第三章　处理机调度与死锁

BACK

单选题 1分

设置

分配给进程的时间片还没用完，一个优先级更高的进程就绪，此时系统并不立即剥夺当前进程的处理机，而是等当前进程的时间片用完之后才调度优先级更高的进程——这种情况属于抢占式调度还是非抢占式调度？

A 抢占式调度

B 非抢占式调度

2

提交

思考

分配给进程的时间片还没用完，一个优先级更高的进程就绪，此时系统并不立即剥夺当前进程的处理机，而是等当前进程的时间片用完之后才调度优先级更高的进程——这种情况属于抢占式调度还是非抢占式调度？

抢占式调度

——时间片属于抢占式调度

3.4 实 时 调 度

在实时系统中，可能存在着两类不同性质的实时任务，即HRT任务和SRT任务，它们都联系着一个截止时间。

为保证系统能正常工作，实时调度必须能满足实时任务对截止时间的要求。

3.4.1　实现实时调度的基本条件

1. 提供必要的信息

为了实现实时调度，系统应向调度程序提供有关任务的信息：

(1) 就绪时间，是指某任务成为就绪状态的起始时间，在周期任务的情况下，它是事先预知的一串时间序列。

(2) 开始截止时间和完成截止时间，对于典型的实时应用，只须知道开始截止时间，或者完成截止时间。

(3) 处理时间，一个任务从开始执行，直至完成时所需的时间。

(4) 资源要求，任务执行时所需的一组资源。

(5) 优先级，如果某任务的开始截止时间错过，会引起故障，则应为该任务赋予“绝对”优先级；如果其开始截止时间的错过，对任务的继续运行无重大影响，则可为其赋予“相对”优先级，供调度程序参考。

2. 系统处理能力强

在实时系统中，若处理机的处理能力不够强，则有可能因处理机忙不过，而致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HRT，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系统才是可调度的：

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

2. 系统处理能力强

在实时系统中，若处理机的处理能力不够强，则有可能

处理时间：任务使用的CPU时间

周期时间：两次任务之间的时间间隔

硬实时任务HRT，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系统才是可调度的：

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

一个实时系统使用了4个周期事件，其周期分别为50ms，100ms，200ms，250ms。假设这4个周期事件分别需要35ms，20ms，10ms和xms的CPU时间，保持系统可调度的最大x值是多

解法1：

$35/50+20/200+10/200+x/250\leq 1$

解法2：

1s中，4个事件的分别需要的CPU时间为：

$1000/50\times 35 = 700ms$

$1000/100\times 20 = 200ms$

$1000/200\times 10 = 50ms$

$1000/250\times x = 4xms$

$700+200+50+4x\leq 1000$

$x \leq 12.5ms$

提高系统处理能力的途径有二：一是采用单处理机系统，但须增强其处理能力，以显著地减少对每一个任务的处理时间；二是采用多处理机系统。假定系统中的处理机数为N，则应将上述的限制条件改为：

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

3. 采用抢占式调度机制

在含有HRT任务的实时系统中，广泛采用抢占机制。这样便可满足HRT任务对截止时间的要求。

抢占调度机制比较复杂。对于一些小的实时系统，若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，以简化调度程序和在任务调度时所花费的系统开销。

设计抢占调度机制时，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，**及时将自己阻塞起来**，以便释放出处理机，供调度程序去调度那个开始时间即将到达的任务。

