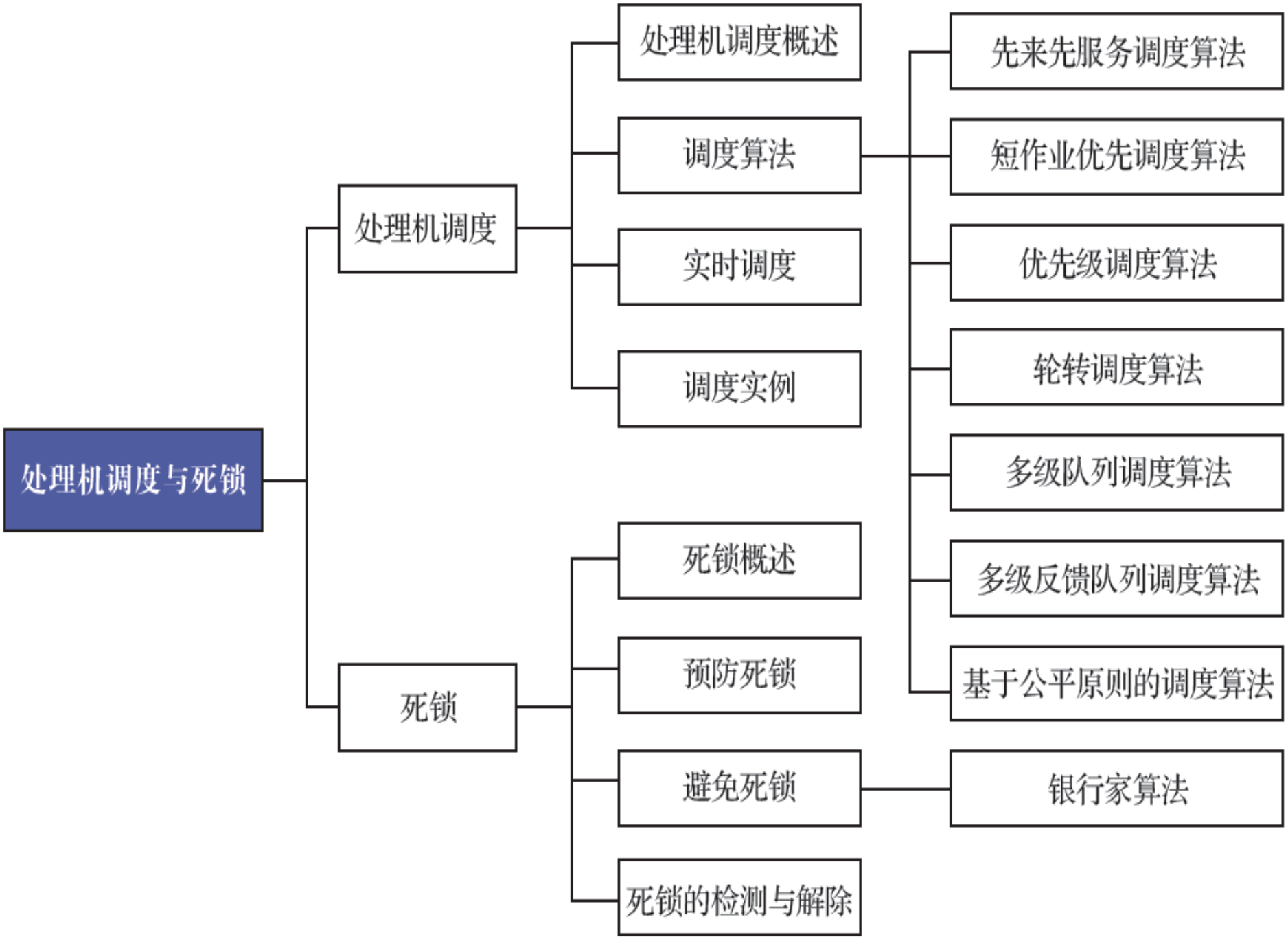
海南大学
HAINAN UNIVERSITY

《操作系统原理及安全》

教师：秦小立
学院：网络空间安全学院
邮箱：xlqin@hainanu.edu.cn
办公地点：学院309

课程知识导图

第1章	操作系统引论
第2章	**进程的描述与控制**
第3章	**处理机调度与死锁**
第4章	**进程同步**
第5章	**存储器管理**
第6章	**虚拟存储器**
第7章	**输入/输出系统**
第8章	文件管理
第9章	**磁盘存储器管理**
第10章	多处理机操作系统
第11章	虚拟化和云计算
第12章	保护和安全


处理机调度与死锁
处理机调度
处理机调度概述
调度算法
先来先服务调度算法
短作业优先调度算法
优先级调度算法
轮转调度算法
多级队列调度算法
多级反馈队列调度算法
基于公平原则的调度算法
实时调度
调度实例
死锁
死锁概述
预防死锁
避免死锁
银行家算法
死锁的检测与解除

第3章 处理机调度与死锁

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

3.1.1 处理机调度的层次

3.1.2 作业和作业调度

3.1.3 进程调度

3.1.4 处理机调度算法的目标

3.1.1 处理机调度的层次

◆处理机调度的类型/层次：

- 高级调度
- 低级调度
- 中级调度

3.1.1 处理机调度的层次

◆1、高级调度

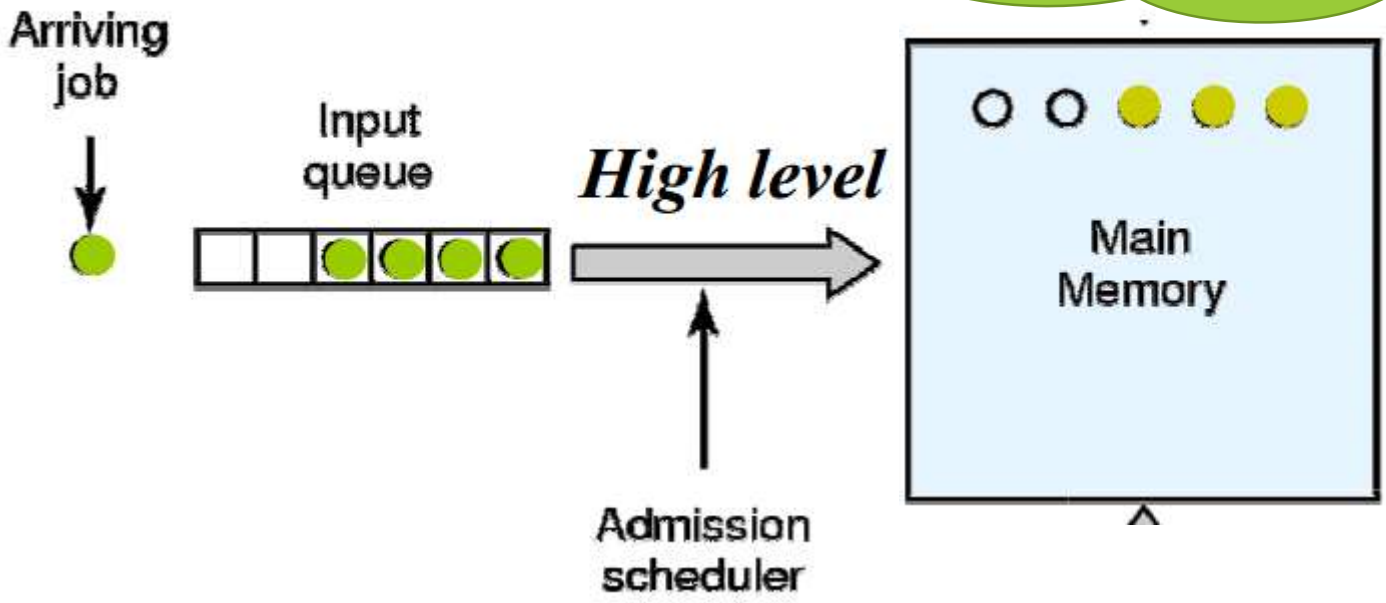
- 高级调度，又叫做长程调度、作业调度、宏观调度

  根据某种算法，决定将外存上处于后备队列中的作业调入内存，并为它们创建进程和分配必要的资源。然后，将新创建的进程排在就绪队列上等待调度。
- 调度对象是作业，主要用于多道批处理系统中。

3.1.1 处理机调度的层次

◆1、高级调度

分/实时系统中一般
不设置高级调度



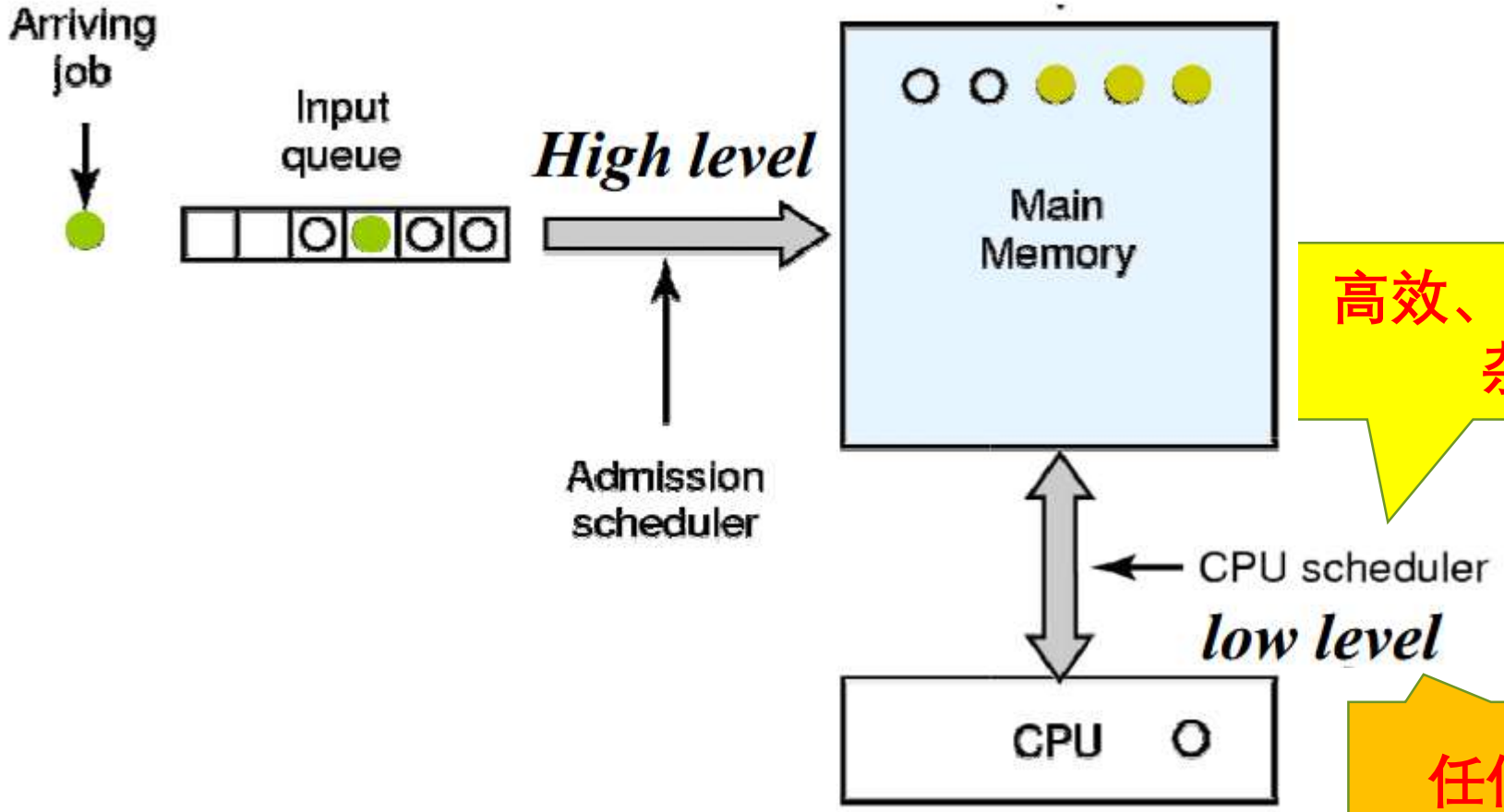
3.1.1 处理机调度的层次

◆2、低级调度

- 低级调度，又叫做短程调度、进程调度、微观调度。
  根据某种算法，决定就绪队列中哪个进程应获得处理机，再由分派程序执行 ，再由分派程序执行把处理机分配给被选中的进程。
- 调度对象是进程（或LWP），应用在于多道批处理、分时和实时OS。
- 包括进程调度或线程调度两种。

3.1.1 处理机调度的层次

◆2、低级调度



3.1.1 处理机调度的层次

◆2、低级调度

➢ 低级调度的主要功能

- 保存处理机进程的现场信息。如程序计数器、通用寄存器中的信息等，并将信息直接保存在PCB。
- 按照规则从就绪队列中选取进程进占CPU 。如优先级法、轮转法等，并将就绪态切换为运行态。
- 若因资源不满足，则调用阻塞原语，将退出进程加入阻塞队列。并把CPU的控制权交给选中进程。

