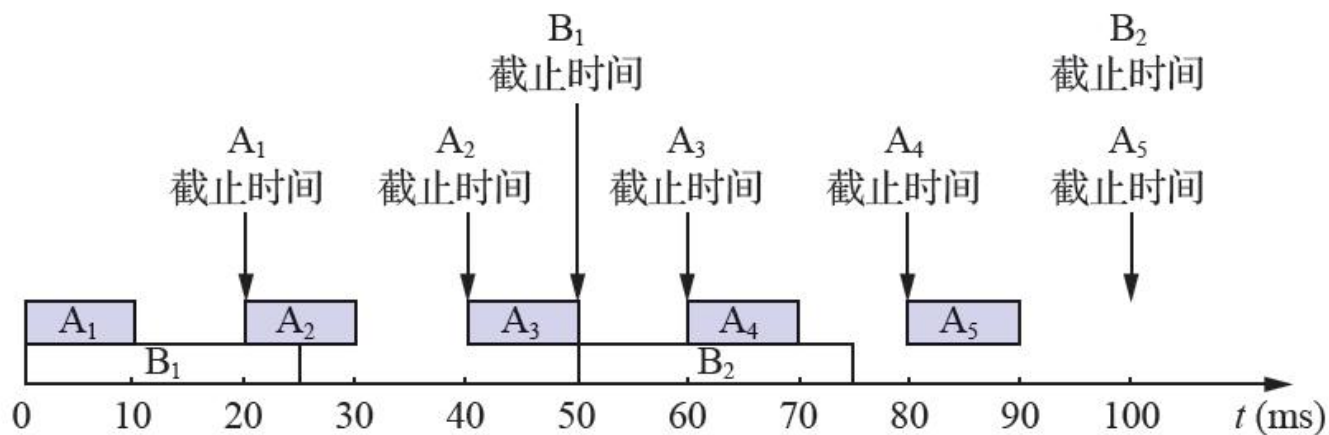
任务	到达时间	开始截止时间	执行时间
A	0	2	4
B	2	15	5
C	3	6	5
D	6	10	4



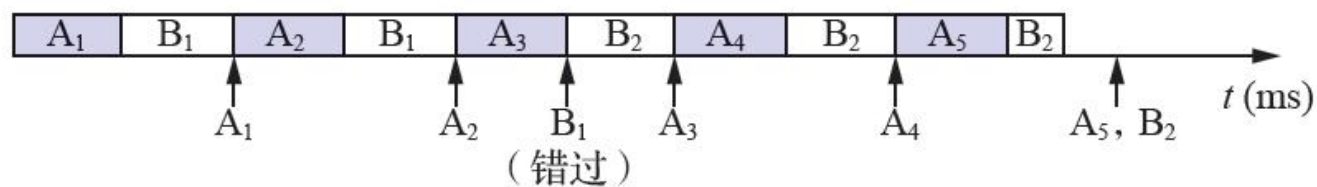


抢占式EDF例子

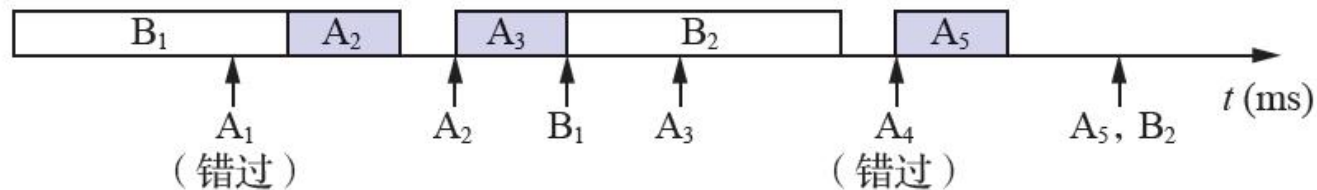
到达时间、执行时间和最后截止时间



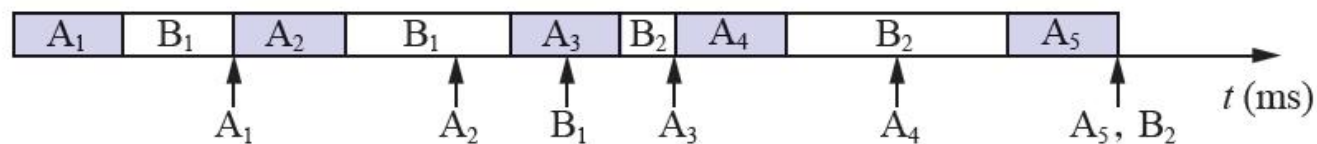
固定优先级调度



固定优先级调度



使用完成截止时间最早和最后截止时间调度





根据任务的紧急程度（**松弛度**）确定任务优先级

- 紧急程度越高（松弛度越低），优先级越高
- $\text{松弛度} = \text{必须完成时间} - \text{其本身的运行时间} - \text{当前时间}$