4. 具有快速切换机制

为保证硬实时任务能及时运行，在系统中还应具有快速切换机制，使之能进行任务的快速切换。该机制应具有如下两方面的能力：

(1) 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)。

(2) 快速的任务分派能力。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。

3.4.2　实时调度算法的分类

可以按不同方式对实时调度算法加以分类：

① 根据实时任务性质，可将实时调度的算法分为硬实时调度算法和软实时调度算法；

② 按调度方式，则可分为非抢占调度算法和抢占调度算法。

1. 非抢占式调度算法

(1) 非抢占式轮转调度算法。

(2) 非抢占式优先调度算法。

非抢占式轮转调度算法：

一台计算机控制若干个相同（或类似）的对象，为每个被控对象建立一个实时任务，并排成一个轮转队列。

调度程序每次调度队首的任务运行，任务完成后，将它挂在轮转队列末尾等待，调度程序再选择下一个队首任务运行。

该算法可获得数秒至数十秒的响应时间，用于要求不太严格的实时系统中。

1. 非抢占式调度算法

(1) 非抢占式轮转调度算法。

(2) 非抢占式优先调度算法。

非抢占式优先调度算法：

系统中还含有少量具有一定要求的实时任务，需要采用这种方式。

这些任务到达时，把它们安排在就绪队列的队首，等待当前任务自我终止或运行完成后，便调度队首的高优先进程。

该算法经过精心处理后，响应时间可以减少到数秒至数百毫秒，可用于有一定要求的实时系统中。

2. 抢占式调度算法

可根据抢占发生时间的不同而进一步分成以下两种调度算法：

(1) 基于时钟中断的抢占式优先级调度算法。

(2) 立即抢占(Immediate Preemption)的优先级调度算法。

基于时钟中断的抢占式调度算法：

高优先级任务到达后，并不立即抢占当前任务的处理机，而是等时钟中断发生时，调度程序才剥夺当前任务的执行，将处理机分配给新到的高优先级任务。

该算法可获得较好的相应效果，调度延迟降为几十至几百毫秒，可用于大多数的实时系统中。

2. 抢占式调度算法

可根据抢占发生时间的不同而进一步分成以下两种调度算法：

立即抢占的抢占式调度算法：

该调度策略要求操作系统具有快速响应外部事件中断的能力。

一旦出现外部中断，只要当前任务未处于临界区，便能立即剥夺当前任务的执行，把处理机分配给请求中断的紧迫任务。

该算法可获得非常快的响应效果，调度延迟降为几毫秒至100毫秒，甚至更低。

3.4.3　最早截止时间优先EDF(Earliest Deadline First)算法

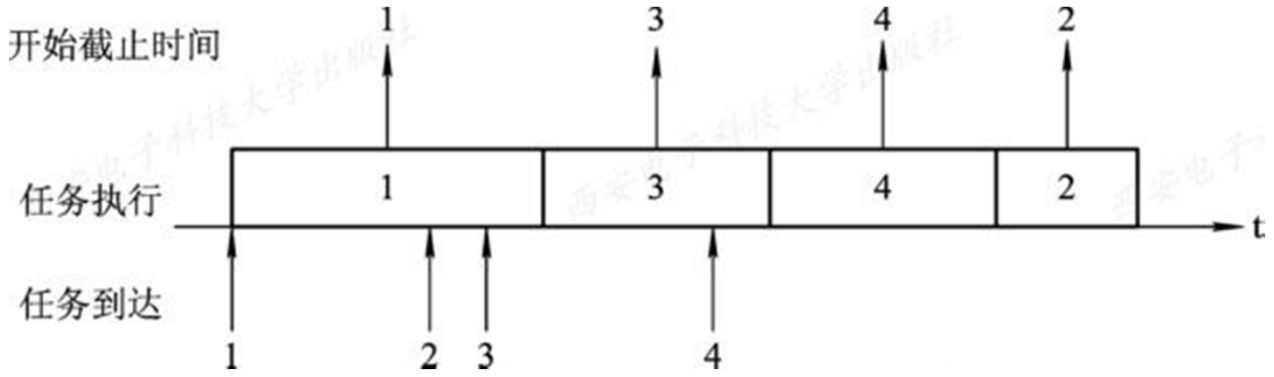
该算法根据任务的截止时间确定任务的优先级，任务截止时间越早，优先级越高，具有最早截止时间的任务排在队列的队首。