3.1.1 处理机调度的层次

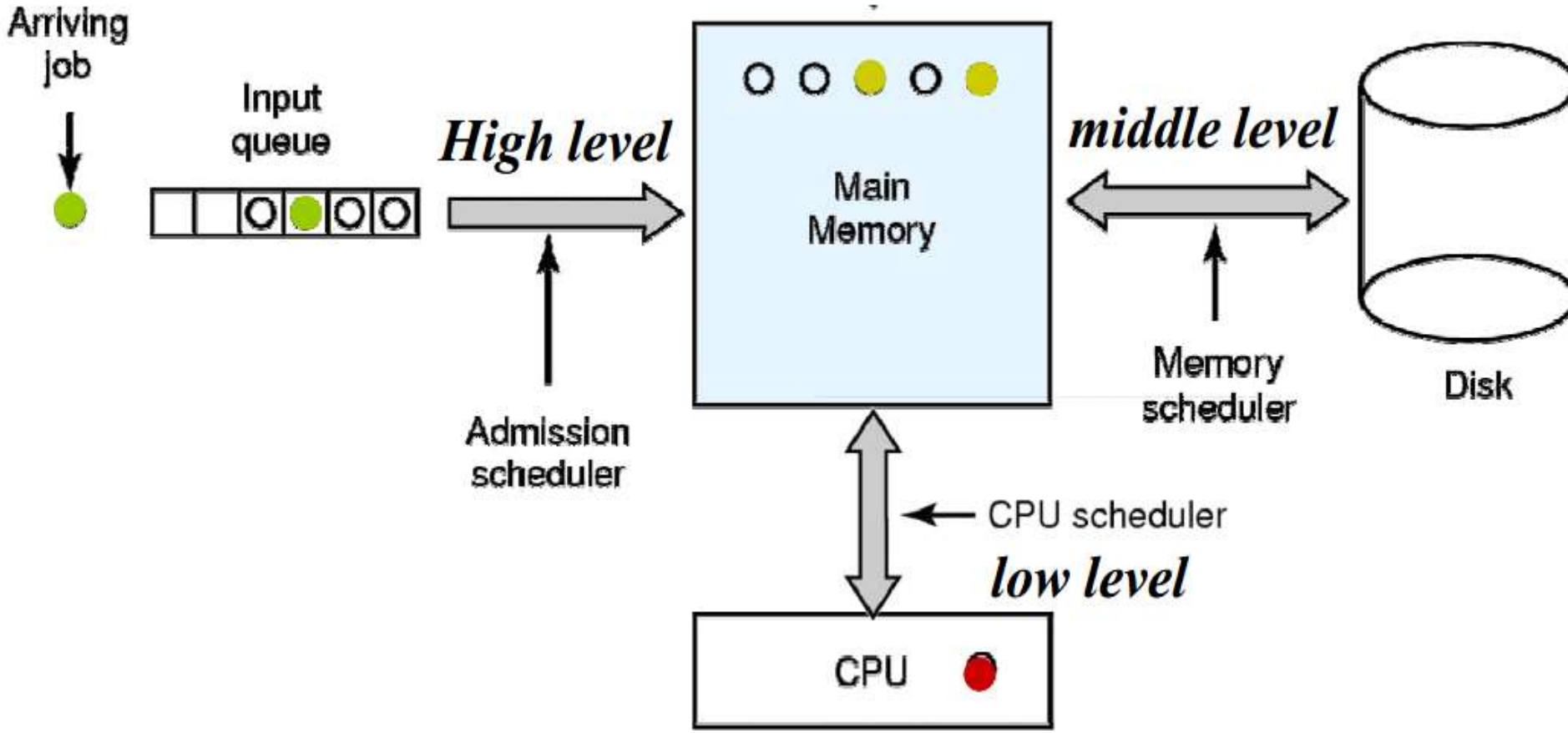
◆3、中级调度

- 中级调度，又叫做中程调度、内存调度。

  将暂不运行的进程，调至外存等待；将处于外存上的急需运行的进程，调入内存运行，即“对换”功能。
- 引入中级调度的目的主要是为了提高内存利用率和系统吞吐量。
- 调度对象：进程

3.1.1 处理机调度的层次

◆3、中级调度

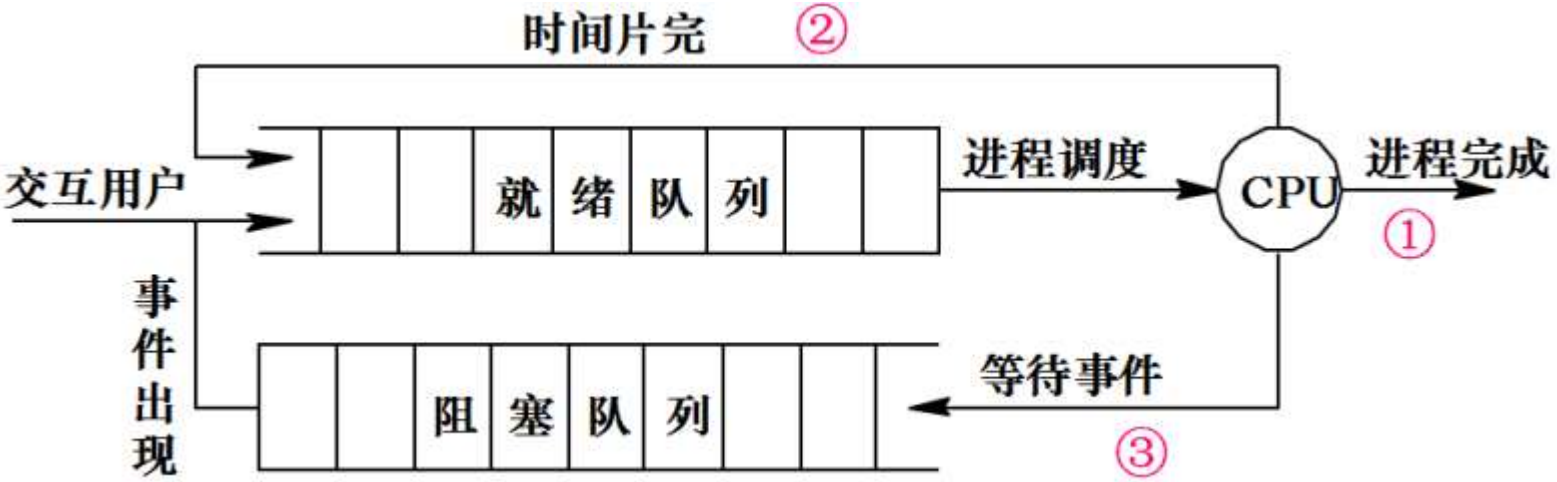


3.1.1 处理机调度的层次

◆调度队列模型

1）进程调度队列模型

●在分时系统中，进程调度的调度队列模型，通常把就绪进程组织成FIFO队列形式。每创建一个新进程，便将它挂在就绪队列的末尾，然后按时间片轮转方式运行。其模型如图1所示。



仅有进程调度的调度队列模型

3.1.1 处理机调度的层次

◆调度队列模型

2、高级与低级调度相结合的调度队列模型

●在批处理系统中，由作业调度将一批作业调入内存，并为其建立进程（由装入程序实现）并加入就绪队列后，择之一进占CPU。其模型如下图所示。该模型与仅有进程调度的模型主要区别有二。

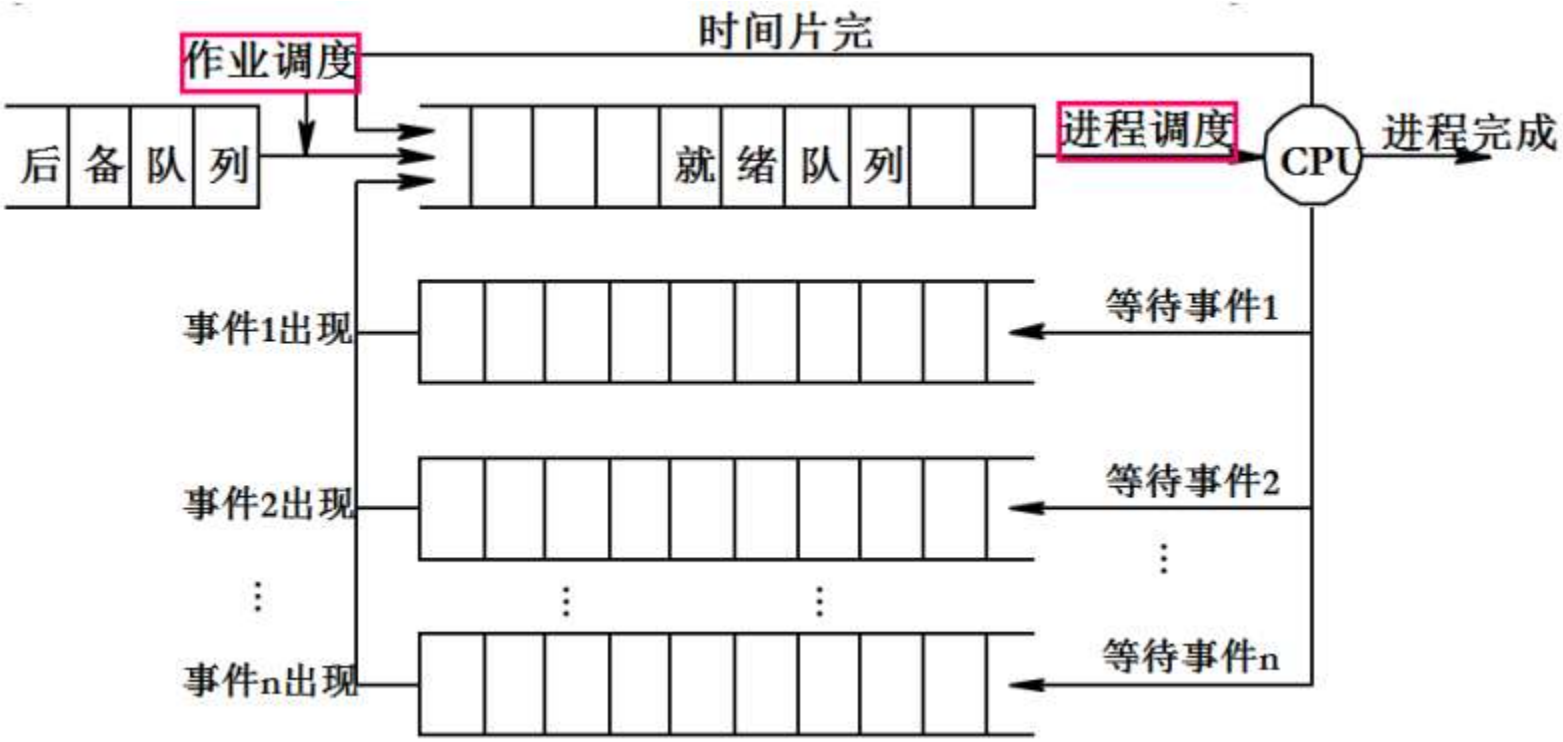
◆该调度队列模型，最常用的是最高优先权优先调度算法，最常用的就绪队列形式是优先权队列。

◆该模型设置多个阻塞队列。对于小型系统，可以只设置一个阻塞队列；但当系统较大时，若仍只有一个阻塞队列，其长度必然会很长，队列中的进程数可以达到数百个，这将严重影响对阻塞队列操作的效率。故在大、中型系统中通常都设置了若干个阻塞队列，每个队列对应于某一种进程阻塞事件。

3.1.1 处理机调度的层次

◆调度队列模型

2、高级（作业）与低级（进程）调度相结合的调度队列模型



3.1.1 处理机调度的层次

◆调度队列模型

3、具有高、中、低三级调度的队列模型（重要）

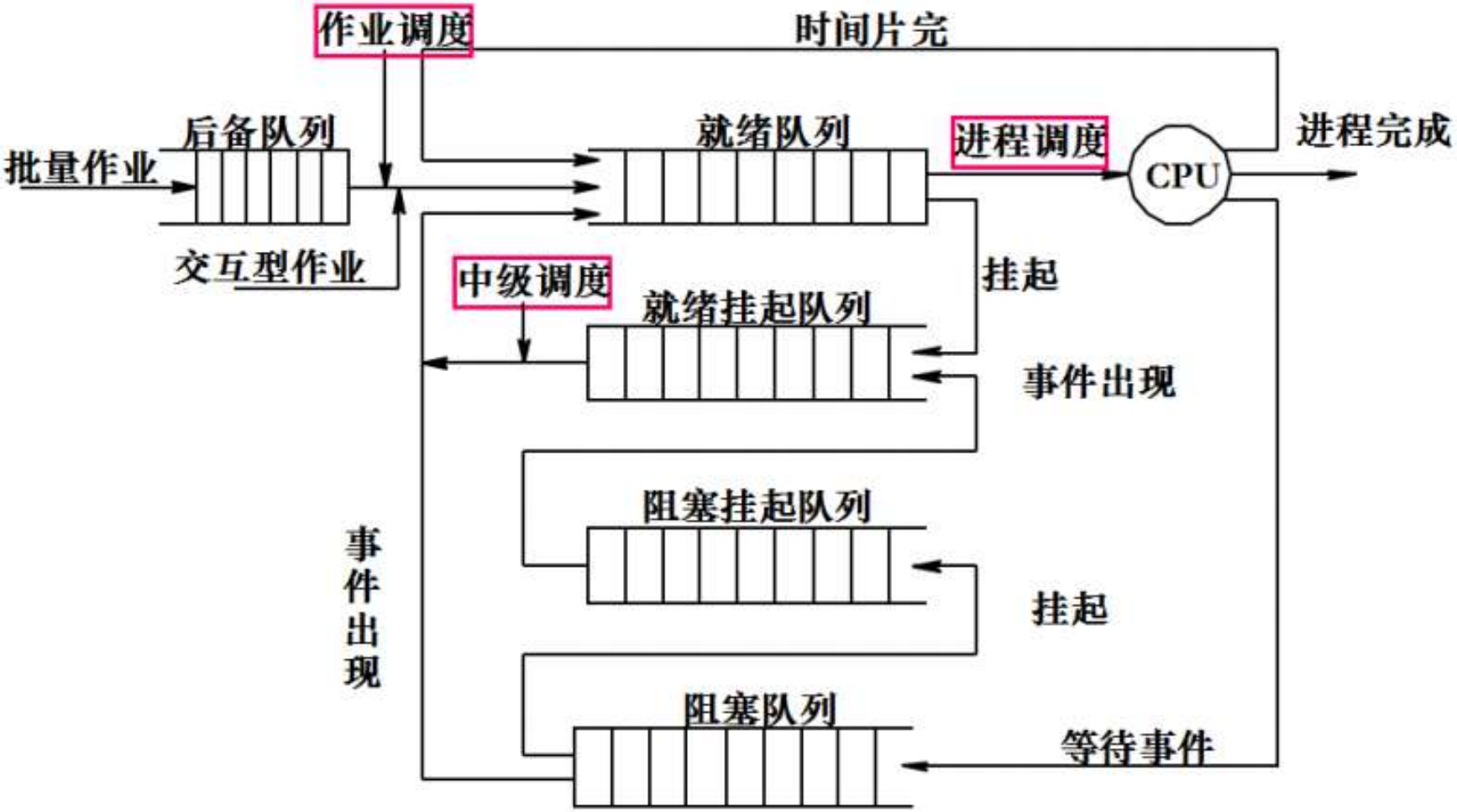
●为了提高内存 利用率，中级调度的基本思想就是把进程的就绪状态分为内存就绪和外存就绪两种状态。同样，也可把阻塞状态进一步分成内存阻塞和外存阻塞两种状态。