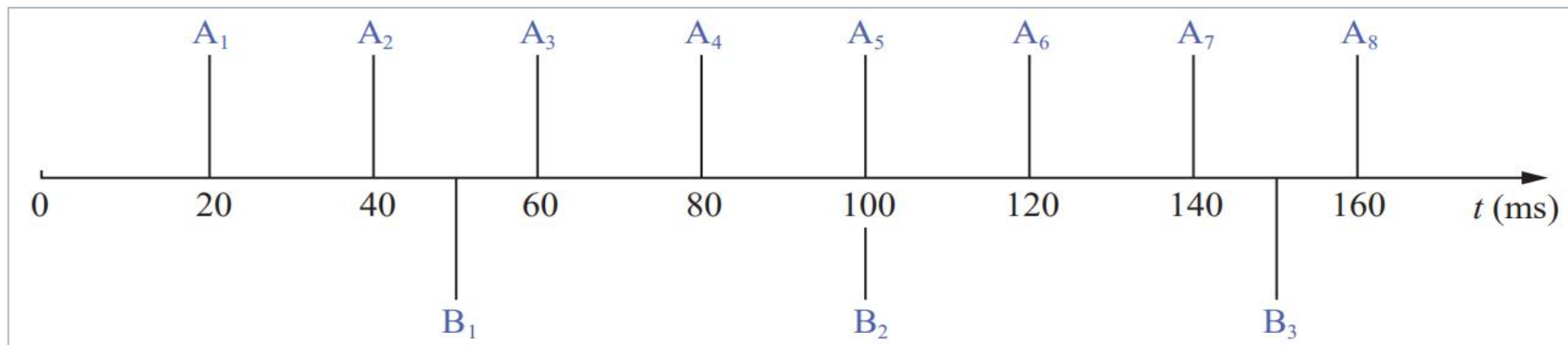
主要用在抢占式调度方式中



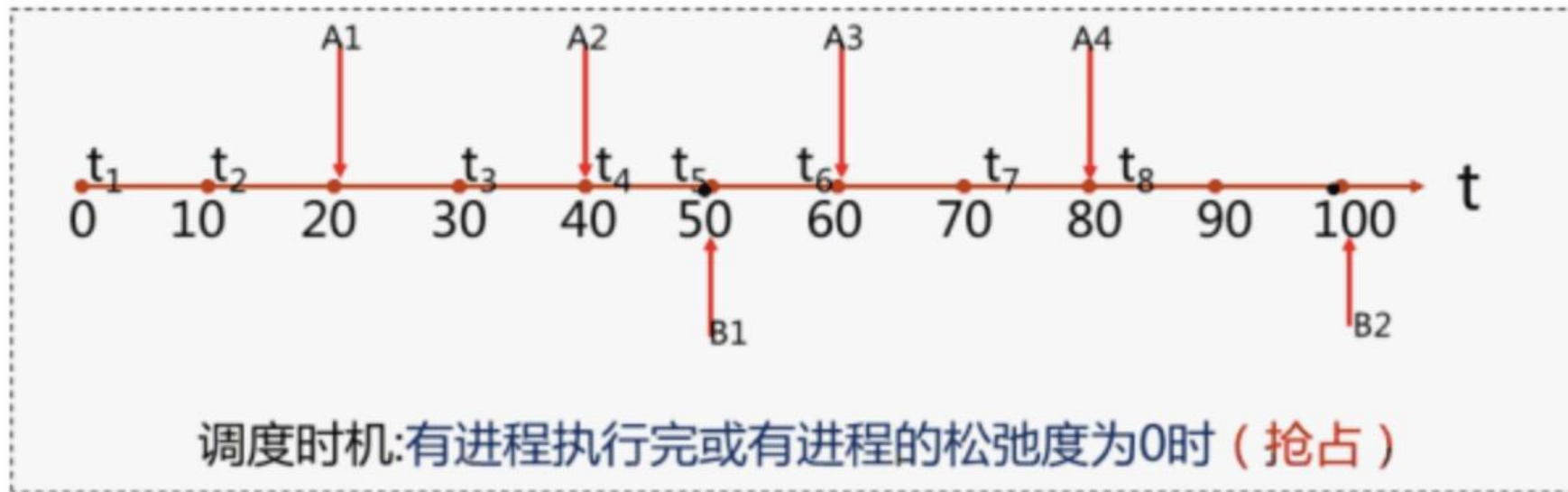
例子：

- 一个任务在 200ms 时必须完成，而它本身所需的运行时间就有 100 ms，因此，调度程序必须在100ms之前调度执行，该任务的紧急程度(松弛程度)为100ms
- 另一任务在400 ms 时必须完成，它本身需要运行150ms，则其松弛程度为250ms

- 两个周期性实时任务A和B，任务A要求每20ms执行一次，执行时间为10 ms，任务B要求每50ms执行一次，执行时间为25 ms



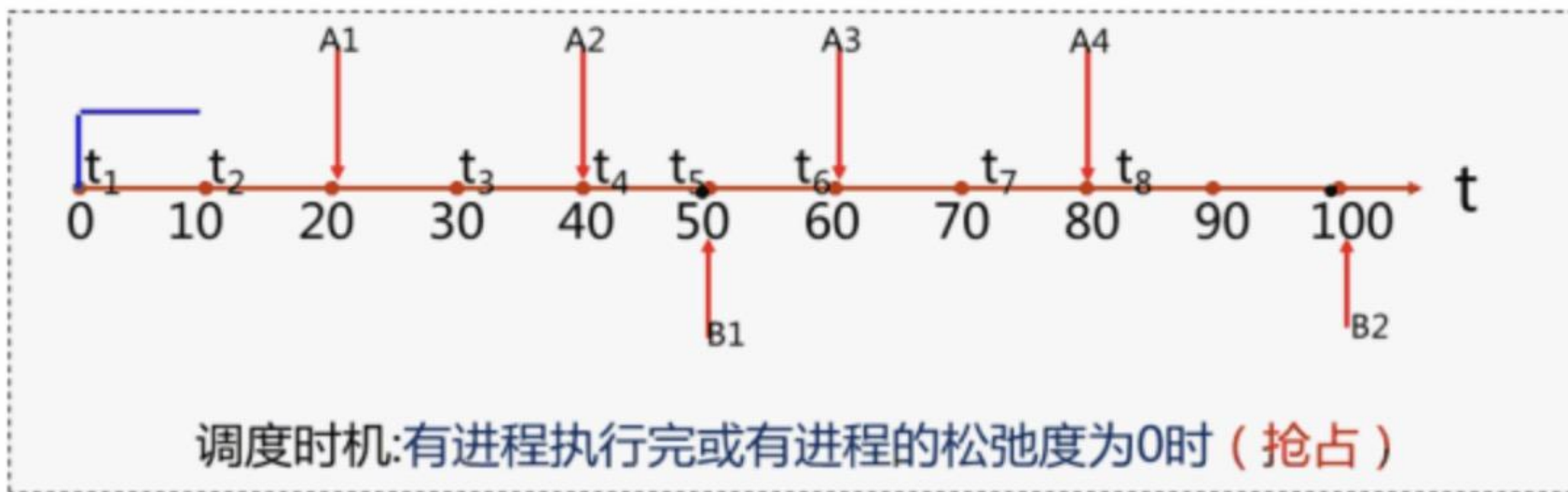
- 两个周期性实时任务A和B，任务A要求每20ms执行一次，执行时间为10 ms，任务B要求每50ms执行一次，执行时间为25 ms