该算法既可用于抢占式调度方式，也可用于非抢占式调度方式。

1. 非抢占式调度方式用于非周期实时任务

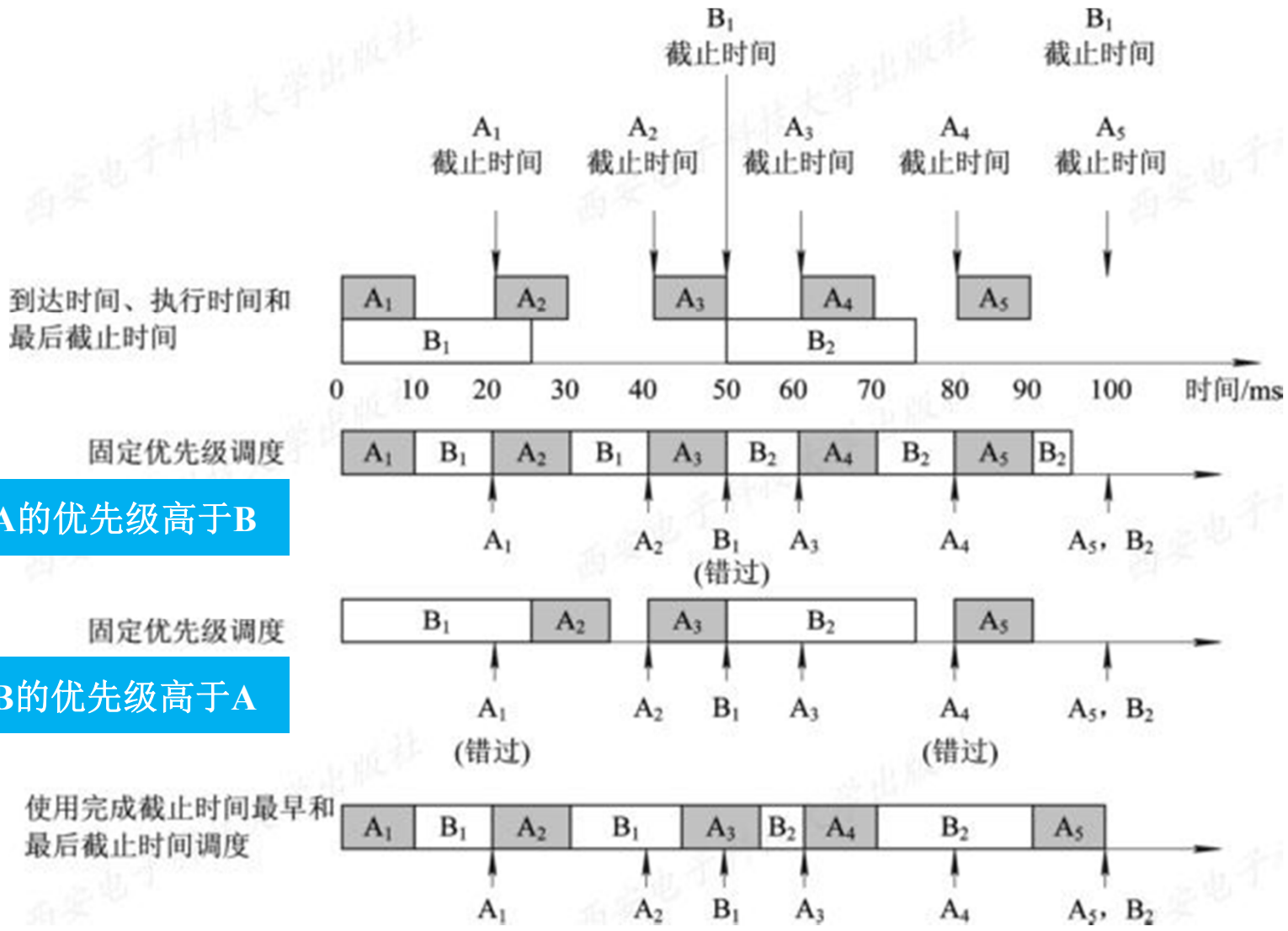
举例如下：

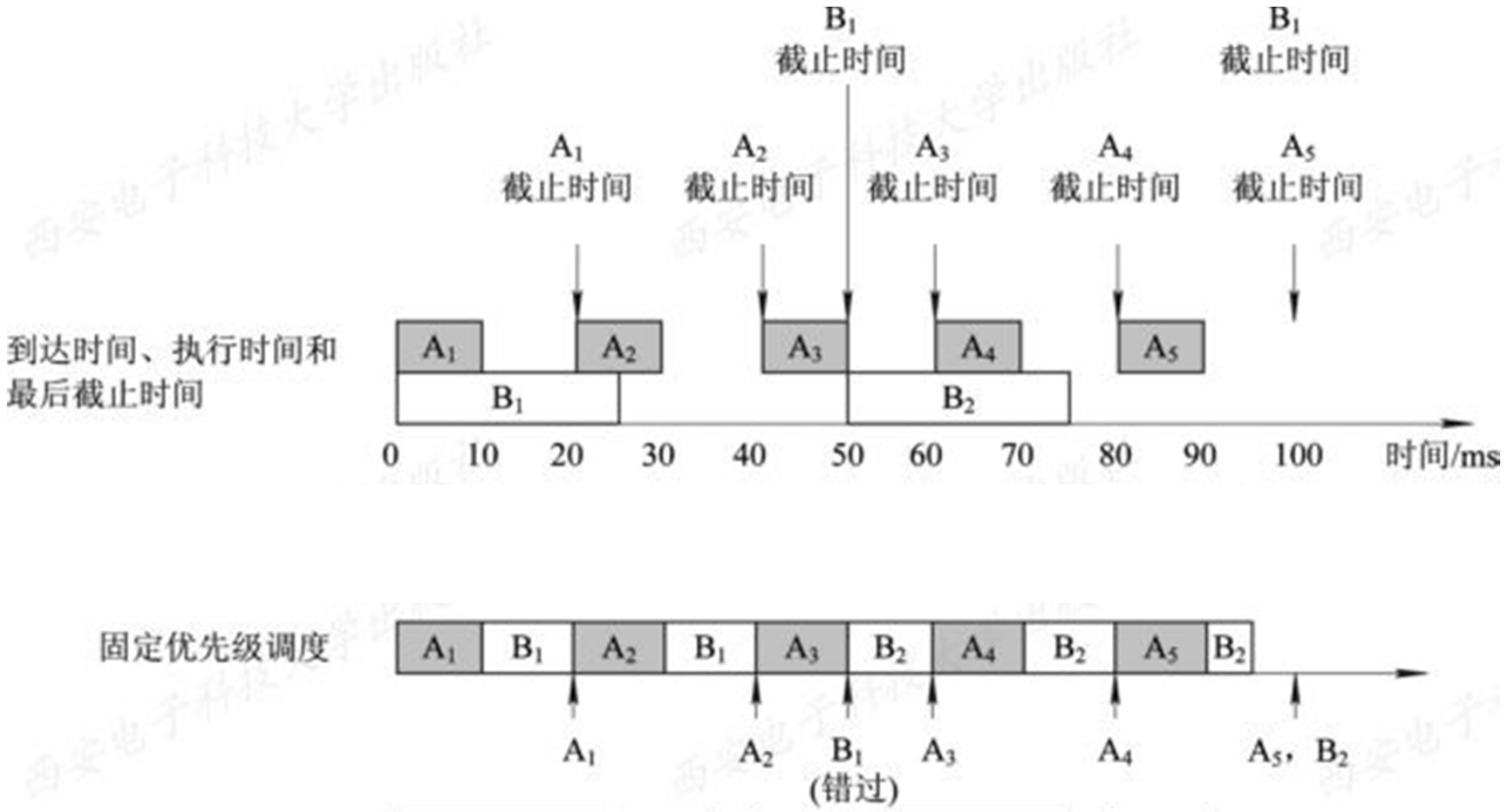
四个非周期任务，任务1先执行，执行期间，任务2、3先后到达。任务3的开始截止时间早于任务2，所以任务1执行后先调度任务3执行。任务3执行期间任务4到达，它的开始截止时间依然早于任务2，因此任务3执行完，系统先调度任务4，最后才调度任务2。