●在中级调度进程的操作下，可使进程状态由内存就绪转为外存就绪，由内存阻塞转为外存阻塞；在中级调度的作用下，又可使外存就绪转为内存就绪，其模型如图3所示。

●高中低三级调度结构图更能说明三种调度之间的内在关系。

3.1.1 处理机调度的层次

◆调度队列模型



3.1.1 处理机调度的层次

◆三种调度总结

	高级调度	低级调度	中级调度
调度对象	作业	进程	进程
调度目的	作业进内存、创建进程并置入就绪队列	选择就绪进程分配CPU	进程挂起转至外存
适用 OS	批处理系统	各种OS	应用较广泛
调度频率（间隔时间）	最低（几分钟）	最高（10-100ms）	介于高、低级两者之间

3.1.2 作业和作业调度

◆作业的概念

- 作业由程序、数据以及作业说明书组成。

  系统根据该说明书来对程序的运行进行控制。

  多道批处理系统中，是以作业为基本单位从外存调入内存的。

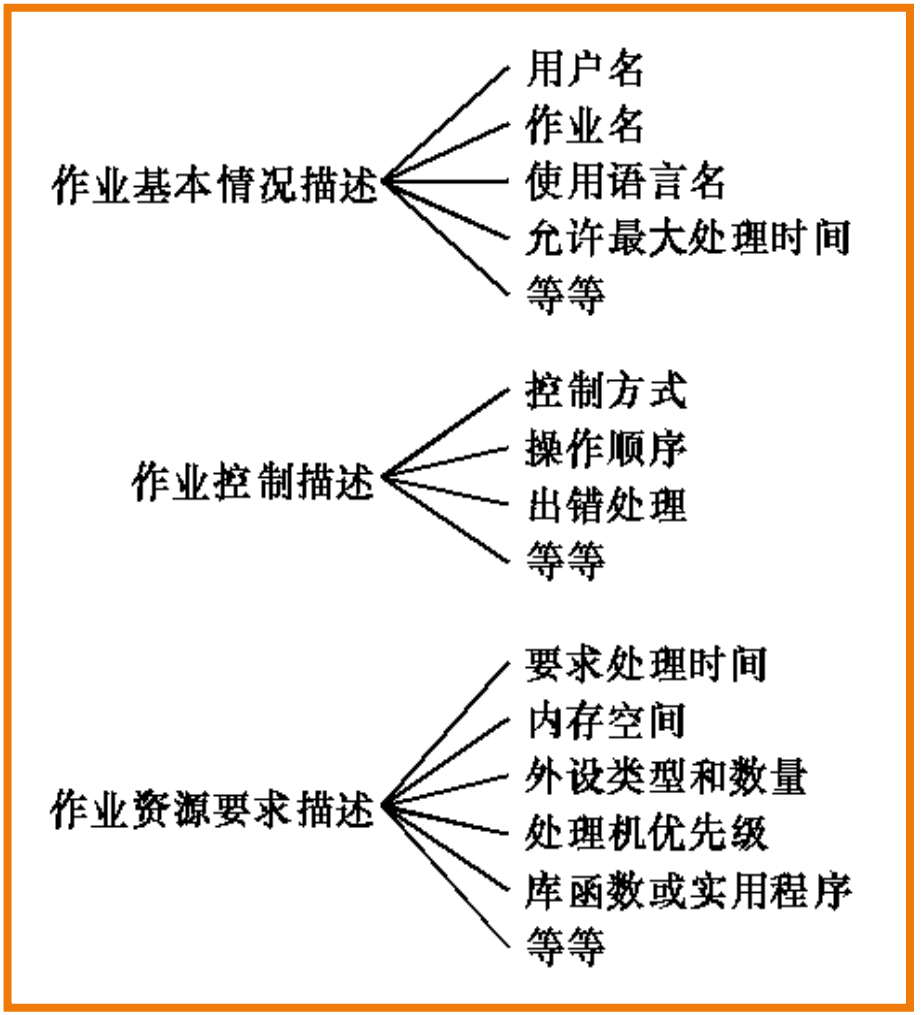
3.1.2 作业和作业调度

◆作业的概念

作业说明书

●作业说明书的表现形式为JCB，其信息包括：

- 基本信息
- 控制信息
- 资源信息



3.1.2 作业和作业调度

◆作业的概念

- 作业步(Job Step):是在作业运行期间，每个作业都必须经过的若干相对独立又相互关联的顺序步骤。

  例如:一个典型的作业可分成:

  ①“编译”作业步；②“链接”作业步；

  ③“装配”作业步；　④“运行”作业步。

- 作业流(JobStream):是将相互关联的作业按照一定的依赖关系组织而成的一个作业执行序列，是实现作业执行流程自动化的一种解决方案。

3.1.2 作业和作业调度

◆作业控制块

- 作业控制块（JCB）是作业在系统中存在的标志，保存了系统对作业进行管理和调度所需的信息。
- JCB通常含有：作业标识、用户名称、用户帐户、作业类型、作业状态、调度信息、资源需求、进入系统时间、开始处理时间、作业完成时间、作业退出时间、资源使用情况等

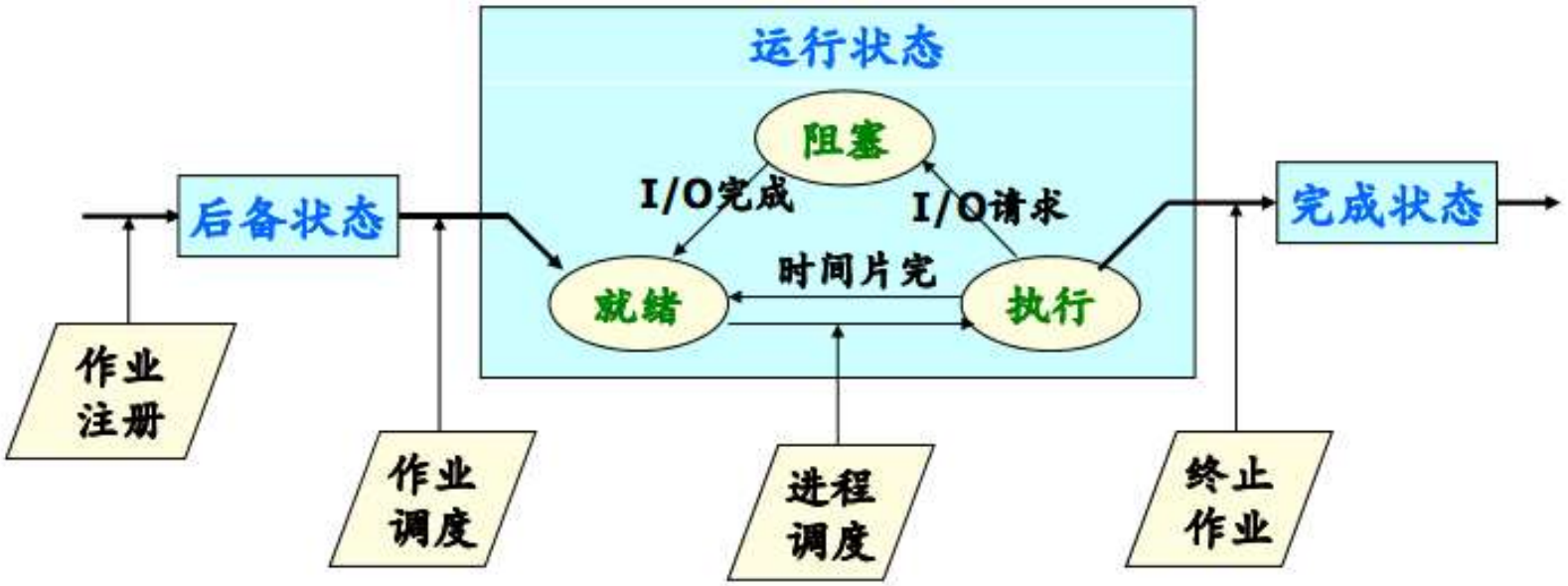
● 作业执行流程：

进入系统—>(作业注册程序)建立JCB —>插入相应后备队列—>作业调度—>作业运行—>作业结束—>回收资源—>撤销JCB。

3.1.2 作业和作业调度

◆批处理中的作业调度

- 作业调度：一个作业进入系统到运行结束，一般需要经历收容、运行、完成三个阶段，与之相对应的是作业的三种状态:后备状态、运行状态、完成状态。



3.1.2 作业和作业调度

◆作业调度的主要任务

- **作业调度的主要任务：**是根据一定的算法从外存中选取作业调入内存，为其创建相应的进程并分配资源，然后将进程插入就绪队列。
- **调度决策：**

（1）**接纳作业的数量（取决于多道程序）**：数量太多会导致平均周转时间显著延长，影响系统的服务质量；数量太少，系统效率低。

（2）**接纳哪些作业（取决调度算法）**：

先来先服务、短作业优先、基于作业优先级等

3.1.3 进程调度

进程调度包括调度任务、调度机制、调度方式

◆1、进程调度度任务

(1) 保存CPU现场信息。

如程序计数器、多个通用寄存器中的内容。

(2) 按某种算法选取进程。

按照调度算法，从就绪队列中取一个进程，将它的状态由就绪态转换为运行态。

(3) 把CPU分配给进程

分派程序将CPU分配给上述进程，并将其PCB中的现场信息恢复，使得该进程能从上次的断点处恢复运行。

3.1.3 进程调度

◆2、进程调度机制

- 1）排队器（机制）：将系统中的所有就绪进程排队，当有进程转变为就绪态时将其插入相应的就绪队列。
- 2）分派器（分派程序）：将进程调度程序选择的进程从就绪队列中取出，并将CPU分配给该进程。
- 3）上下文切换器（运行环境的切换），包括两对切换：