A每次10ms
B每次25ms

初始： $t=0$ 时刻，计算A，B松弛度：
 $A_1 = 20 - 10 - 0 = 10$
 $B_1 = 50 - 25 - 0 = 25$

- 两个周期性实时任务A和B，任务A要求每20ms执行一次，执行时间为10 ms，任务B要求每50ms执行一次，执行时间为25 ms



A每次10ms
B每次25ms

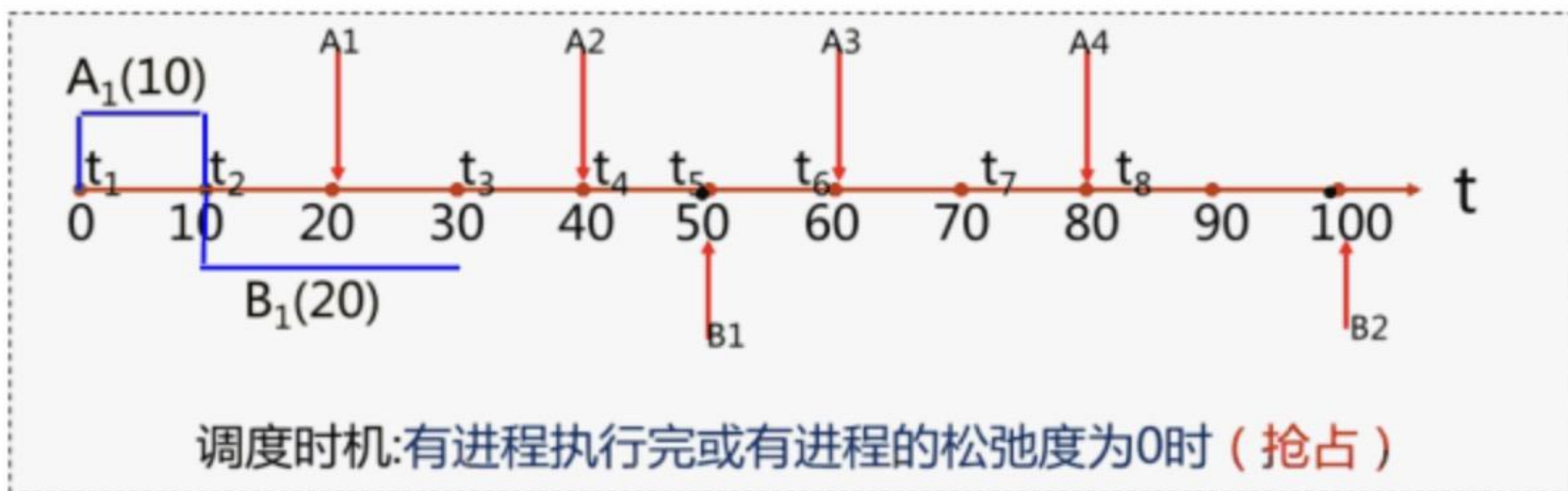
$t=10$ 时刻，计算A，B松弛度：

$$A_2 = 40 - 10 - 10 = 20$$

$$B_1 = 50 - 25 - 10 = 15$$

松弛度 = 必须完成的时间点 - 本身剩余运行时间 - 当前时间

- 两个周期性实时任务A和B，任务A要求每20ms执行一次，执行时间为10 ms，任务B要求每50ms执行一次，执行时间为25 ms

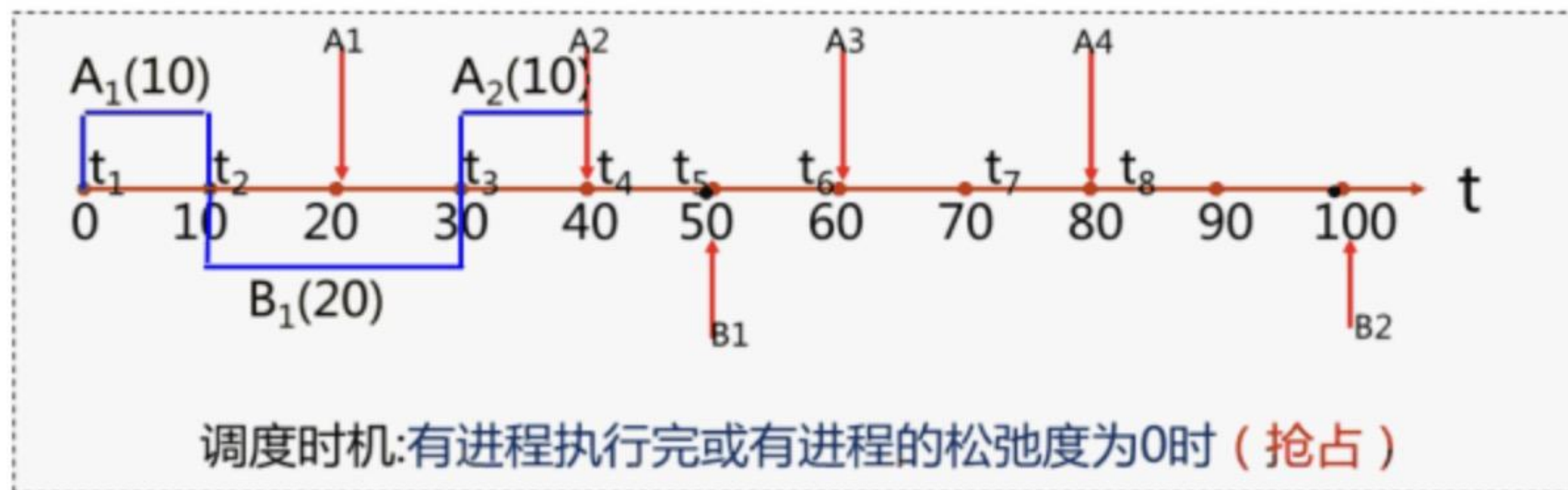