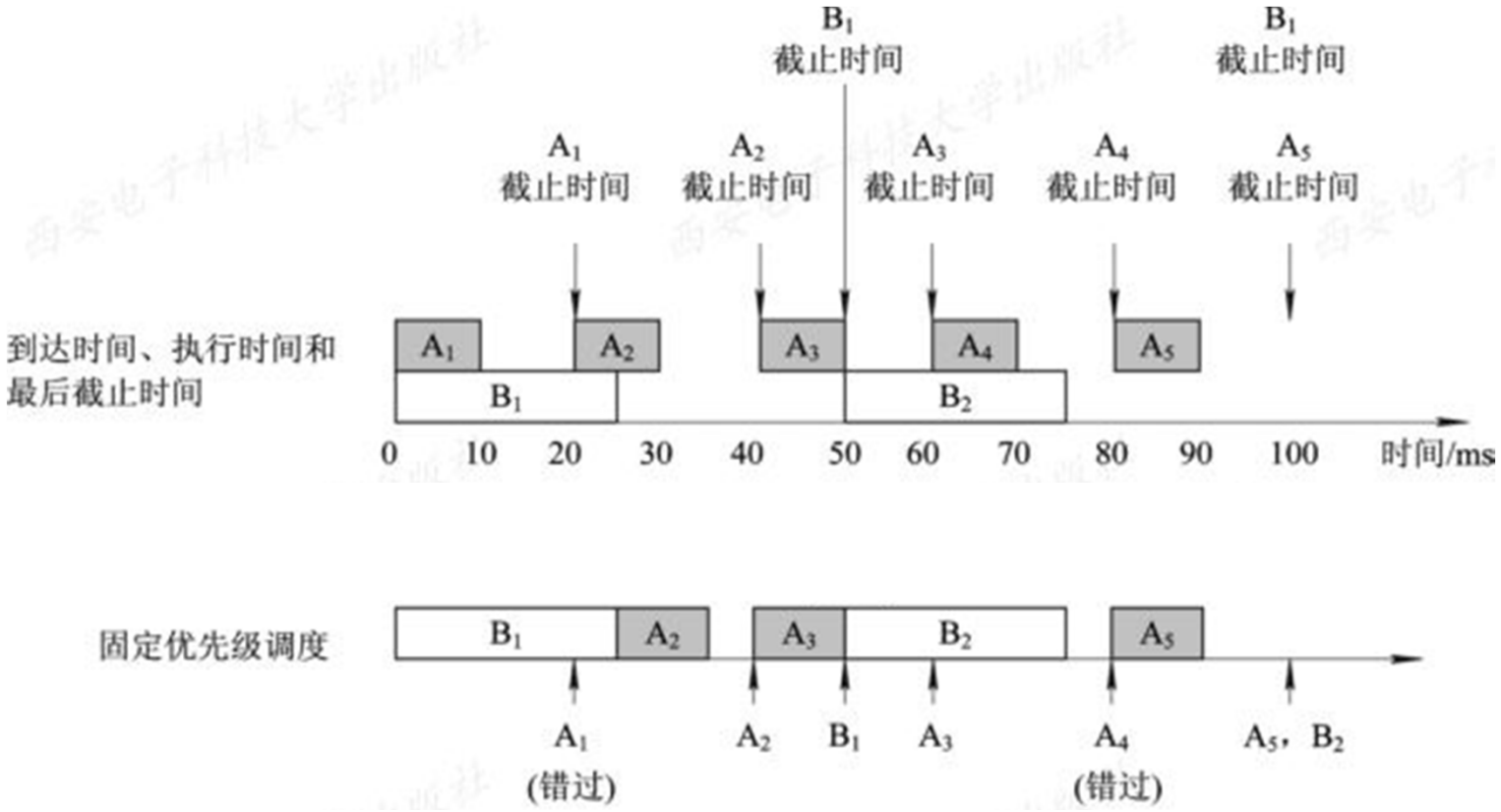


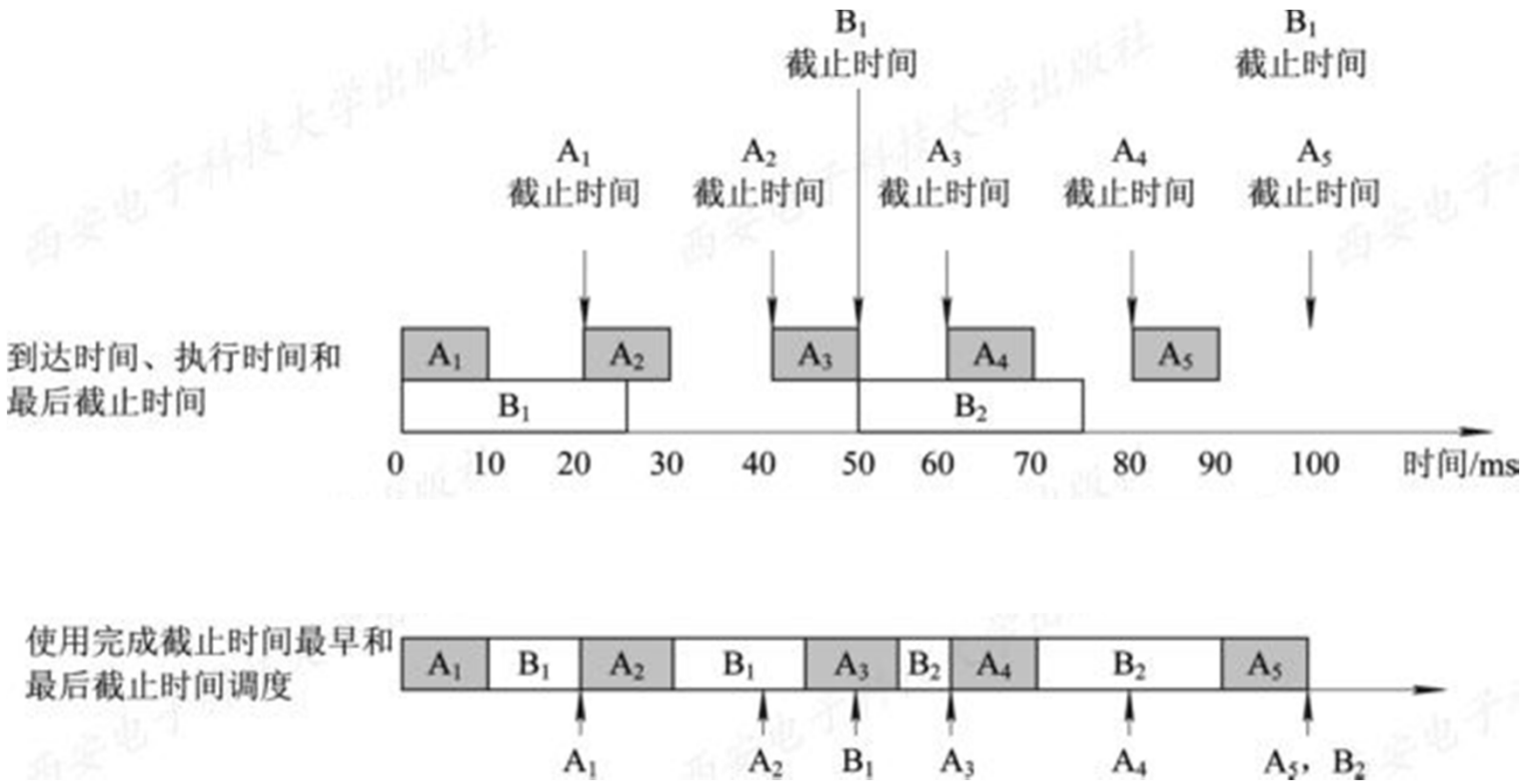
2. 抢占式调度方式用于周期实时任务

图3-7示出了将该算法用于抢占调度方式之例。在该例中有两个周期任务，任务A和任务B的周期时间分别为20 ms和50 ms，每个周期的处理时间分别为10 ms和25 ms。


B1 截止时间
B1 截止时间
A1 截止时间
A2 截止时间
A3 截止时间
A4 截止时间
A5 截止时间
到达时间、执行时间和最后截止时间
A1
A2
A3
A4
A5
B1
B2
0 10 20 30 40 50 60 70 80 90 100 时间/ms
固定优先级调度
A的优先级高于B
A1 B1 A2 B1 A3 B2 A4 B2 A5 B2
A1 A2 B1 (错过) A3 A4 A5, B2
固定优先级调度
B的优先级高于A
B1 A2 A3 B2 A5
A1 (错过) A2 B1 A3 A4 (错过) A5, B2
使用完成截止时间最早和最后截止时间调度
A1 B1 A2 B1 A3 B2 A4 B2 A5
A1 A2 B1 A3 A4 A5, B2


B1
截止时间
B1
截止时间
A1
截止时间
A2
截止时间
A3
截止时间
A4
截止时间
A5
截止时间
到达时间、执行时间和
最后截止时间
A1
A2
A3
A4
A5
B1
B2
0
10
20
30
40
50
60
70
80
90
100
时间/ms
固定优先级调度
A1
B1
A2
B1
A3
B2
A4
B2
A5
B2
A1
A2
B1
(错过)
A3
A4
A5, B2


B1
截止时间
B1
截止时间
A1
截止时间
A2
截止时间
A3
截止时间
A4
截止时间
A5
截止时间
到达时间、执行时间和
最后截止时间
A1
A2
A3
A4
A5
B1
B2
0
10
20
30
40
50
60
70
80
90
100
时间/ms
固定优先级调度
B1
A2
A3
B2
A5
A1
(错过)
A2
B1
A3
A4
(错过)
A5，B2


B1
截止时间
B1
截止时间
A1
截止时间
A2
截止时间
A3
截止时间
A4
截止时间
A5
截止时间
到达时间、执行时间和
最后截止时间
A1
A2
A3
A4
A5
B1
B2
0
10
20
30
40
50
60
70
80
90
100
时间/ms
使用完成截止时间最早和
最后截止时间调度
A1
B1
A2
B1
A3
B2
A4
B2
A5
A1
A2
B1
A3
A4
A5, B2

3.4.4　最低松弛度优先LLF(Least Laxity First)算法

该算法在确定任务的优先级时，根据的是任务的紧急(或松弛)程度。任务紧急程度愈高，赋予该任务的优先级就愈高，以使之优先执行。

例如：一个任务在200ms时必须完成，但它本身需要的运行时间是100ms，因此调度程序必须在100ms之前调度执行，该任务的松弛程度（紧急程度）为100ms；另一个任务在400ms时必须完成，它本身需要运行150ms，则其松弛程度为250ms。