  待终止进程与分派程序的上下文切换

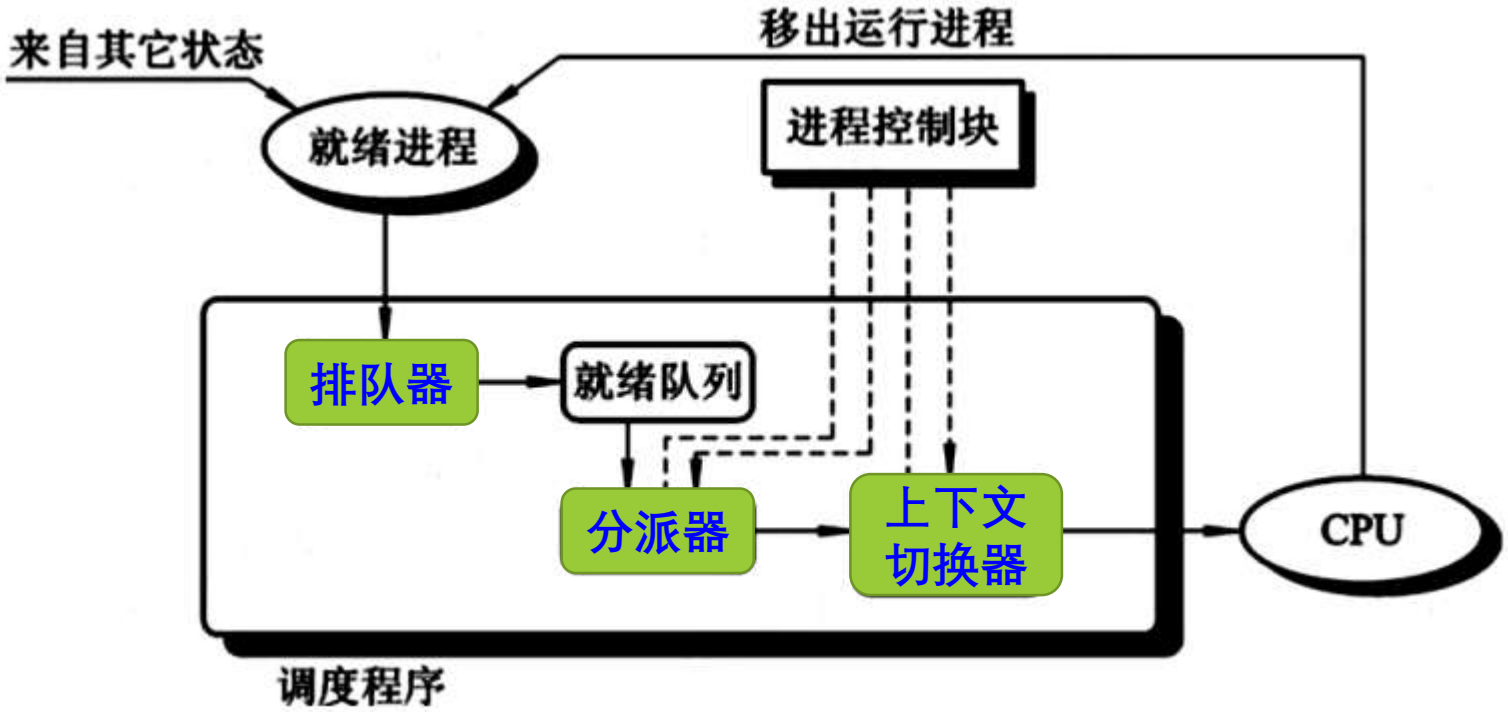
  分派程序与待执行进程的上下文切换

3.1.3 进程调度

◆2、进程调度机制

- 3）上下文切换器（运行环境的切换），包括两对切换：

待终止进程与分派程序的上下文切换



3.1.3 进程调度

◆3、进程调度方式

（1）非抢占调度方式

非抢占式是指一旦把CPU分配给某进程，便一直执行，直至该进程完成或发生某事件而被阻塞时，才再把CPU分配给其他进程，决不允许某进程抢占已经分配出去的CPU。

（一旦分配，直到完成或阻塞才可重新分配CPU使用权）

优点：实现简单、系统开销小，适合批处理系统。

缺点：实时性差，不能用于分时系统和大多数实时系统。

3.1.3 进程调度

◆3、进程调度方式

调度时机（采用非抢占调度，可能引起进程调度的因素）：

1）进程正常终止或异常终止。

2）进程因某种原因阻塞，如正在执行进程提出I/O请求。

3）在进程通信或同步过程中执行了某种原语，如Block原语。

3.1.3 进程调度

◆3、进程调度方式

（2）抢占调度方式

抢占式允许调度程序根据某种原则去暂停某个正在执行的进程，将已分配给该进程的处理机重新分配给另一进程 。

（更高优先级新的进程可以抢占当前进程的CPU使用权）

优点：公平、实时性好。

缺点：系统开销大。适合实时系统和分时系统。

抢占原则：主要有优先权原则、短进程优先原则、时间片原则

3.1.3 进程调度

◆3、进程调度方式

抢占原则：

优先权原则：允许优先权高的新到进程抢占当前进程的处理机。

短作业优先原则：短作业可以抢占当前较长作业的处理机。

时间片原则：各进程按时间片运行，当一个时间片用完后，便停止该进程的执行而重新进行调度。

3.1.4 处理机调度算法的目标

◆处理机调度算法的共同目标

- 处理机调度的共同目标
- 批处理系统的目标
- 分时系统的目标
- 实时系统的目标

3.1.4 处理机调度算法的目标

◆1、处理机调度算法的共同目标

(1) 资源的利用率：

$$\text{CPU的利用率}=\frac{\text{CPU有效工作时间}}{\text{CPU有效工作时间}+\text{CPU空闲时间}}$$

(2)公平性：防止饥饿、兼顾紧急程度与重要性。

(3)平衡性：保证资源使用的平衡。

(4)策略强制执行：对制订的策略，必须予以准确的执行，即会使造成某些工作的延迟也要执行。

3.1.4 处理机调度算法的目标

◆2、批处理系统中处理机调度算法的目标

(1) 平均周转时间短

概念:作业周转时间是指从作业被提交给系统开始到作业完成为止的这段时间间隔。

组成：作业周转时间主要包括四部分:作业在外存后备队列上等待(作业)调度的时间、进程在就绪队列上等待进程调度的时间、进程在CPU上执行的时间以及进程等待I/O操作完成的时间。（一个作业过程中后3部分可多次出现）

3.1.4 处理机调度算法的目标

◆2、批处理系统中处理机调度算法的目标

指标:

平均周转时间：

$$T = \frac{1}{n}\left[\sum_{i=1}^{n} T_i\right]$$

带权平均周转时间：

$$W = \frac{1}{n}\left[\sum_{i=1}^{n} \frac{T_i}{T_{si}}\right]$$

Ti为第i个作业的周转时间，T_{Si}为OS实际给第i个作业服务的时间周转时间=完成时间-提交时间，带权周转时间=周转时间/服务时间

3.1.4 处理机调度算法的目标

◆2、批处理系统中处理机调度算法的目标

(2) 系统吞吐量高

吞吐量:单位时间内系统完成的作业数，与批处理作业的平均长度有关。

(3) 处理机利用率高

3.1.4 处理机调度算法的目标

◆3、分时系统中处理机调度算法的目标

(1)响应时间快

概念：响应时间是从用户通过键盘提交一个请求开始，直至系统首次产生响应为止的时间。

组成：响应时间包括三部分时间:从键盘输入的请求信息传送到处理机的时间、处理机对请求信息进行处理的时间、以及将所形成的响应信息回送到终端显示器的时间。

(2)均衡性:响应时间的快慢应与用户请求服务的复杂性相适应。

3.1.4 处理机调度算法的目标

◆4、实时系统中处理机调度算法的目标

(1)截止时间的保证

概念：截止时间是指某任务必须开始执行的最迟时间(开始截止时间）或必须完成的最迟时间(完成截止时间)。

硬实时任务与软实时任务对截止时间要求不一样

(2)可预测性

什么时间做什么工作应该是可预测的。

3.1.4 处理机调度算法的目标

◆调度算法的目标—小结

1. 面向用户的准则

- 周转时间短（评价批处理系统的指标）

  周转时间；　　平均周转时间

  带权周转时间；　平均带权周转时间

- 响应时间快：　　响应时间（评价分时系统的指标）
- 截止时间的保证：　截止时间（评价实时系统的指标）
- 优先权准则

3.1.4 处理机调度算法的目标

◆调度算法的目标—小结

2、面向系统的准则

- 系统吞吐量高 （评价批处理系统的指标）
- 处理机利用率好
- 各类资源的平衡利用

第3章 处理机调度与死锁

3.2 调度算法

◆应用范围与含义

1、 先来先服务调度算法

2、 短作业优先调度算法

3、 优先级调度算法

4、 轮转调度算法

5、 多级队列调度算法

6、多级反馈队列调度算法

7、基于公平原则的调度算法

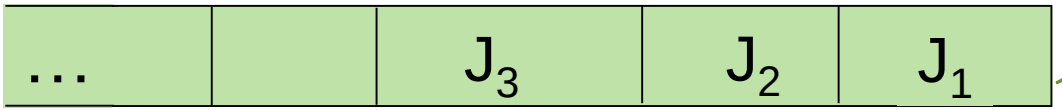
作业调度、进程调度均适用

3.2.1 先来先服务调度算法

◆FCFS 含义

作业调度:按进入后备队列的先后顺序选择一个或多个作业，将它们调入内存，为它们分配资源、创建进程，并放入就绪队列。

进程调度:按照进程就绪的先后次序来调度进程，为之分配处理机,使之投入运行。



简单、公平
易编程实现

3.2.1 先来先服务调度算法

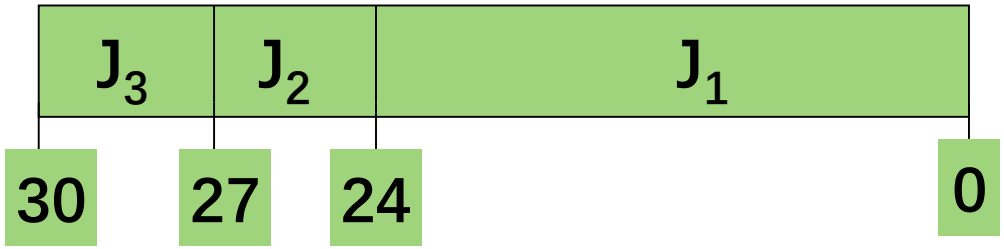
◆先来先服务调度算法（FCFS）

按照作业到达的先后次序来进行调度

作业	运行时间
J1	24
J2	3
J3	3

假定作业到达顺序如下: J1 , J2 , J3

该调度的甘特图(Gantt)为:



- 平均等待时间 =(0 + 24 + 27)/3 = 17
- 平均周转时间 = (24 + 27 + 30)/3 = 27

3.2.1 先来先服务调度算法

◆先来先服务调度算法（FCFS）

假定作业到达顺序如下: J2 , J3，J1， 该调度的甘特图为:

- 平均等待时间 = (6 + 0 + 3)/3 = 3
- 平均周转时间 = (30+ 3 + 6)/3 = 13

比较结论：比前例好得多，此结果产生是由于短进程先于长进程到达

3.2.1 先来先服务调度算法

◆实例 下表列出了A、B、C、D四个作业分别到达系统的时间、要求服务的时间、开始执行的时间及各自的完成时间，并计算出各自的周转时间和带权周转时间。

进程名	到达时间	服务时间	开始执行时间	完成时间	周转时间	带权周转时间
A	0	1	0			
B	1	100	1			
C	2	1	101			
D	3	100	102			

从表上可以看出，其中短作业C的带权周转时间竟高达100，这是不能容忍的；而长作业D的带权周转时间仅为1.99。FCFS调度算法有利于CPU繁忙型的作业，而不利于I/O繁忙型的作业（进程）

3.2.1 先来先服务调度算法

◆实例

下表列出了A、B、C、D四个作业分别到达系统的时间、要求服务的时间、开始执行的时间及各自的完成时间，并计算出各自的周转时间和带权周转时间。

进程名	到达时间	服务时间	开始执行时间	完成时间	周转时间	带权周转时间
A	0	1	0	1	1	1
B	1	100	1	101	100	1
C	2	1	101	102	100	100
D	3	100	102	202	199	1.99

从表上可以看出，其中短作业C的带权周转时间竟高达100，这是不能容忍的；而长作业D的带权周转时间仅为1.99。FCFS调度算法有利于CPU繁忙型的作业，而不利于I/O繁忙型的作业（进程）

3.2.1 先来先服务调度算法

◆优缺点

有利于长作业/进程，不利于短作业/进程。

有利于CPU密集/繁忙型的作业，不利于I/O密集/繁忙型的作业。

注：CPU密集型进程—进程运行时需要大量的CPU时间，而很少请求I/O。

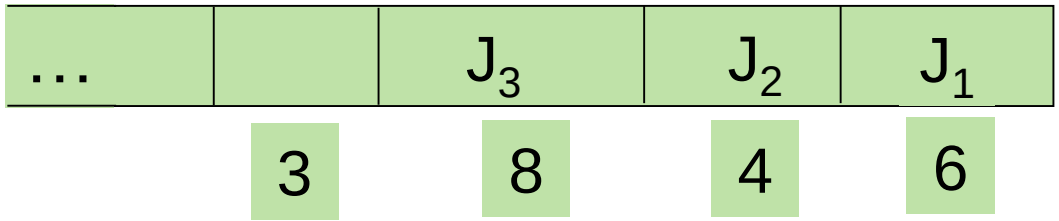
IO密集型进程—进程运行时需要频繁请求I/O。随着计算机运行速度加快，进程越来越趋向于I/O密集型进程。

3.2.2 短作业优先调度算法

◆SJF含义

短作业优先调度（SJF）:从后备队列选择一个或多个估计运行时间最短的作业，将它们调入内存，为它们分配资源、创建进程，并放入就绪队列。

短进程优先调度（SPF）:从就绪队列中选择估计运行时间最短的进程，将处理机分配给它，使它立即执行。



3.2.2 短作业优先调度算法

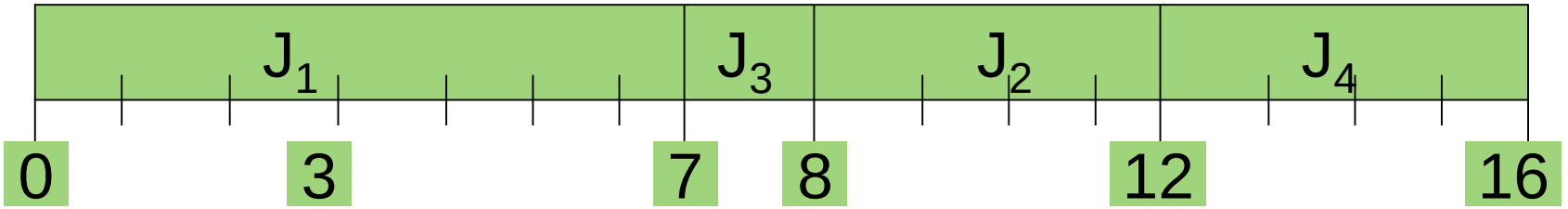
◆SJF含义

- ◆ 对进程调度，SJF有两种模式：
  - ➢非抢占式SJF
  - ➢抢占式SJF：–抢占发生在有比当前进程剩余时间片更短的进程到达时，也称为最短剩余时间优先调度
- ◆ SJF是最优的（对一组指定的进程而言），它给出了最短的平均等待时间。