A每次10ms
B每次25ms

$t=30$ 时刻, 计算A, B松弛度:
 $A_2 = 40 - 10 - 30 = 0$
 $B_1 = 50 - 5 - 30 = 15$

松弛度 = 必须完成的时间点 - 本身剩余运行时间 - 当前时间

- 两个周期性实时任务A和B，任务A要求每20ms执行一次，执行时间为10 ms，任务B要求每50ms执行一次，执行时间为25 ms



A每次10ms
B每次25ms

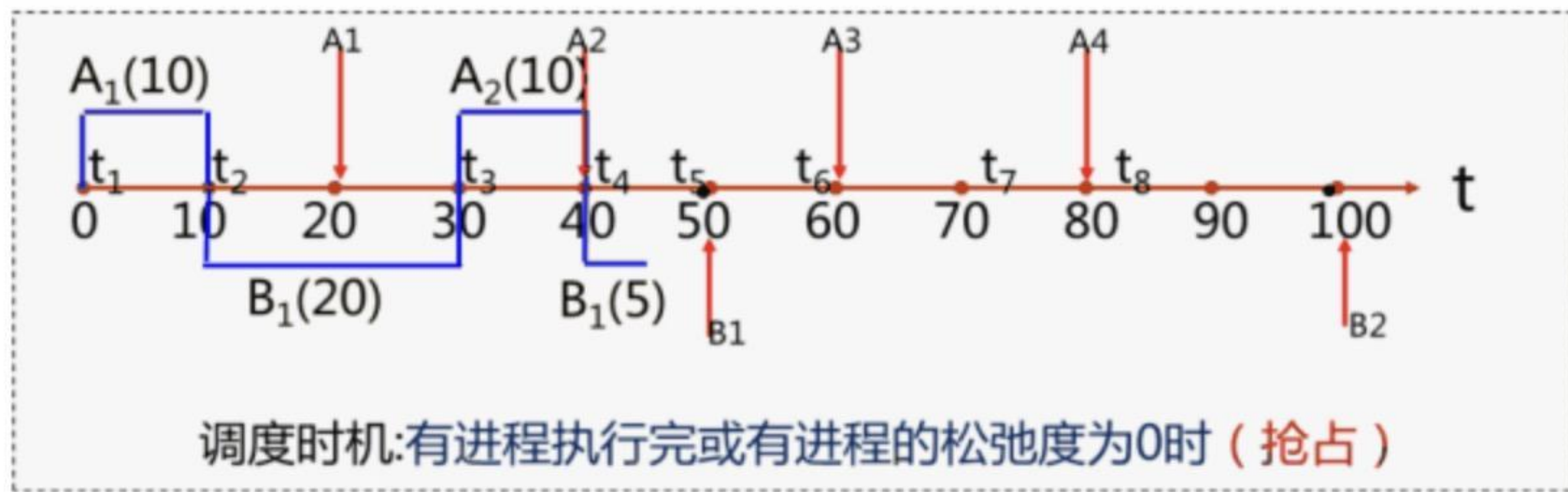
$t=40$ 时刻，计算A，B松弛度：

$$A_3 = 60 - 10 - 40 = 10$$

$$B_1 = 50 - 5 - 40 = 5$$

松弛度 = 必须完成的时间点 - 本身剩余运行时间 - 当前时间

- 两个周期性实时任务A和B，任务A要求每20ms执行一次，执行时间为10 ms，任务B要求每50ms执行一次，执行时间为25 ms