就绪队列按照松弛度排队，松弛度最低的任务排在最前面，因此第一个任务的优先级更高

3.4.4　最低松弛度优先LLF(Least Laxity First)算法

该算法在确定任务的优先级时，根据的是任务的紧急

该算法主要用于可抢占调度方式中

例如：一个任务在200ms时必须完成，但它本身需要的运行时间是100ms，因此调度程序必须在100ms之前调度执行，该任务的松弛程度（紧急程度）为100ms；另一个任务在400ms时必须完成，它本身需要运行150ms，则其松弛程度为250ms。

就绪队列按照松弛度排队，松弛度最低的任务排在最前面，因此第一个任务的优先级更高

假如在一个实时系统中有两个周期性实时任务A和B，任务A要求每20 ms执行一次，执行时间为10 ms，任务B要求每50 ms执行一次，执行时间为25 ms。由此可知，任务A和B每次必须完成的时间分别为：A_1、A_2、A_3、…和B_1、B_2、B_3、…，。

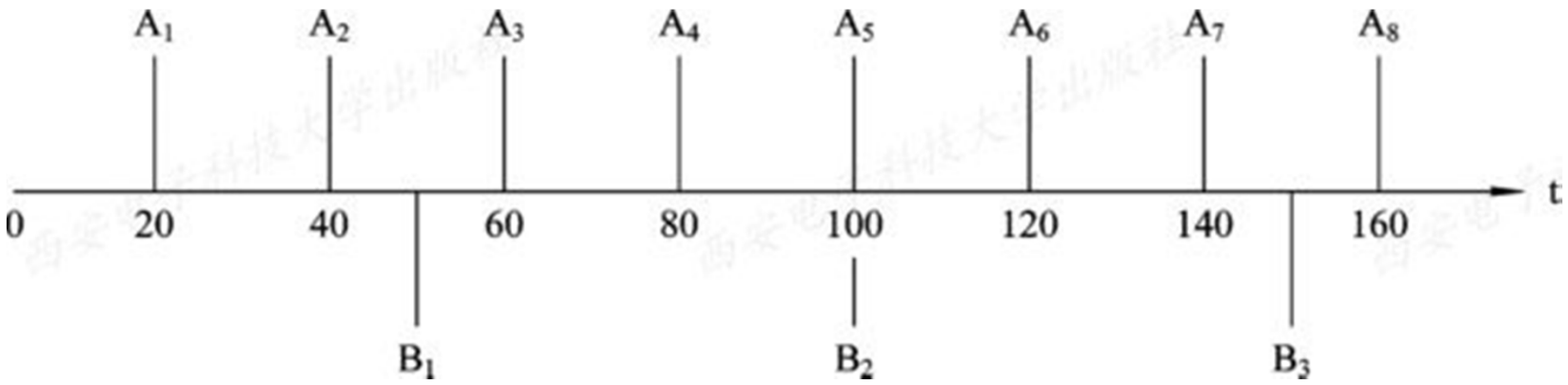
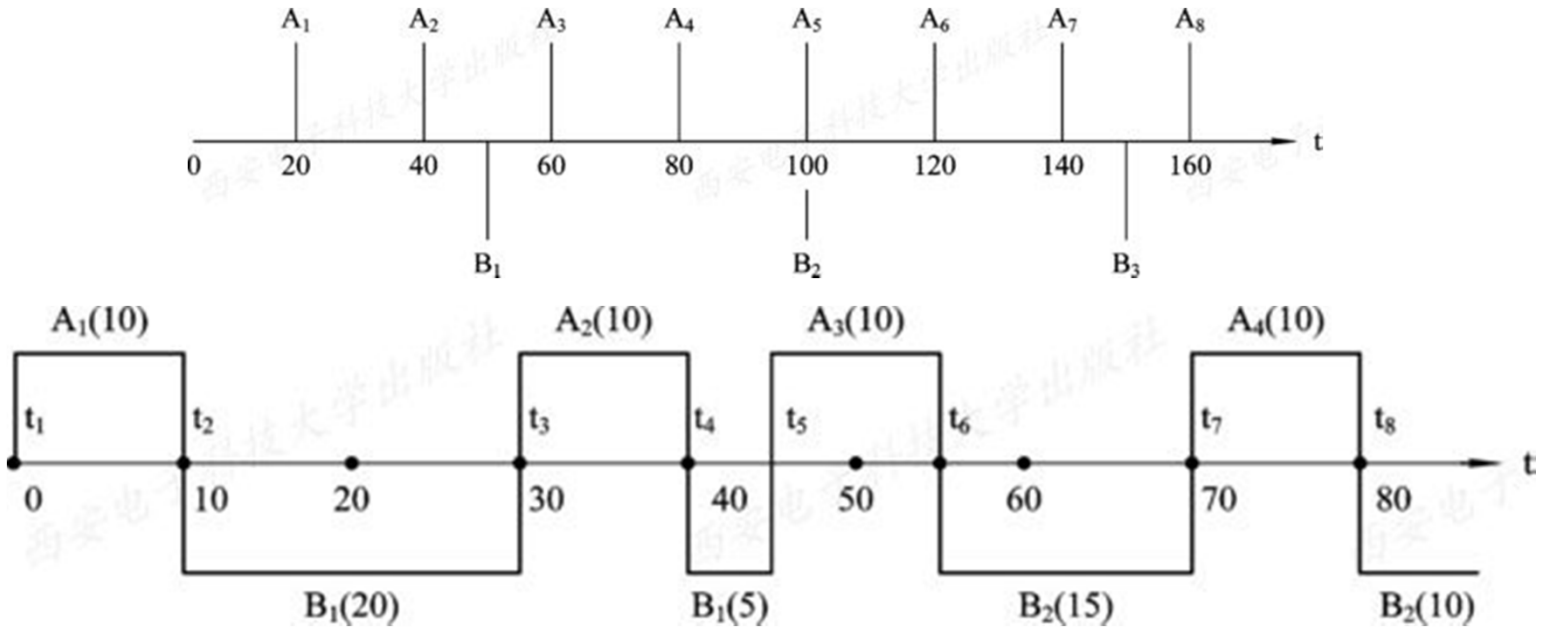