3.2.2 短作业优先调度算法

◆非抢占式SJF举例

进程	到达时间	运行时间
J_1	0.0	7
J_2	2.0	4
J_3	4.0	1
J_4	5.0	4



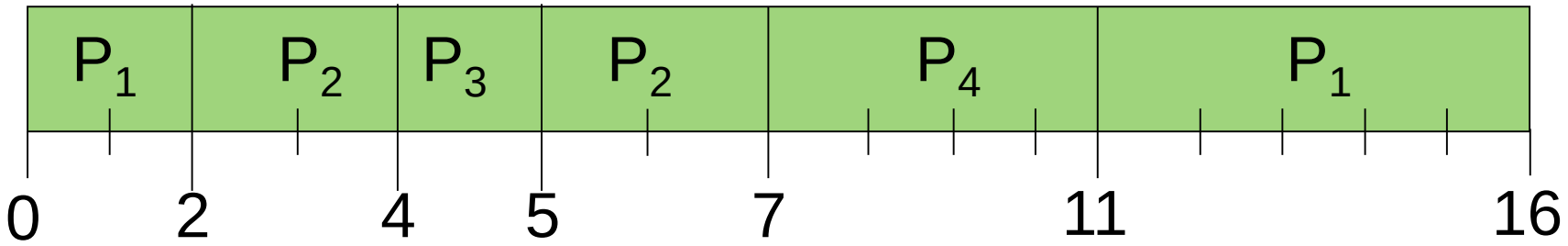
- 平均等待时间 = (0 + 6 + 3 + 7)/4 = 4
- 平均周转时间=(7+10+4+11)/4= 8

3.2.2 短作业优先调度算法

◆抢占式SJF举例

前例:

进程	到达时间	区间时间
P1	0	7
P2	2	4
P3	4	1
P4	5	4



- 平均等待时间 = (9 + 1 + 0 +2)/4 = 3
- 平均周转时间 = (16+ 5 +1+ 6)/4 = 7

3.2.2 短作业优先调度算法

◆示例

作业情况 / 调度算法	进程名	A	B	C	D	E	平均
	到达时间	0	1	2	3	4	
	服务时间	4	3	5	2	4	
FCFS	完成时间						
	周转时间						
	带权周转时间						
SJF	完成时间						
	周转时间						
	带权周转时间						

采用SJF算法后，不论是平均周转时间还是平均带权周转时间，都较FCFS调度算法有明显的改善，尤其是对短作业D。而平均带权周转时间从2.8降到了2.1。这说明SJF调度算法能有效的降低作业的平均等待时间，提高系统吞吐量。

3.2.2 短作业优先调度算法

◆示例

作业情况 / 调度算法	进程名	A	B	C	D	E	平均
	到达时间	0	1	2	3	4	
	服务时间	4	3	5	2	4	
FCFS	完成时间	4	7	12	14	18	
	周转时间	4	6	10	11	14	9
	带权周转时间	1	2	2	5.5	3.5	2.8
SJF	完成时间	4	9	18	6	13	
	周转时间	4	8	16	3	9	8
	带权周转时间	1	2.67	3.2	1.5	2.25	2.1

采用SJF算法后，不论是平均周转时间还是平均带权周转时间，都较FCFS调度算法有明显的改善，尤其是对短作业D。而平均带权周转时间从2.8降到了2.1。这说明SJF调度算法能有效的降低作业的平均等待时间，提高系统吞吐量。

3.2.2 短作业优先调度算法

◆SJF优缺点

SJF优点：能有效地降低作业的平均等待时间，提高系统吞吐量。

SJF缺点：①必须预知作业的运行时间，用户对作业的运行时间估计并不一定准确，不一定能真正做到短作业优先；

② 对长作业不利；

③ 人机无法实现交互；

④ 未考虑作业紧迫程度（FCFS也存在这个缺点)。

3.2.3 优先级调度算法

◆优先级调度算法（PR）

含义：基于作业的紧迫程度，由外部赋予作业(进程)相应的优先级，然后据此进行调度。

高响应比优先调度算法是一种优先级调度算法，用于作业调度。

注:虽然FCFS的等待时间、SJF/SPF的长短也可看作是优先级，但是并不能反映作业/进程的紧迫程度，并不属于此处所说的优先级调度算法。

3.2.3 优先级调度算法

◆1、优先级调度算法（PR）的类型

（1）非抢占式优先权算法、（2）抢占式优先权算法

◆2、优先级的类型

（1）静态优先级： 静态优先级一经确定将不再改变。

优点：简单易行，系统开销小;

缺点：不够精确，可能会使优先级低的作业或进程长期得不到调度。

（2）动态优先级： 动态优先级在创建进程时确定后，将随进程的推进或等待时间的增加而动态调整。

3.2.3 优先级调度算法

◆3、高响应比优先调度算法（HRRN）

（1）优先级：

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

（2）响应比：

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

3.2.3 优先级调度算法

◆3、高响应比优先调度算法（HRRN）

（1）特点：

① 如果作业等待时间相同，则要求服务时间（即处理时间)越短，响应比越高，有利于短作业。类似SJF。

② 如运行时间相同，取决于等待时间，类似于FCFS。即等待时间越长，响应比越高。

③ 对于长作业，随等待时间增加，响应比增高，最后同样可获得处理机。

（2）缺点：每次调度之前都需要进行响应比的计算，增加开销。

3.2.3 优先级调度算法

◆进程优先级高低的确定依据

（1）进程类型：

系统进程高于一般用户进程的优先级

（2）进程对资源的要求，对CPU和内存要求少、运行时间短的优先级高些

（3）用户要求：

根据用户进程的紧迫程度和用户所付费用决定

3.2.3 优先级调度算法

◆优先级调度算法（PR）的优缺点

（1）优点

实现简单，考虑了进程的紧迫程度灵活，可模拟其它算法

（2）缺点：

存在饥饿问题，低优先级的进程可能永远得不到运行。

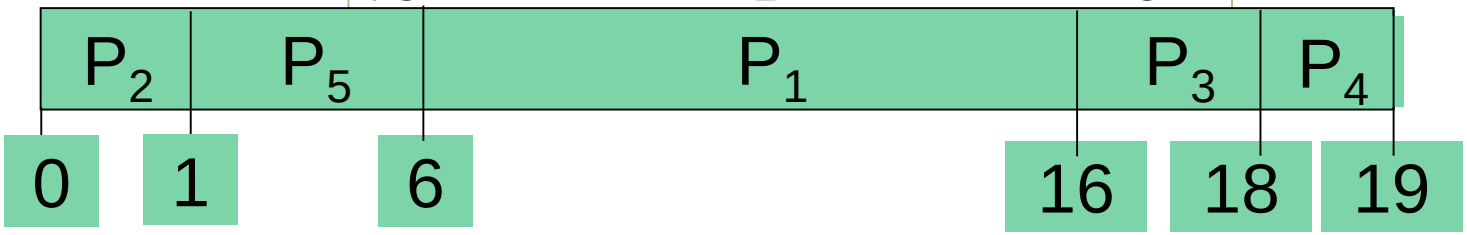
（3）解决方法：

视进程等待时间的延长提高其优先数。

3.2.3 优先级调度算法

◆非抢占PR算法案例

进程	优先级	运行时间
P1	3	10
P2	1	1
P3	3	2
P4	4	1
P5	2	5

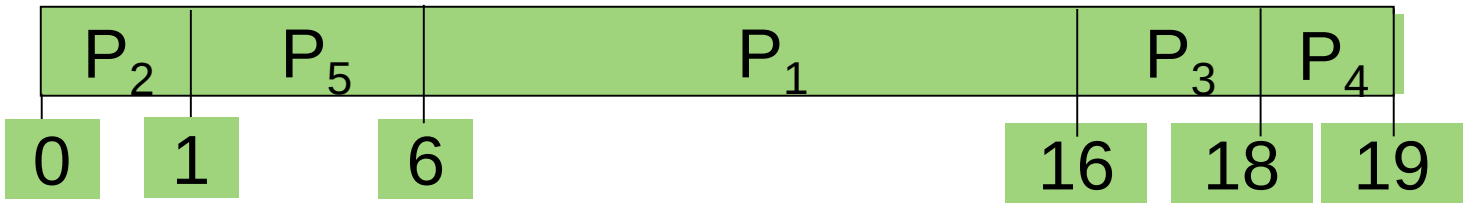


- 平均等待时间 = (0 + 5 + 16+18+1)/5 =8
- 平均周转时间 = (16 + 1 + 18+19+6)/5 = 12

3.2.3 优先级调度算法

◆非抢占PR案例

进程	优先级	运行时间
P1	3	10
P2	1	1
P3	3	2
P4	4	1
P5	2	5



- 平均等待时间 = (0 + 5 + 16+18+1)/5 =8
- 平均周转时间 = (16 + 1 + 18+19+6)/5 = 12

3.2.3 优先级调度算法

◆FCFS/SJF/HRRN调度示例

作业	进入时间	估计运行时间（min）	开始时间	结束时间	周转时间（min）	带权周转时间
JOB1	8:00	120				
JOB2	8:50	50				
JOB3	9:00	10				
JOB4	9:50	20				
作业的平均周转时间T= 作业带权平均周转时间W=						

3.2.3 优先级调度算法

◆FCFS调度结果

作业	进入时间	估计运行时间（min）	开始时间	结束时间	周转时间（min）	带权周转时间
JOB1	8:00	120	8:00	10:00	120	1
JOB2	8:50	50	10:00	10:50	120	2.4
JOB3	9:00	10	10:50	11:00	120	12
JOB4	9:50	20	11:00	11:20	90	4.5
作业的平均周转时间T=112.5 作业带权平均周转时间W=4.975					450	19.9

JOB1 >>JOB2 >>JOB3 >>JOB4

3.2.3 优先级调度算法

◆SJF调度结果

作业	进入时间	估计运行时间（min）	开始时间	结束时间	周转时间（min）	带权周转时间
JOB1	8:00	120	8:00	10:00	120	1
JOB2	8:50	50	10:30	11:20	150	3
JOB3	9:00	10	10:00	10:10	70	7
JOB4	9:50	20	10:10	10:30	40	2
作业的平均周转时间T=95 作业带权平均周转时间=3.25					380	13

JOB1 >>JOB3 >>JOB4 >>JOB2

3.2.3 优先级调度算法

◆HRRN调度结果

作业	进入时间	估计运行时间（min）	开始时间	结束时间	周转时间（min）	带权周转时间
JOB1	8:00	120	8:00	10:00	120	1
JOB2	8:50	50	10:10	11:00	130	2.6
JOB3	9:00	10	10:00	10:10	70	7
JOB4	9:50	20	11:00	11:20	90	4.5
作业的平均周转时间T=102.5 作业带权平均周转时间=3.775					410	15.1

JOB1 >>JOB3 >>JOB2 >>JOB4

3.2.4 轮转调度算法

◆1、轮转（RR）调度算法基本原理

在轮转(RR)法中，系统将所有的就绪进程按 FCFS策略排成一个就绪队列。就绪队列各进程轮流获得一个时间片的cpu 使用权。

专为分时系统设计，类似于FCFS，但增加了抢占

2、进程切换时机

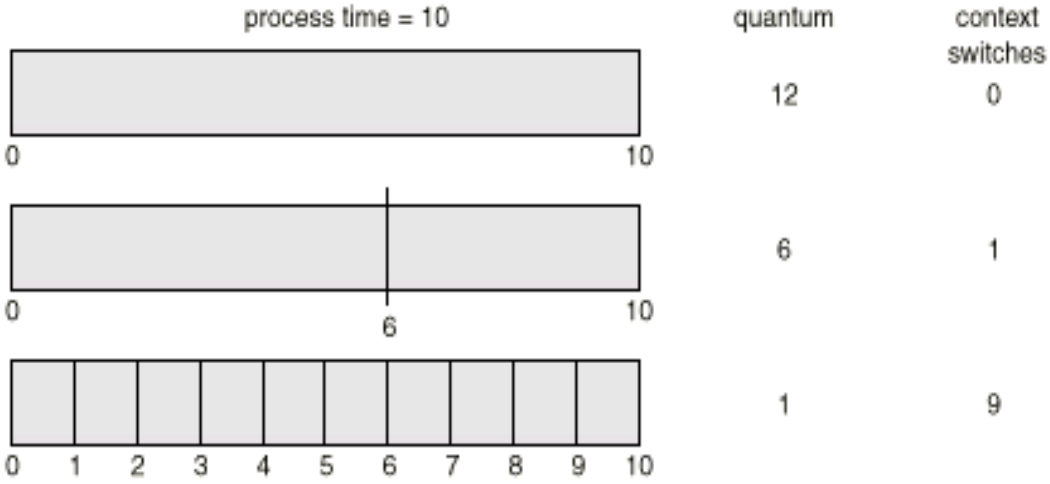
① 若一个时间片尚未用完，正在运行的进程便已经完成。则激活调度程序，删除该进程，调度下一进程，开启新时间片。

② 在一个时间片用完时，激活计时器中断处理。

3.2.4 轮转调度算法

◆2、时间片大小选取

- 时间片太大，就退化为FCFS；无法满足短作业和交互式用户的需求。
- 时间片太小会导致频繁的进程切换，系统开销过大。
- 时间片设置应考虑：系统对响应时间的要求、就绪队列中进程的数目以及系统的处理能力。
- 较为可取的大小：略大于一次典型交互所需要的时间。



3.2.4 轮转调度算法

◆时间片轮转调度实例（P82）

作业情况 / 时间片	进程名	A	B	C	D	E	平均
	到达时间	0	1	2	3	4	
	服务时间	4	3	4	2	4	
RR q=1	完成时间	15	12	16	9	17	
	周转时间	15	11	14	6	13	11.2
	带权周转时间	3.75	3.67	3.5	3	3.25	3.35
RR q=4	完成时间	4	7	11	13	17	
	周转时间	4	6	9	10	13	8.4
	带权周转时间	1	2	2.25	5	3.25	2.7

3.2.5 多队列调度算法

◆多队列调度算法

单就绪队列调度存在的问题

无法满足不同用户对进程调度策略的不同要求，特别对多处理机系统，这一缺点更为突出。

原理： 将就绪队列按进程的类型或性质拆分成为若干个，不同就绪队列可采用不同的调度算法，同一就绪队列的进程还可设置不同优先级，不同就绪队列也可设置不同的优先级。

OS可根据进程所处队列安排相应的调度算法,满足不同用户需求。在多处理机系统中，为每个处理机分配不同的就绪队列。

3.2.5 多队列调度算法

◆多队列调度算法

就绪队列从一个分为多个，如:

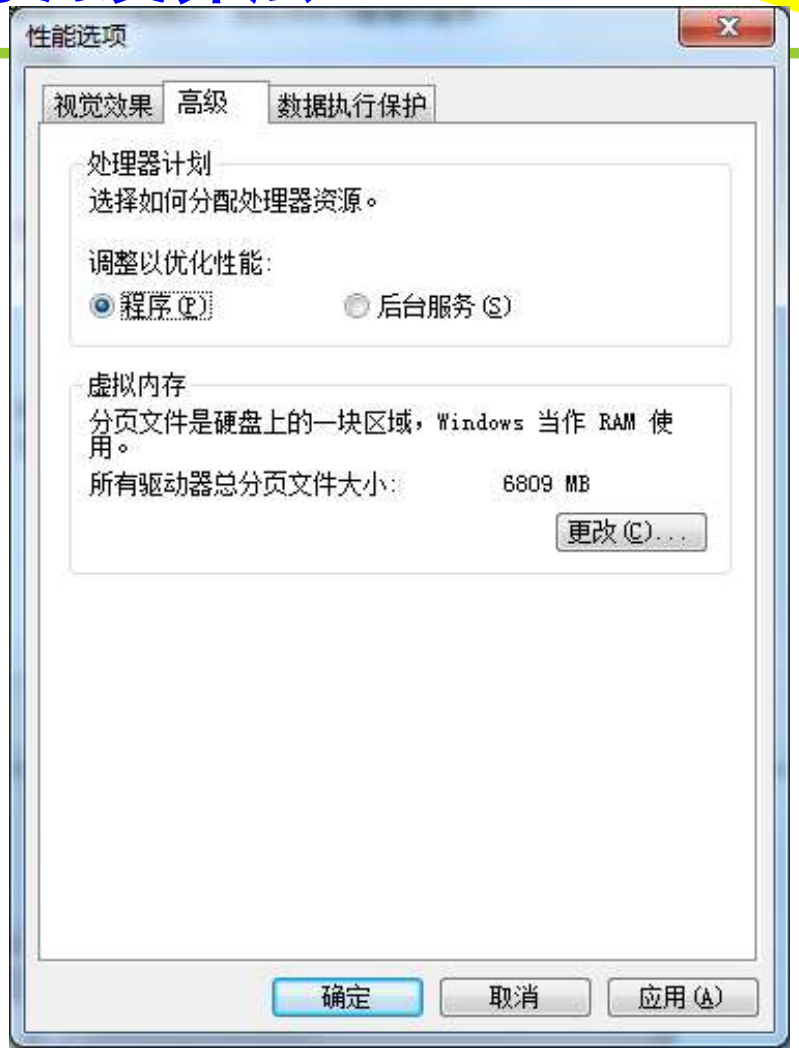
- 前台[交互式]
- 后台[批处理]

每个队列有自己的调度算法

- 前台 – RR
- 后台 – FCFS

调度须在队列间进行

- 固定优先级调度，即前台运行完后再运行后台，有可能产生饥饿。
- 给定时间片调度，即每个队列得到一定的CPU时间，进程在给定时间内执行;
- 如：80%的时间执行前台的RR调度，20%的时间执行后台的FCFS调度

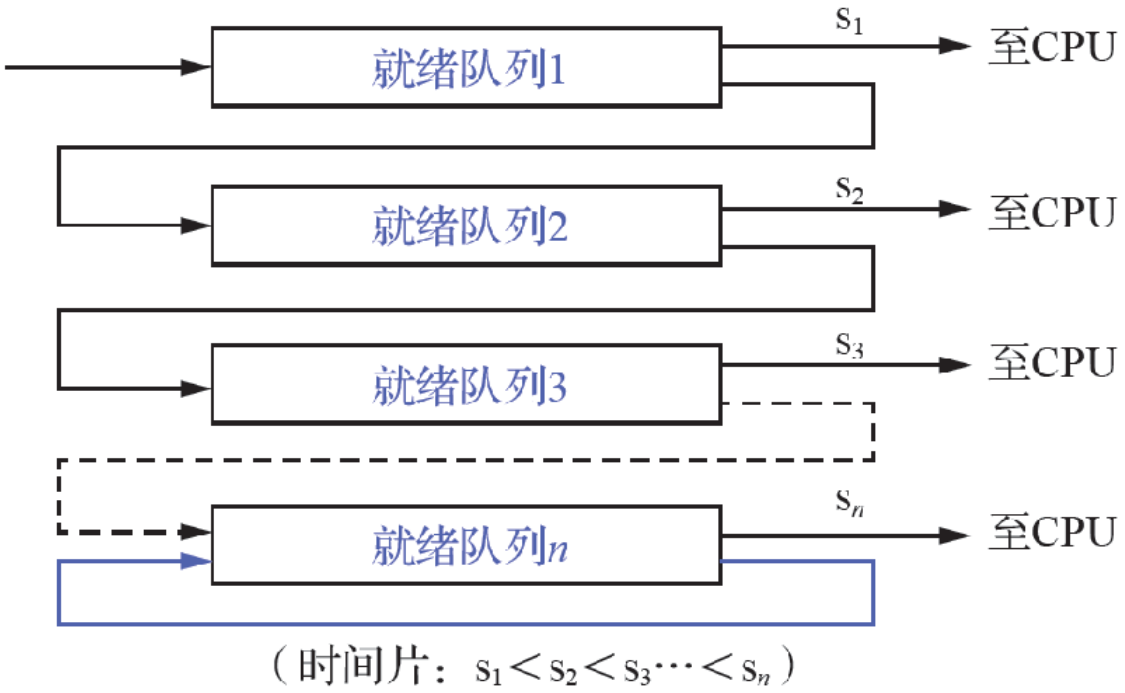


3.2.6 多级反馈队列调度算法

◆多级反馈队列调度算法

◆基本原理：

（1）设置多个就绪队列，并为各个队列优先级递减，但时间片长度倍增。

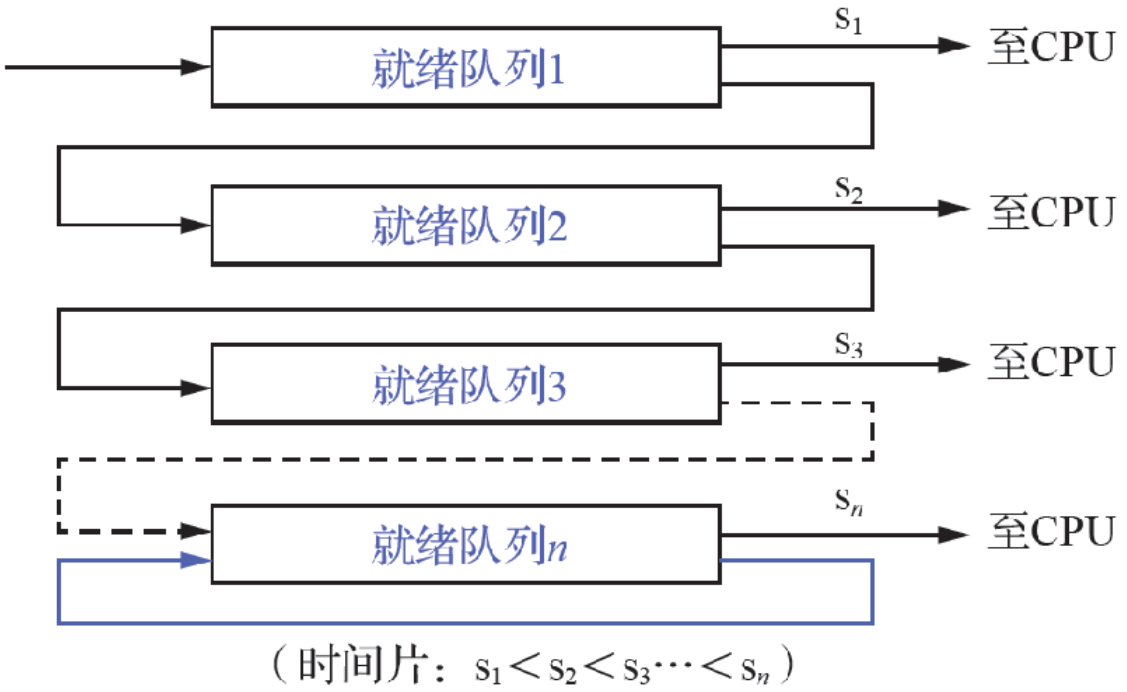


3.2.6 多级反馈队列调度算法

◆基本原理

（2）每个队列都采用FCFS调度。

① 当进程进入内存时，将它放入第一个队列的队尾，按FCFS原则等待调度。



（时间片：$S_1 < S_2 < S_3 \cdots < S_n$）

② 轮到该进程执行时，如果在时间片内完成则退出系统，若未完成则将其插到下一级队列的队尾，以此类推。

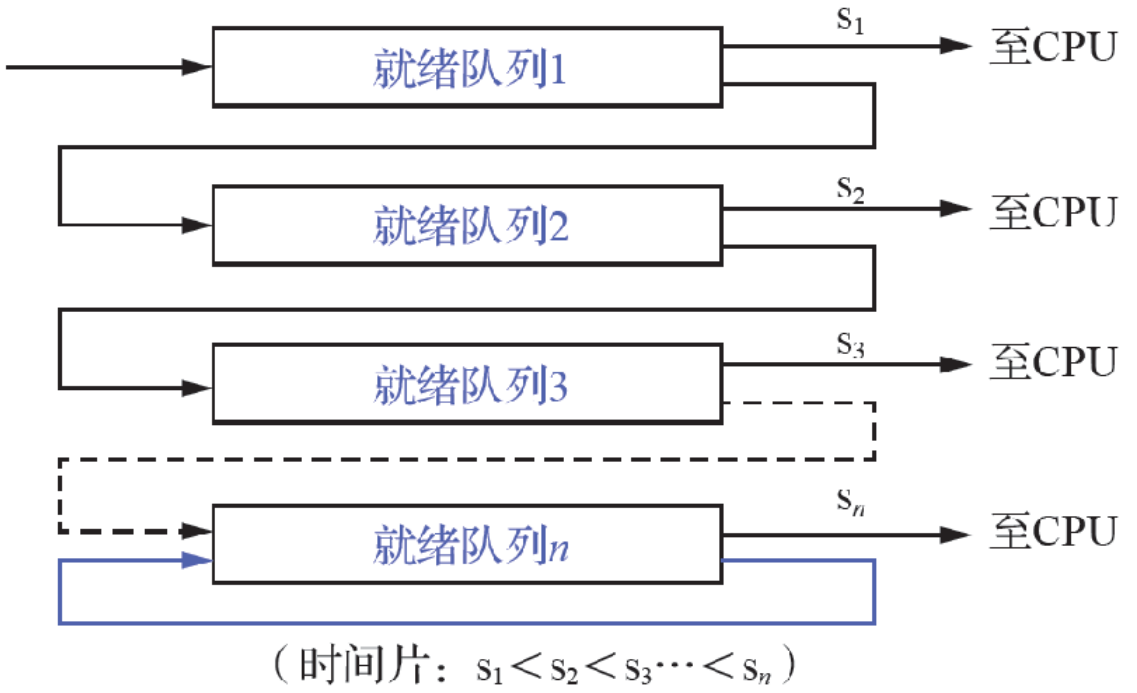
③ 当其到达第n个队列时则按照轮转调度算法的方式进行执行。

3.2.6 多级反馈队列调度算法

◆基本原理

（3）**按队列优先级调度。**

①当更高优先级的队列空时，下一级队列中的进程才能得到执行。



② 当在执行时**有新进程进入更高级队列**则将正在运行的进程插到当前队尾，处理机**分配给更高级队列**的进程。

3.2.6 多级反馈队列调度算法

◆算法性能：

当第一个队列的时间片选取合适时，能够很好地满足各类用户的需要：

① 终端型作业用户：通常在第一级队列的时间片可完成

② 短批处理作业用户：两至三次就可完成，周转时间较短

③ 长批处理作业用户：不必担心长期得不到调度，比简单轮转性能好。

3.2.7 基于公平原则的调度算法

◆ 基于公平原则的调度介绍2种：保障调度算法和公平分享调度算法

◆ 保证调度算法

该算法向用户做出的保证并不是优先运行，而是保证系统中相同类型的进程获得的CPU时间相同。

实施该算法时所必须的功能：

（1）跟踪每个进程自创建已执行的处理时间

（2）计算每个进程应获得的处理时间。

（3）计算每个进程获得的处理时间的比率。

（4）比较各进程获得的处理机时间的比率

（5）选取比率最小的进程获得CPU。

3.2.7 基于公平原则的调度算法

◆公平分享调度算法

公平分享调度算法则是保证两个用户获得相同的的CPU时间或者指定的CPU时间比率。

分配给每个进程相同的处理机时间，显然，这对诸进程而言，是体现了一定程度的公平，但如果各个用户所拥有的进程数不同，就会发生对用户的不公平问题。

- 调度的公平性主要针对用户而言；
- 使所有用户能获得相同的处理机时间或时间比例。

第3章 处理机调度与死锁

3.3.1 实现实时调度的基本条件

◆基本条件

在实时系统中，硬实时任务和软实时任务都联系着一个截止时间。为保证系统能正常工作，实时调度必须满足实时任务对截止时间的要求，为此，实现实时调度应具备下列4个条件：

1、提供必要的信息

2、系统处理能力强

3、采用抢占式调度机制

4、具有快速切换机制

3.3.1 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下述信息：

① 就绪时间：该任务成为就绪状态的时间。

② 开始截止时间、完成截止时间。只需知道一个。

③ 处理时间：从开始执行到完成所需时间。

④ 资源要求：任务执行时所需的一组资源。

⑤ 优先级：根据任务性质赋予不同优先级。

3.3.1 实现实时调度的基本条件

◆ 2、系统处理能力强

假如系统中有M个周期性的硬实时任务（HRT），处理时间为Ci，周期时间表示为Pi，

（1）单机系统必须满足条件：

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

（2）多处理机系统满足：

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

实例问题：

如果系统中有6个硬实时任务，他们的周期时间都是50ms，每次的处理时间是10ms.