A每次10ms
B每次25ms

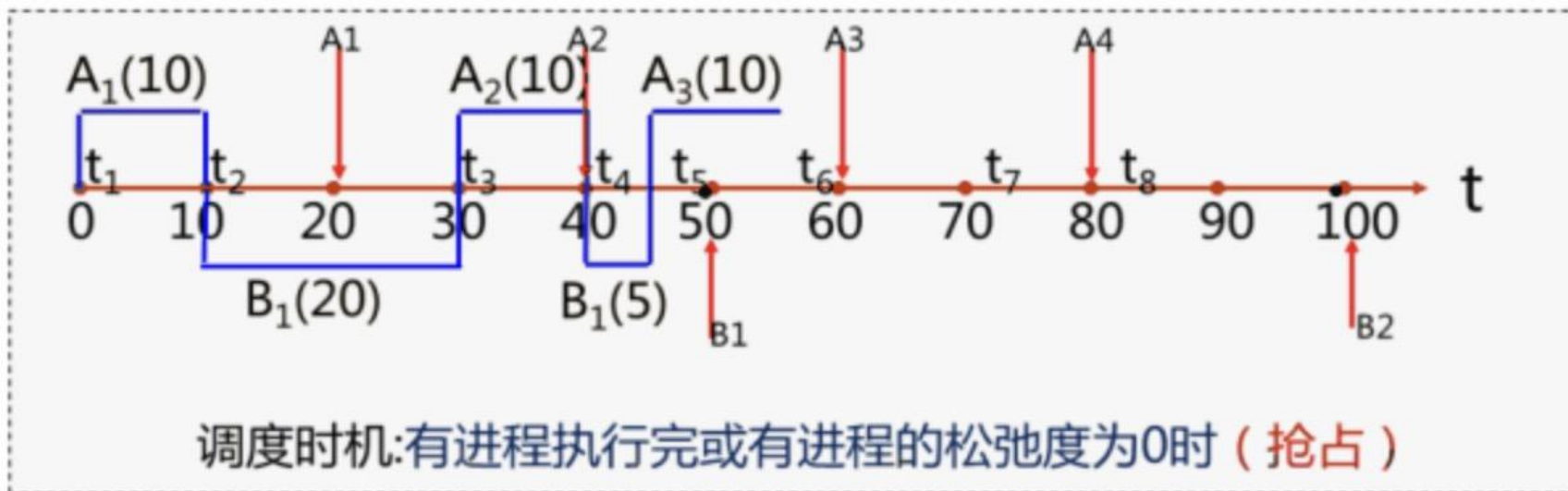
t=45时刻，计算A，B松弛度：

$$B_2 = 100 - 25 - 45 = 20$$

$$A_3 = 60 - 10 - 45 = 5$$

松弛度 = 必须完成的时间点 - 本身剩余运行时间 - 当前时间

- 两个周期性实时任务A和B，任务A要求每20ms执行一次，执行时间为10 ms，任务B要求每50ms执行一次，执行时间为25 ms



A每次10ms
B每次25ms

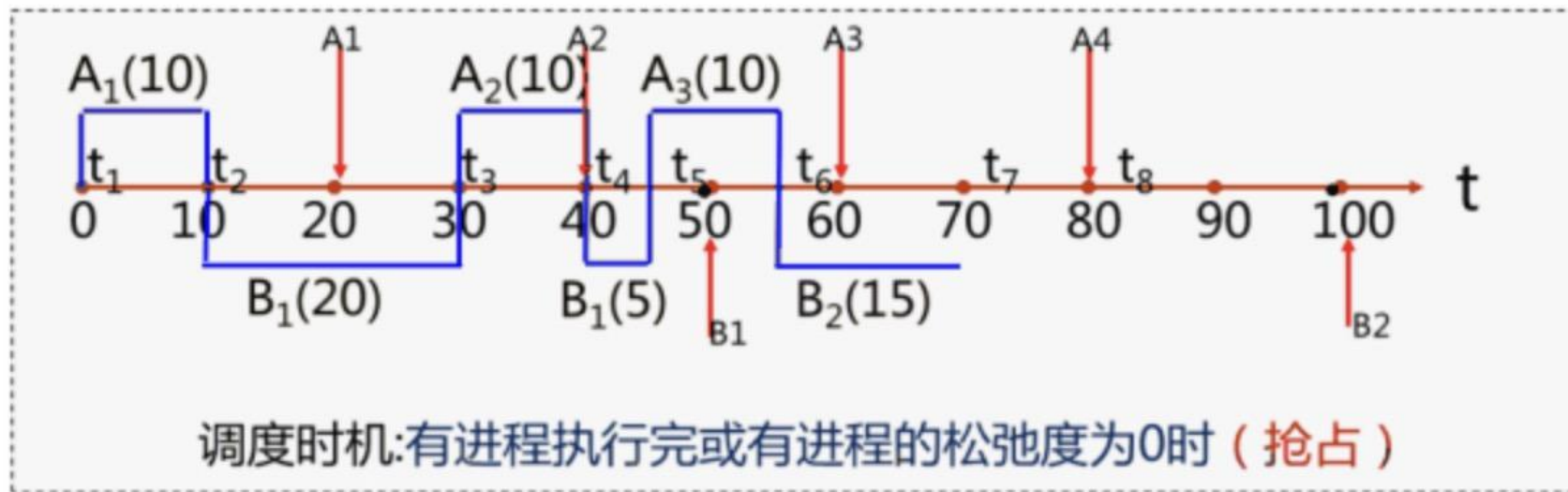
t=55时刻，计算A，B松弛度：

$$A_4 = 80 - 10 - 55 = 35$$

$$B_2 = 100 - 25 - 55 = 20$$

松弛度 = 必须完成的时间点 - 本身剩余运行时间 - 当前时间