图3-8 A和B任务每次必须完成的时间

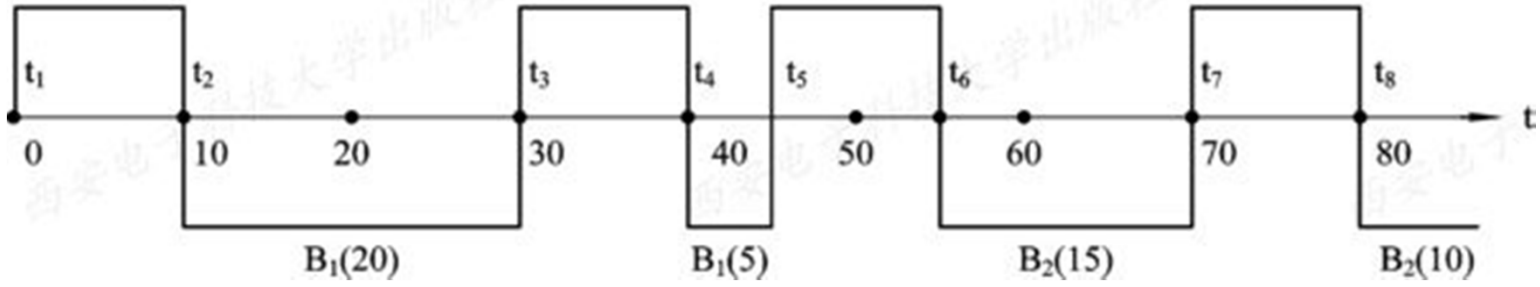
刚开始(t1=0)时，A1必须在20ms时完成，本身运行时间10ms，可算出A1的松弛度为10ms，同理可算出B1的松弛度为25ms，因此A1先执行。

t2=10ms时，A1运行结束，A2还未到达，B1的松弛度为15ms(50-25-10)，因此调度B1运行。



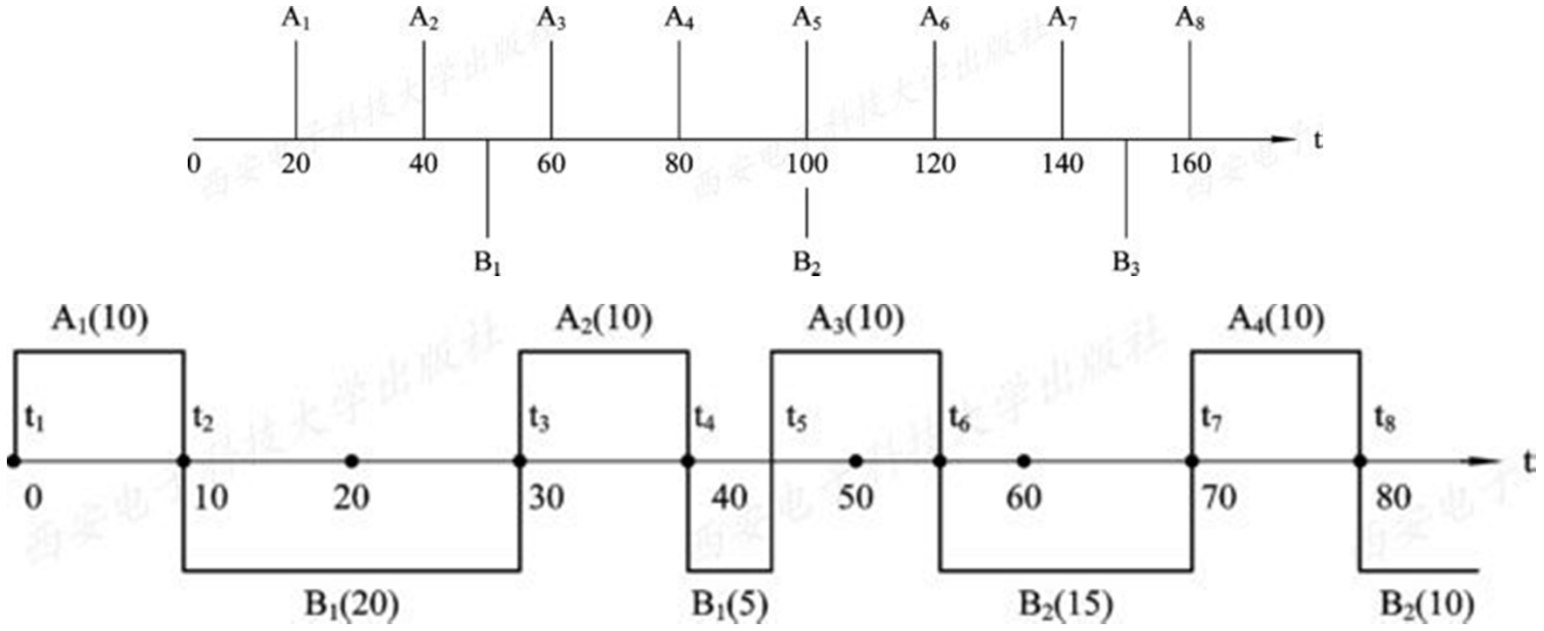
第20ms时，A2到达，此时A2的松弛度为10ms，B1的松弛度为15ms，虽然B1松弛度高，但是A2松弛度未到0，不需要切换。

A2的松弛度	**=必须完成时间-剩余的运行时间-当前时间 =40ms-10ms-20ms =10ms**
B2的松弛度	**=必须完成时间-剩余的运行时间-当前时间 =50ms-15ms-20ms =15ms**