该实时系统可以调度吗？

3.3.1 实现实时调度的基本条件

◆ 3、采用抢占式调度机制

广泛采用抢占调度方式:

抢占方式：对硬实时任务调度一般采用此方式

非抢占方式：适用于能预知任务开始截止时间的小型实时系统或软实时系统，参与调度的实时任务所需CPU时间应该较小，以便及时释放CPU.

◆4、采用快速切换机制（保证HRT及时运行）

- 具有快速响应外部中断能力
- 具有快速分派任务能力

3.3.2 实时调度算法的分类

可按照不同方式对实时调度算法加以分类：

- 根据实时任务性质的不同，分为硬实时调度算法和软实时调度算法；
- 按调度方式的不同，分为非抢占调度算法和抢占调度算法；
- 根据调度程序调度时间的不同，分为静态调度算法和动态调度算法。
- 多处理机环境下，可分为集中式调度和分布式调度两种算法。

3.3.2 实时调度算法的分类

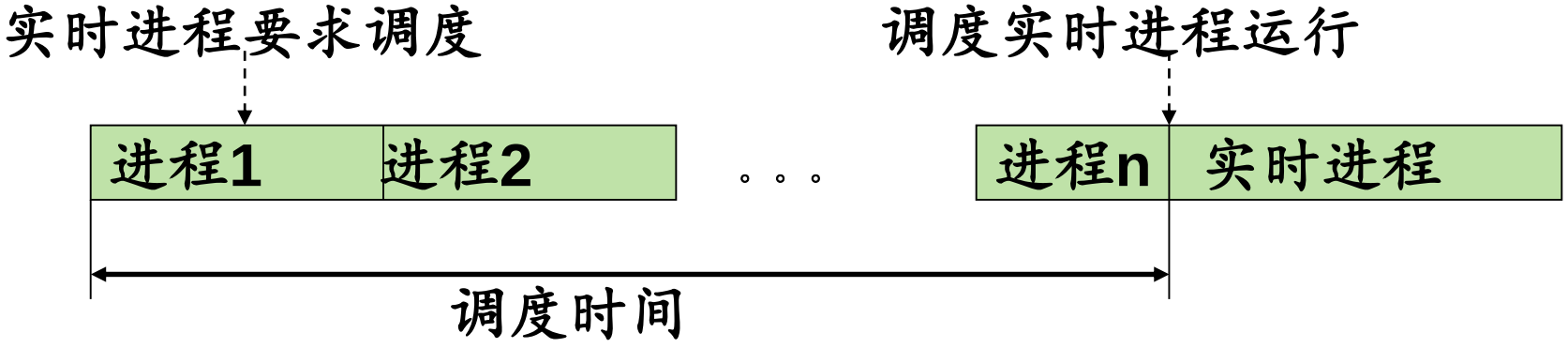
◆ 1、非抢占式调度算法

该算法较简单，用于一些小型实时系统或要求不太严格的实时系统中。又可分为两种：

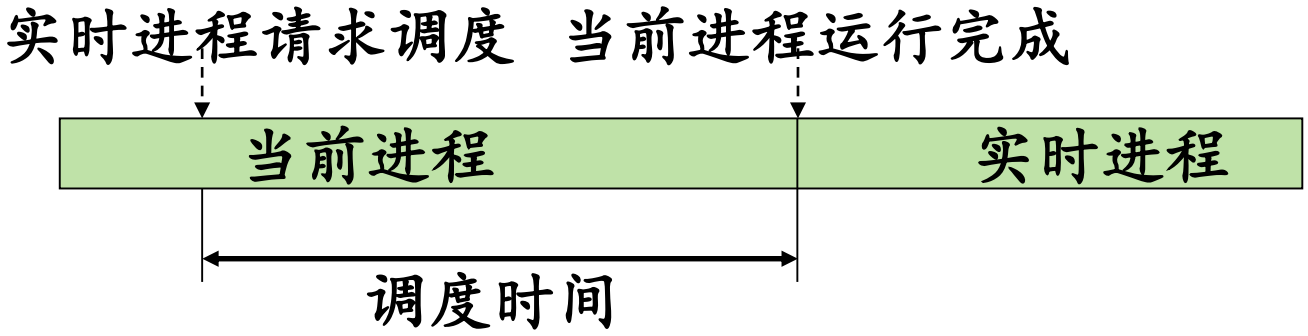
- 非抢占式轮转调度算法。常用于工业生产的群控系统中，要求不太严格（响应时间约为数秒）
- 非抢占式优先调度算法。要求较为严格，根据任务的优先级安排等待位置。可用于有一定要求的实时控制系统中。（响应时间约为数百ms）

3.3.2 实时调度算法的分类

◆ 1、非抢占式调度算法



1） 非抢占式轮转调度图示



2） 非抢占式优先权调度图示

3.3.2 实时调度算法的分类

◆ 2、抢占式调度算法

用于要求较严格的实时系统中，（t约为数十毫秒），采用抢占式优先权调度算法。根据抢占发生时间的不同可分为两种：

- 基于时钟中断的抢占式优先权调度算法

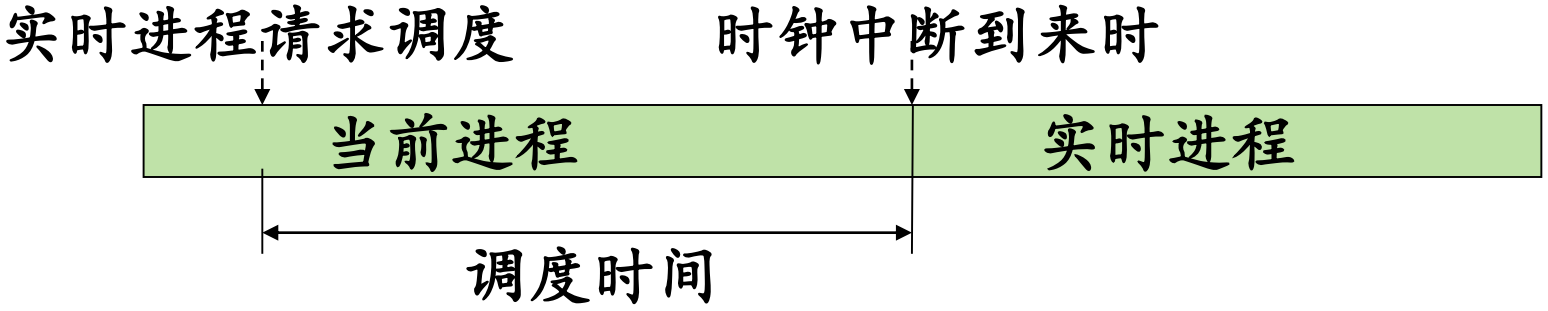
某高优先级任务到达后并不立即抢占，而等下一个时钟中断时抢占。（延时几十毫秒到几毫秒）

- 立即抢占的优先权调度算法

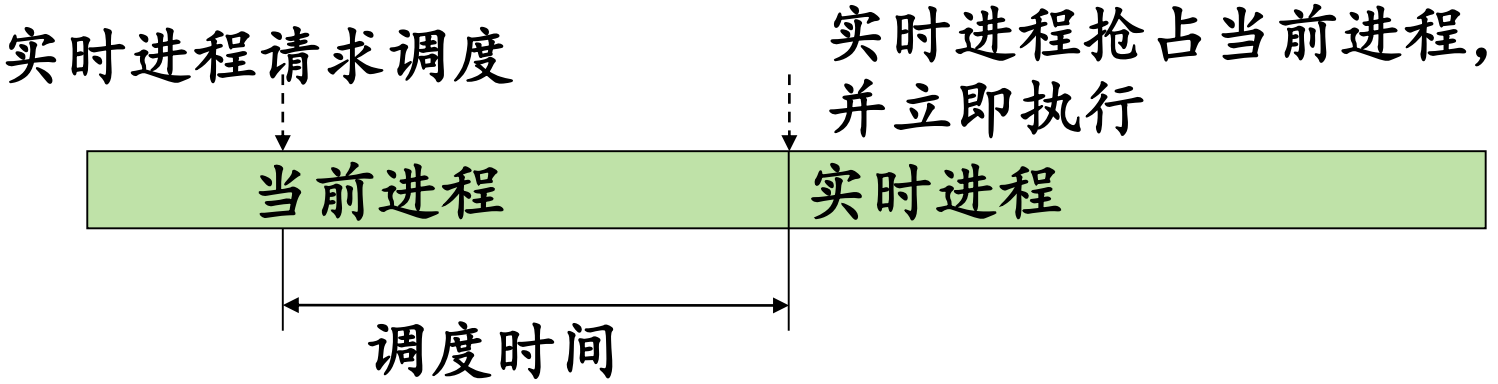
一旦出现外部中断，只要当前任务未处于临界区，就立即抢占处理机。（延时几毫秒到100微秒）

3.3.2 实时调度算法的分类

◆ 2、抢占式调度算法



1） 基于时钟中断抢占的优先权调度图示



2） 立即抢占的优先权调度图示

3.3.3 最早截止时间优先算法

◆（EDF）算法思想

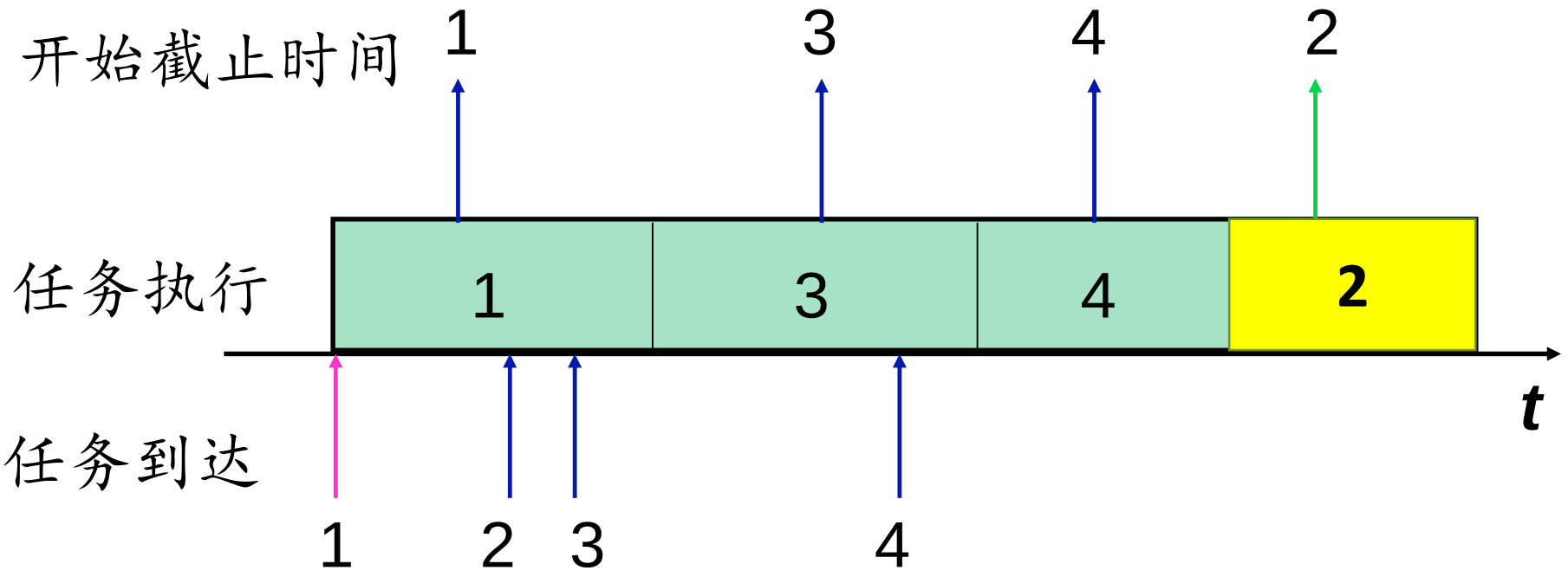
根据任务的开始或完成截止时间来确定任务的优先级，截止时间越早，优先级越高。

EDF调度方式：

- ◆非抢占式调度(通常用于非周期性实时任务)
- ◆抢占式调度(通常用于周期性实时任务)