- 两个周期性实时任务A和B，任务A要求每20ms执行一次，执行时间为10 ms，任务B要求每50ms执行一次，执行时间为25 ms



A每次10ms
B每次25ms

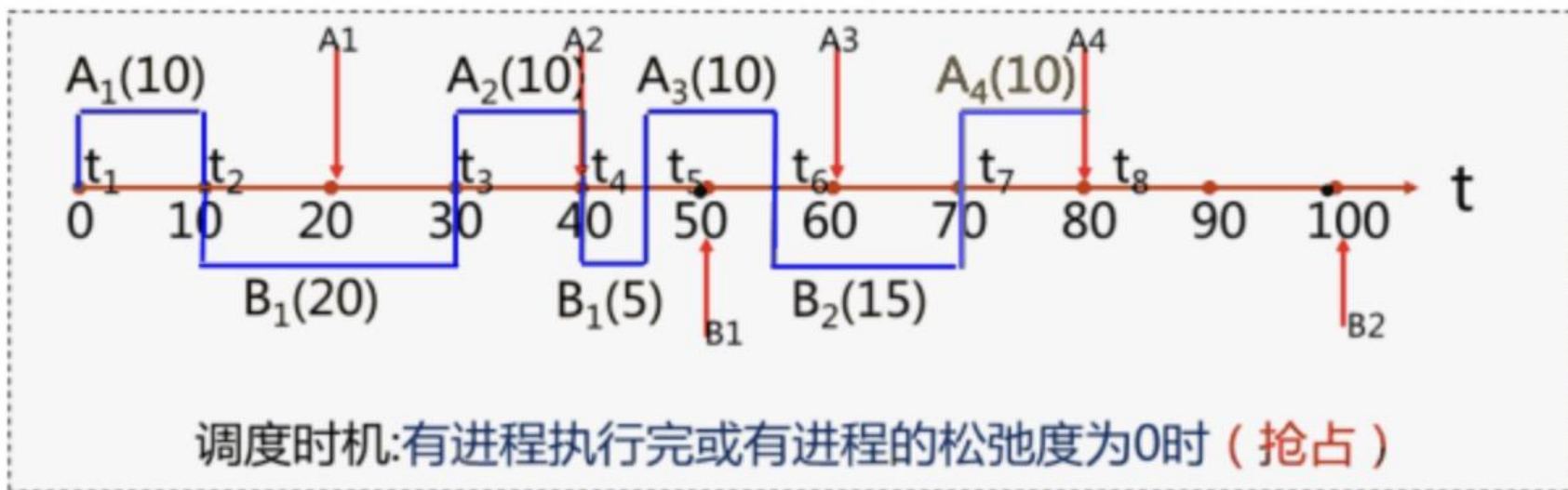
$t=70$ 时刻，计算A，B松弛度：

$$A_4 = 80 - 10 - 70 = 0$$

$$B_2 = 100 - 10 - 70 = 20$$

松弛度 = 必须完成的时间点 - 本身剩余运行时间 - 当前时间

- 两个周期性实时任务A和B，任务A要求每20ms执行一次，执行时间为10 ms，任务B要求每50ms执行一次，执行时间为25 ms



A每次10ms
B每次25ms

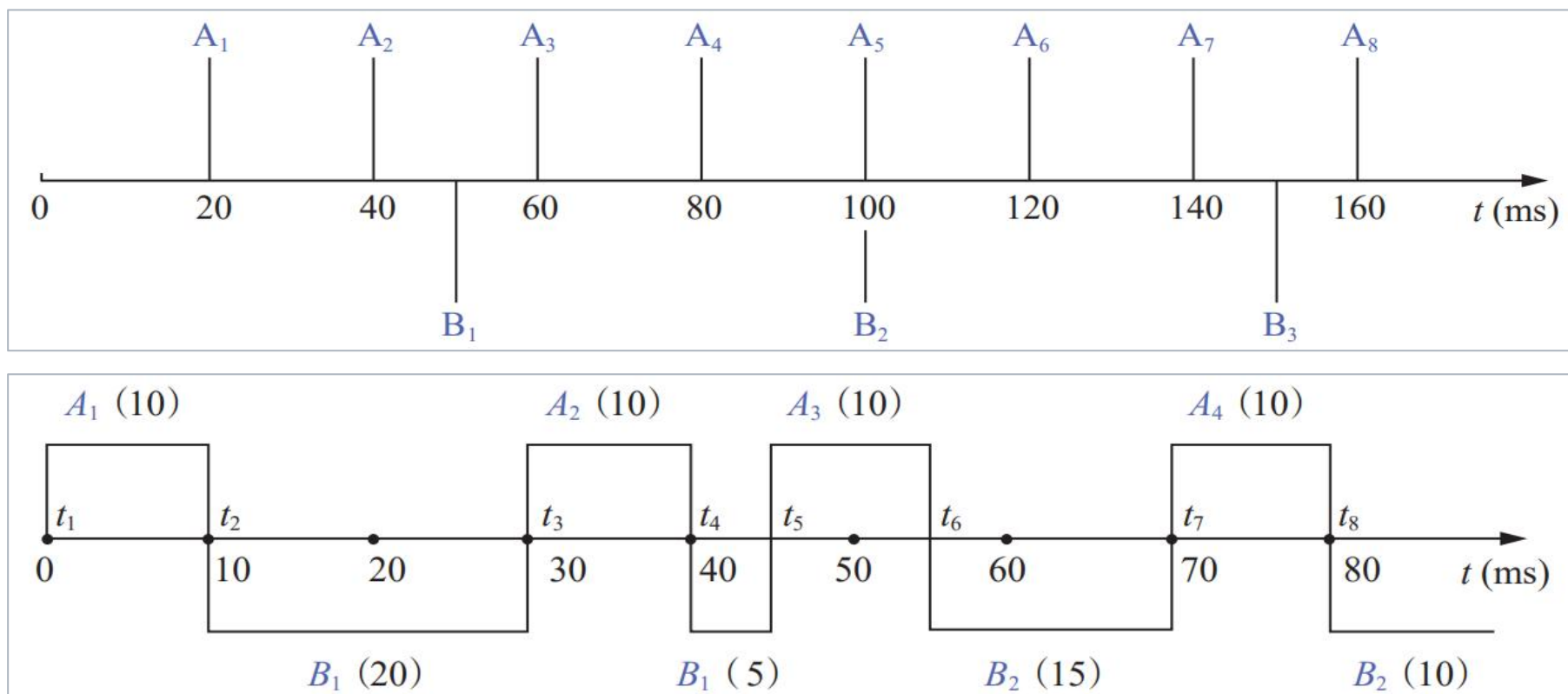
t=80时刻，计算A，B松弛度：

$$A_5 = 100 - 10 - 80 = 10$$

$$B_2 = 100 - 10 - 80 = 10$$