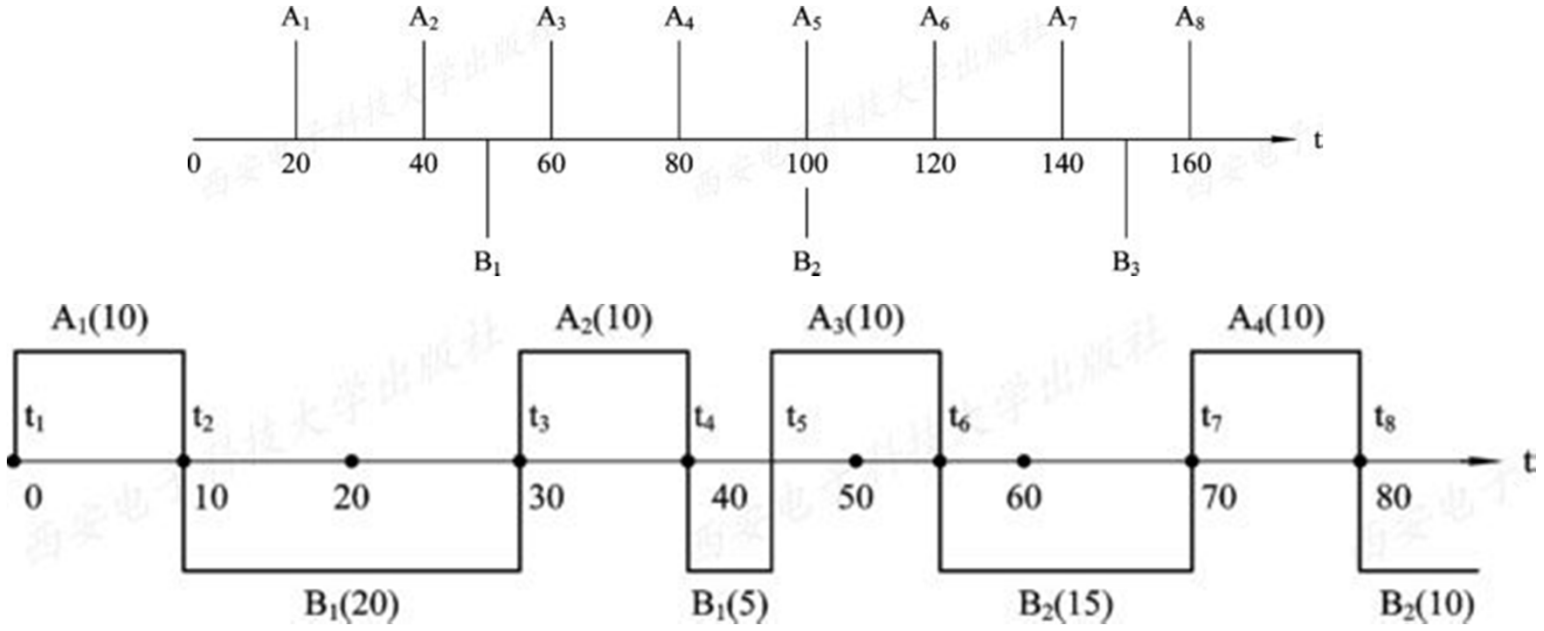
第20ms时，A2到达，此时A2的松弛度为10ms，B1的松弛度为15ms，虽然B1松弛度高，但是A2松弛度未到0，不需要切换。

t3=30ms时，A2的松弛度减为0(40-10-30)，B1的松弛度为15ms(50-5-30)，因此调度程序应抢占B1的处理机而调度A2运行。



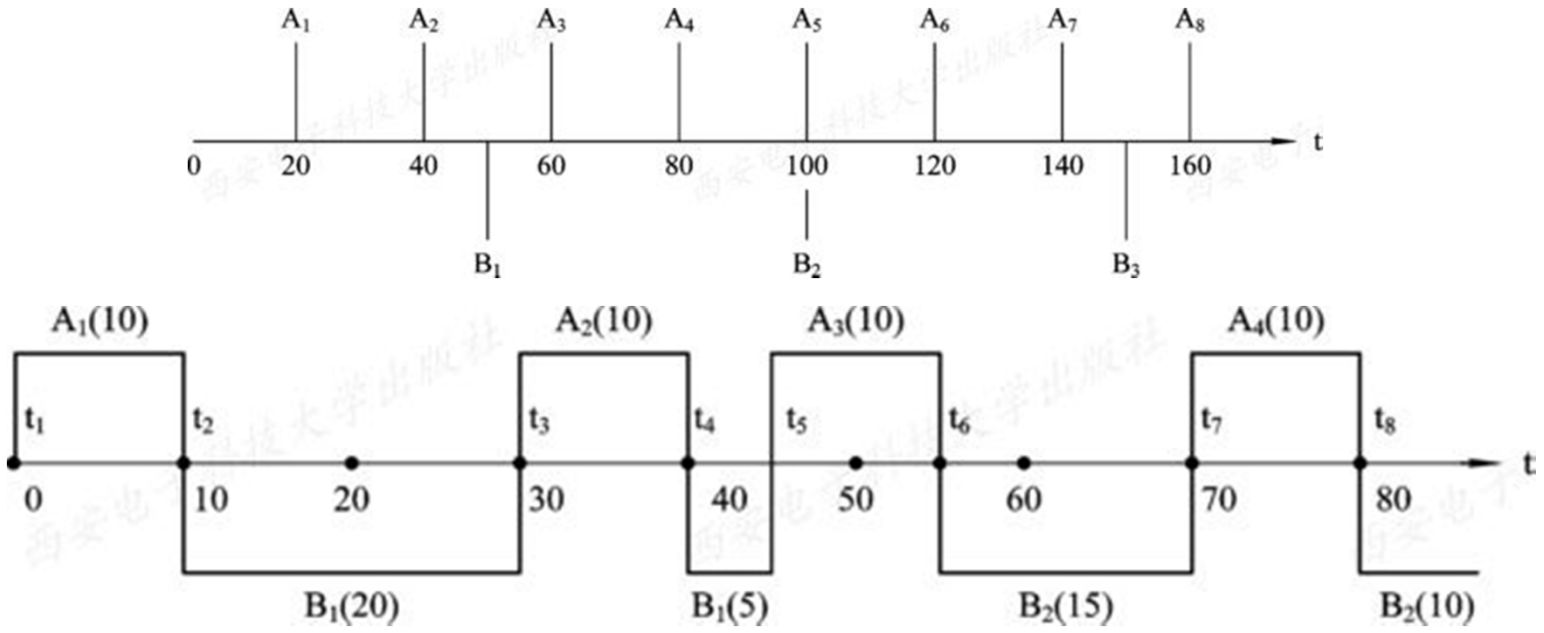
t4=40ms时，A3到达，A3的松弛度为10ms(60-10-40)，而B1的松弛度为5ms(50-5-40)，因此又重新调度B1执行。

t5=45ms时，B1执行完成，此时A3的松弛度减为5ms(60-10-45)，B2还未到达，因此调度A3执行。



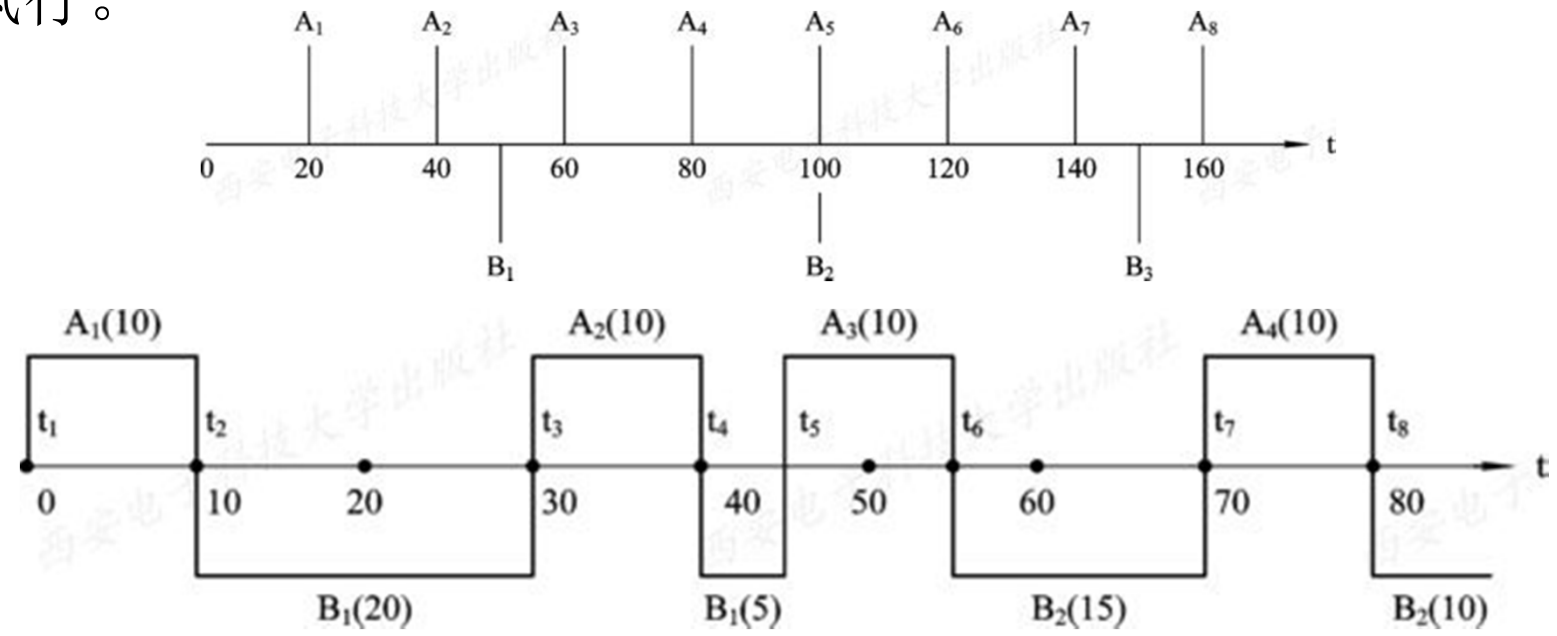
第50ms时，B2到达，此时B2的松弛度为25ms，未到0，不需要切换，保持A3继续运行。

t6=55ms时，A3执行完成，此时任务A尚未进入第4个周期，而任务B已经开始第2个周期，因此再调度B2执行。



第60ms时，A4到达，此时A4的松弛度为10ms，未到0，不需要切换，保持B2继续运行。

t7=70ms时，A4松弛度为0(80-10-70)，B2松弛度为20ms(100-10-70)，因此调度程序抢占B2的处理机而调度A4执行。



3.4.5　优先级倒置(priority inversion problem)

1. 优先级倒置的形成

当前OS广泛采用优先级调度算法和抢占方式，然而在系统中存在着影响进程运行的资源而可能产生“优先级倒置”的现象，即高优先级进程（或线程）被低优先级进程（或线程）延迟或阻塞。

假如P_3最先执行，在执行了P(mutex)操作后，进入到临界区CS-3。

1）在时刻a，P_2就绪，因为它比P_3的优先级高，P_2抢占了P_3的处理机而运行。

2）在时刻b，P1就绪，它比P2的的级别高，因此P1抢占P2的处理机而运行。

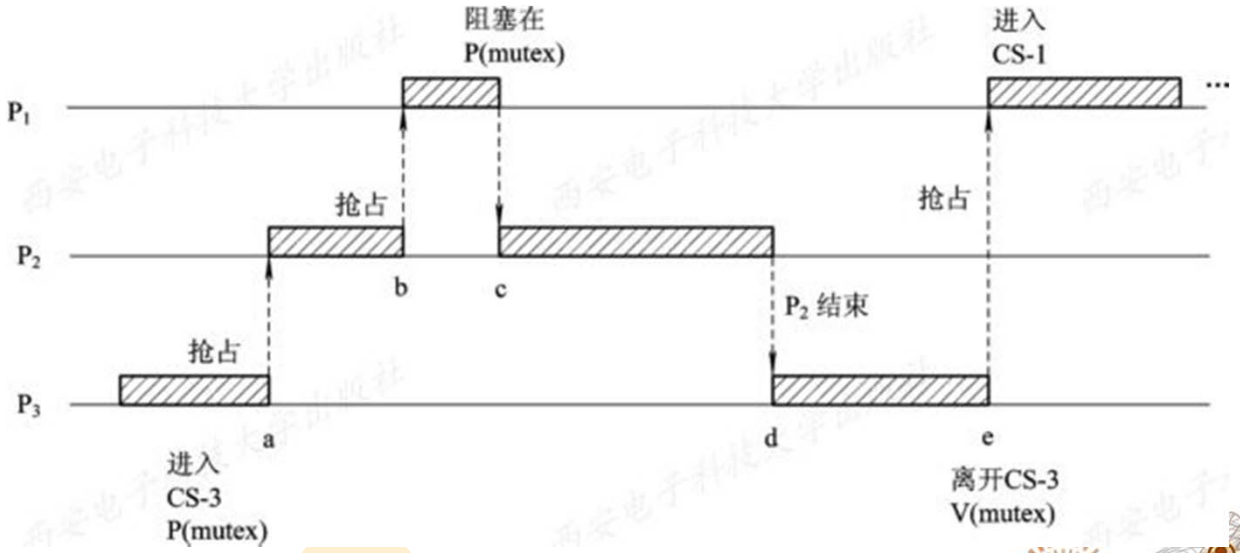
3）在时刻c，P1执行P(mutex)操作，试图进入临界区CS-1，因为临界资源已被P3占用，故P1被阻塞。此时由P2继续运行。

4）在时刻d，P2运行结束，P3接着运行。

5）在时刻e，P3运行V(mutex)退出临界区，唤醒P1；P1优先级高于P3抢占P3的处理机运行。

优先级：$P_1 > P_2 > P_3$

P_1和P_3通过共享的临界资源进行交互



2. 优先级倒置的解决方法

一种简单的解决方法是规定：假如进程P3在进入临界区后P3所占用的处理机就不允许被抢占。

此时，P2优先级即便高于P3也不能执行。 P3可能会较快地退出临界区。

如果系统中的临界区都较短且不多，该方法可行。

如果临界区非常长，则高优先级进程P1依然会等待很长时间，效果无法令人满意。

比较实用的方法是建立在动态优先级继承的基础上：当高优先级进程P1要进入临界区时，如果已有一个低优先级进程P3正在使用该临界资源，此时一方面P1阻塞，另一方面P3继承P1的优先级，并一直保持到P3退出临界区。

这样可以避免让优先级比P3高，但比P1低的进程如P2插进来，导致延缓P3退出临界区。