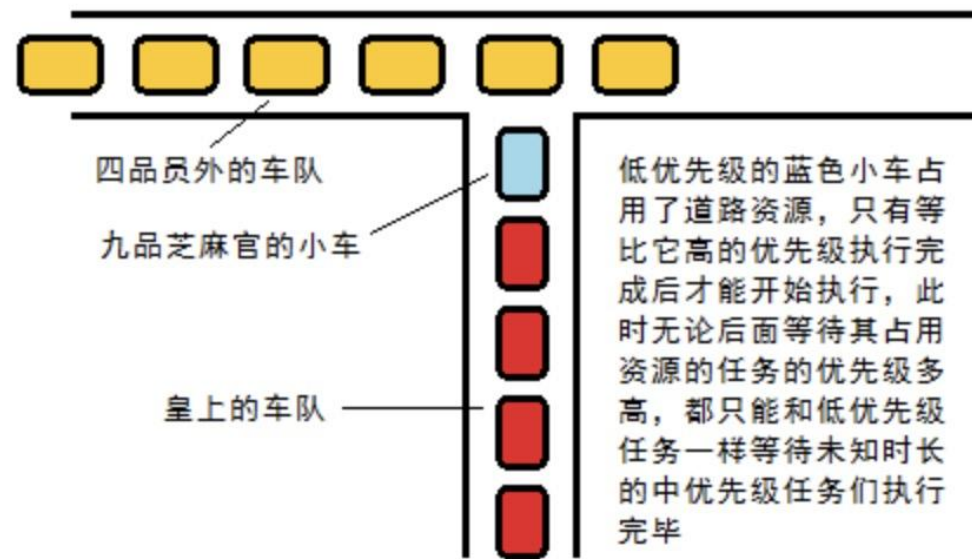
松弛度 = 必须完成的时间点 - 本身剩余运行时间 - 当前时间

- 两个周期性实时任务A和B，任务A要求每20ms执行一次，执行时间为10 ms，任务B要求每50ms执行一次，执行时间为25 ms





采用优先级调度和抢占方式，可能产生**优先级倒置**。现象：高优先级进程被低优先级进程延迟或阻塞。

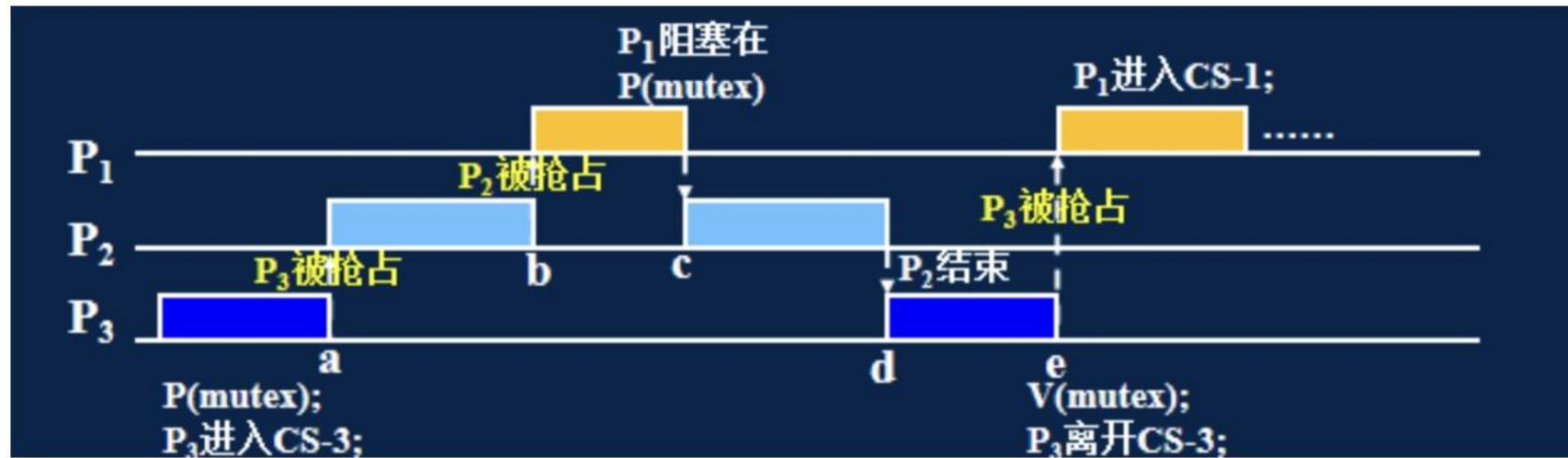


P1: ... P(mutex); CS_1; V(mutex); ...

P2: ... Program2 ...

P3: ... P(mutex); CS_3; V(mutex); ...

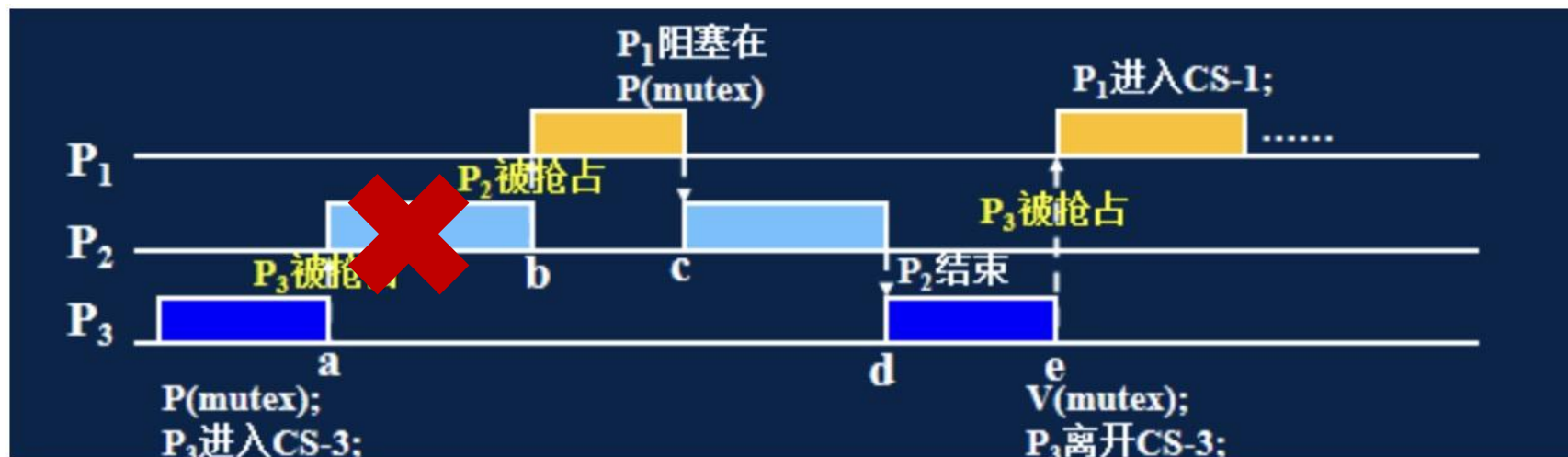
优先级顺序: $P1 > P2 > P1$, P1和P3共享一个临界资源





解决方法:

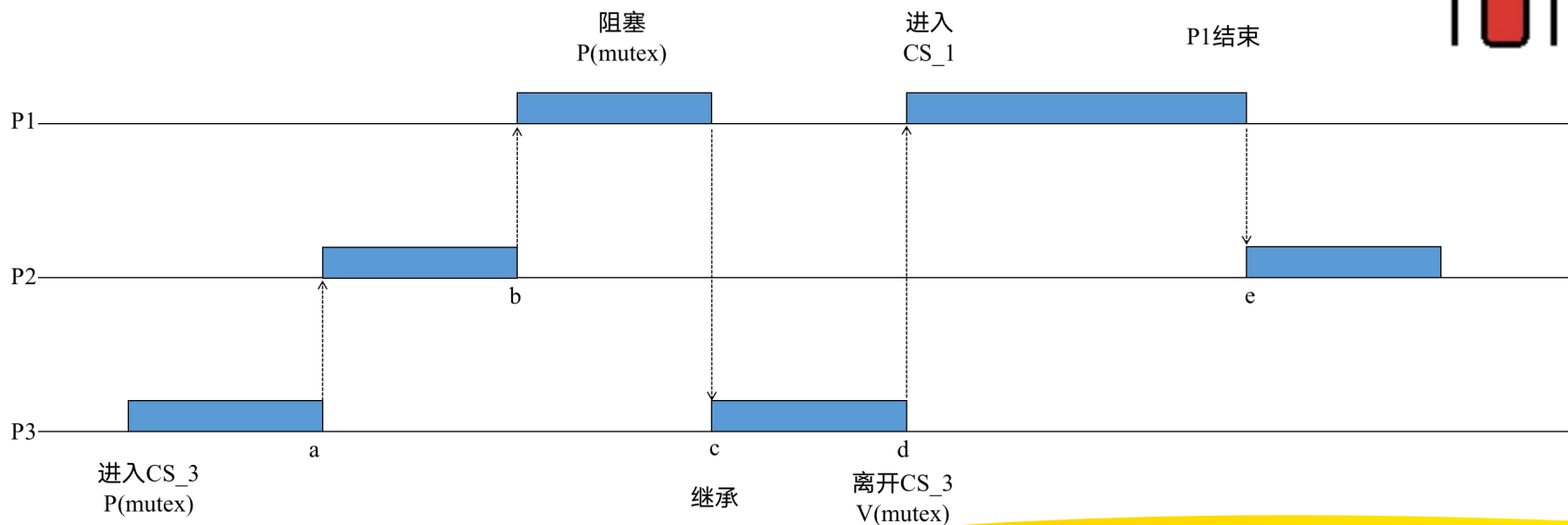
- 制定一些规定，如规定低优先级进程执行后，其所占用的处理机不允许被抢占；





解决方法:

- 建立动态优先级继承。





内容导航:

-  3.1 处理机调度概述
-  3.2 调度算法
-  3.3 实时调度
-  **3.4 Linux进程调度**
-  3.5 死锁概述
-  3.6 预防死锁
-  3.7 避免死锁
-  3.8 死锁的检测与解除

第3章 处理机调度与死锁



- Linux内核2.5版本前，传统的UNIX调度算法
- Linux内核2.5， $O(1)$ 调度算法
- Linux内核2.6.23，完全公平调度算法
- 默认调度算法：完全公平调度（CFS）算法。



默认调度算法：完全公平调度（CFS）算法。

基于调度器类：允许不同的可动态添加的调度算法并存，每个类都有一个特定的优先级。



总调度器：根据调度器类的优先顺序，依次对调度器类中的进程进行调度。



调度器类：使用所选的调度器类算法（调度策略）进行内部的调度。



调度器类的默认优先级顺序为：Stop_Task > Real_Time > Fair > Idle_Task



普通进程调度：

- 采用SCHED_NORMAL调度策略。
- 分配优先级、挑选进程并允许、计算使其运行多久。
- CPU运行时间与友好值（-20~+19）有关，数值越低优先级越高。



实时进程调度：

- 实时调度的进程比普通进程具有更高的优先级。
- SCHED_FIFO：进程若处于可执行的状态，就会一直执行，直到它自己被阻塞或者主动放弃CPU。



普通进程调度：

- 采用SCHED_NORMAL调度策略。
- 分配优先级、挑选进程并允许、计算使其运行多久。
- CPU运行时间与友好值（-20~+19）有关，数值越低优先级越高。



实时进程调度：

- 实时调度的进程比普通进程具有更高的优先级。
- SCHED_FIFO：进程若处于可执行的状态，就会一直执行，直到它自己被阻塞或者主动放弃CPU。
- SCHED_RR：与SCHED_FIFO大致相同，只是进程在耗尽其时间片后，不能再执行，而是需要接受CPU的调度。

进程调度的考量标准

- 等待时间
 - 进程自进入就绪队列开始至进程占用CPU之间的时间间隔
- 周转时间
 - 进程自进入就绪队列开始至进程结束之间的时间间隔
- CPU吞吐量
 - 单位时间内运行结束的进程个数

进程调度的原则

- 公平性原则：
 - 应保证每个进程获得合理的CPU份额
- 有效性原则：
 - CPU资源应得到最大限度的利用
- 友好性原则：响应时间快
 - 与用户（人）交互的时间应尽可能的短
- 快捷性原则：周转时间短
 - 批处理作业的处理时间尽可能的短
- 广泛性原则：吞吐量大
 - 单位时间内完成的作业尽可能的多

时间片轮转调度

- 核心思想：
 - 每个进程运行固定的时间片，然后调入下一个进程
- 实现机理：
 - 维护就绪进程队列，采用FIFO方式一次读取
- 特殊控制：
 - 时间片内发生阻塞或结束，则立即放弃时间片
- 优缺点分析
 - 优点：绝对公平
 - 缺点：公平即合理吗？时间片如何设计才能保证效率？

优先级调度

- 核心思想：
 - 为每个进程赋予不同级别的优先级，越高越优先
- 实现机理：
 - 维护一个优先级队列，自顶向下依次读取
- 特殊控制：
 - 静态优先级与动态优先级概念
- 优缺点分析
 - 优点：响应时间快，易于调整。最通用的方法
 - 缺点：引起饥饿

最短作业优先

- 核心思想：
 - 保证响应时间最快、平均周转时间最短
- 实现机理：
 - 依据先验信息，将进程按照运行时间增序调度
- 特殊控制：
 - 如何确定最短作业？
- 优缺点分析
 - 优点：保证了CPU的利用效率
 - 缺点：无法通用，约束条件多



■ 实时调度

- 针对于专用领域和专用应用目的
- 必须具备前提条件才能进行实时调度
- 特点：系统规模小、中断时间短、进程切换快

- FCFS（先来先服务）：只考虑等待时间；
- SJF（短作业优先算法）：只考虑执行时间；
- HRRN（高响应比优先算法）：介于FCFS和SJF之间

$$\text{优先权} = \frac{\text{等待时间} + \text{要求服务时间}}{\text{要求服务时间}} = \frac{\text{响应时间}}{\text{要求服务时间}}$$

- RR（时间片轮转算法）：机会均等，都能享有；
- FB（多级反馈队列调度算法）
- EDF（最早截止时间优先算法）
- LLF（最低松弛度优先算法）



简答题

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18						

计算题
综合应用题

19	20	21	22
----	----	----	----

23	24	25
----	----	----

标黄色为本次作业