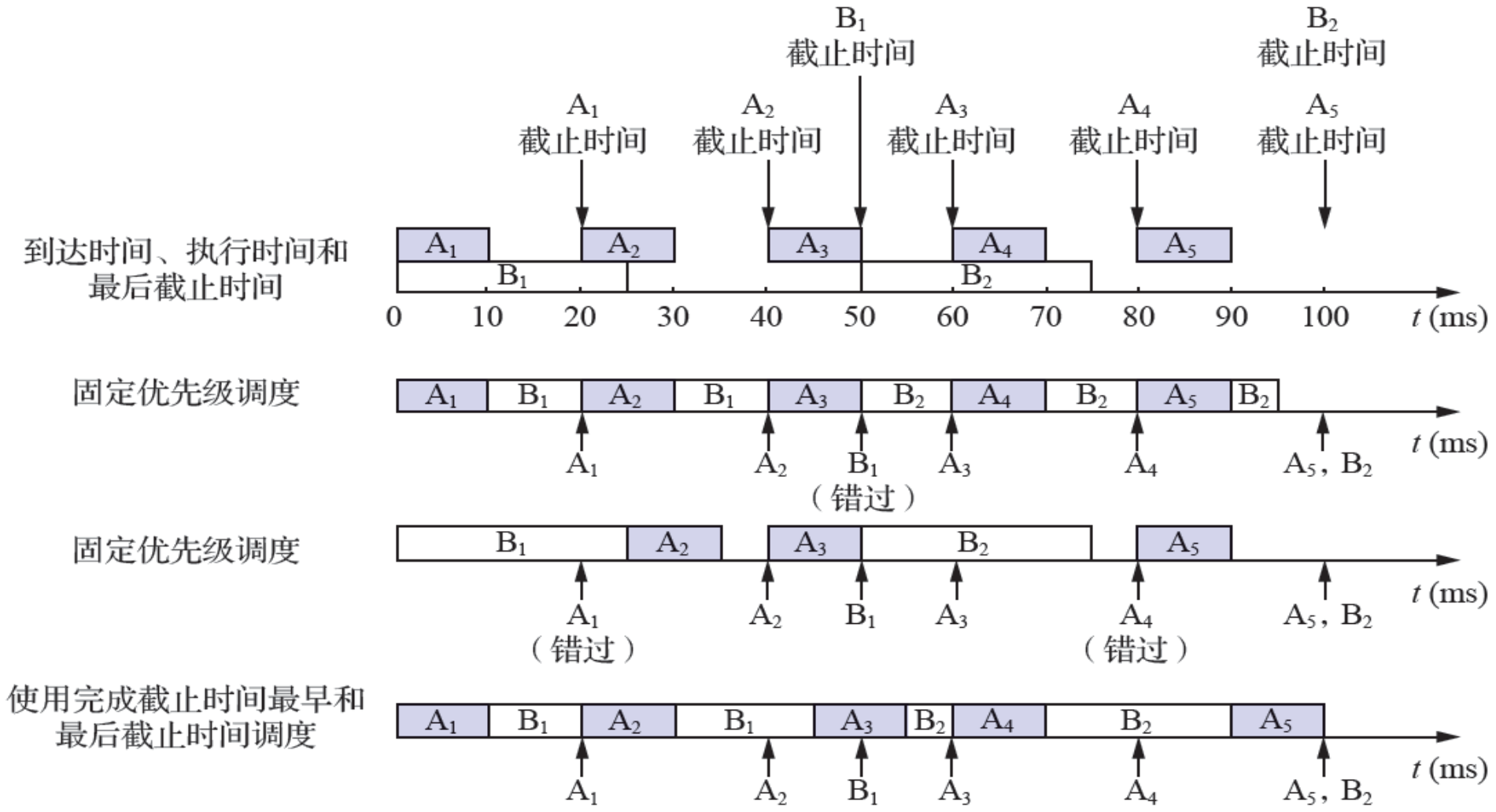
3.3.3 最早截止时间优先算法

1)EDF非抢占式调度方式用于非周期实时任务



3.3.3 最早截止时间优先算法

2）EDF抢占式调度周期实时任务



3.3.3 最早截止时间优先算法

2）EDF抢占式调度周期实时任务

在t=0时，A1和B1同时到达，由于A1的截止时间比B1早，所以调度A1执行；

在t=10时，A1完成，又调度B1执行；

在t=20时A2到达，由于A2的截止时间比B1早，故B1被中断而调度A2执行；

在t=30时，A2执行完成，又重新调度B1执行；

在t=40时A3又到达了，但B1的截止时间要比A3早，仍应让B1继续执行，直到完成（t=45）,然后再调度A3执行；

在t=55时，A3完成，又调度B2执行。

3.3.4 最低松弛度优先算法

◆最低松弛度优先LLF（Least Laxity First）算法

描述：该算法是根据任务紧急（或松弛）的程度，来确定任务的优先级。任务的紧急程度越高（松弛度越低），为之赋予的优先级就越高。

- 松弛度的计算方法：

松弛度=必须完成时间－其本身的运行时间－当前时间

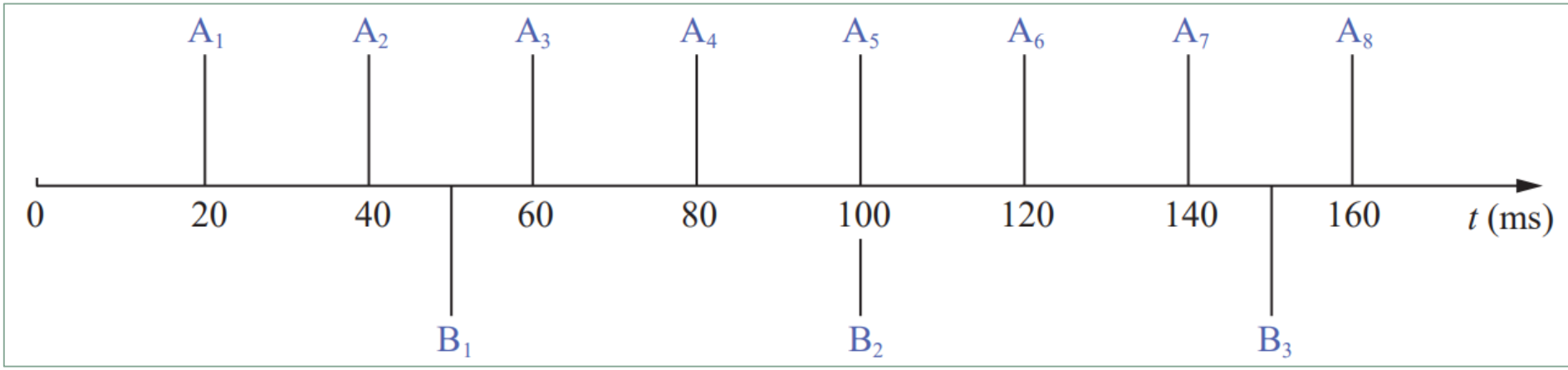
- LLF调度方式：

该算法主要用于抢占调度方式中。采用抢占式调度，当一任务的最低松弛度减为0时，它便立即抢占当前进程。

3.3.4 最低松弛度优先算法

◆最低松弛度优先LLF（Least Laxity First）算法

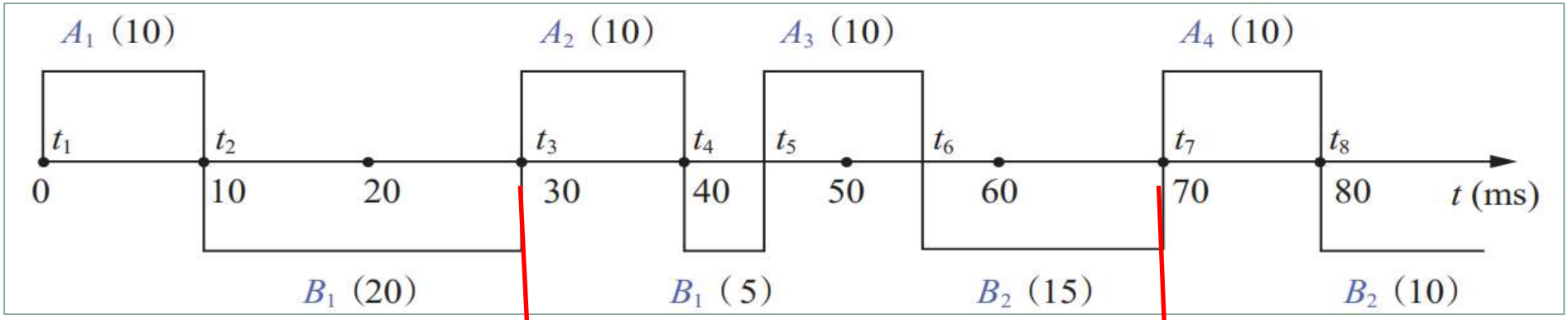
例子:两个周期性实时任务A和B，任务A要求每20ms执行一次，执行时间为10ms，任务B要求每50ms执行一次，执行时间为25ms;由此可得知A，B任务每次必须完成的时间分别为A1、A2、A3…和B1、B2、B3…



任务A和任务B每次必须完成的子任务的时间情况

3.3.4 最低松弛度优先算法

◆最低松弛度优先LLF（Least Laxity First）算法



利用LLF算法进行调度的情况

	t=0	t=10	t=30	t=40	t=45	t=55	t=70	t=80	t=90
A1	10								
A2	-	-	0						
A3	-	-	-	10	5				
A4	-	-	-	-	-	-	0		
B1	25	15	15	5	-	-			
B2	-	-	-	-	-	20	20	10	

松弛度为0
发生抢占

3.3.4 最低松弛度优先算法

◆LLF（Least Laxity First）算法

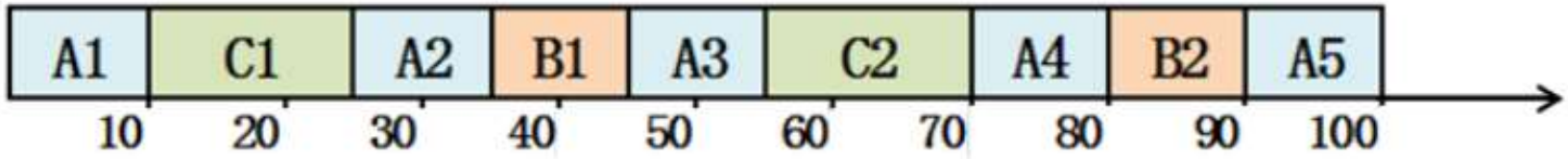
练习： 在一个实时系统中，有三个周期性实时任务 A、B和C。

任务A要求每 20ms执行一次，执行时间为 10ms；

任务B要求每 50ms执行一次，执行时间为 10ms；

任务C 要求每 50ms执行一次，执行时间为 15ms。

请按最低松弛度优先算法画出 0~100ms 的任务调度顺序。



3.3.5 优先级倒置问题

◆ 1、优先级倒置的形成

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。

案例：三个完全独立的进程 P1,P2,P3，优先级依次递减。

P1和P3通过共享一个临界资源进行交互。

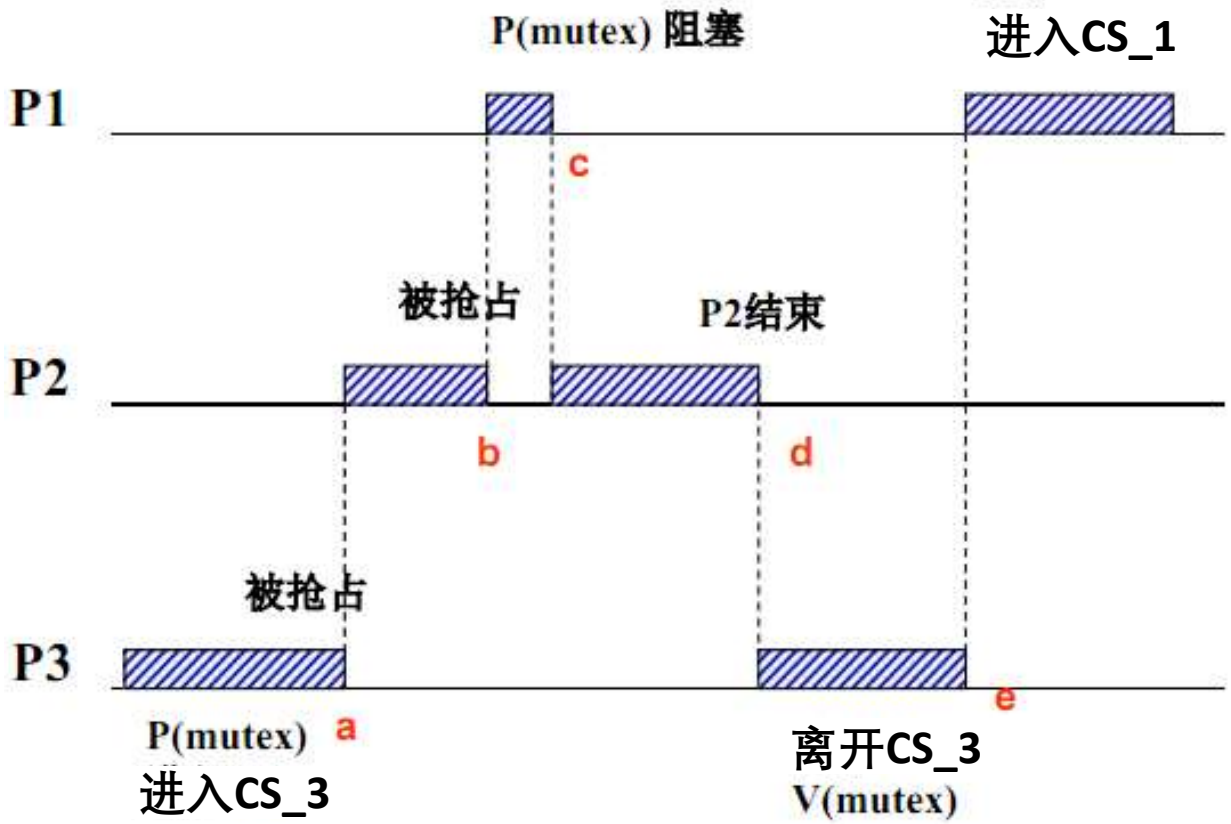
P1：… P(mutex); CS_1; V(mutex); …

P2：… program2;…

P3：… P(mutex); CS_3; V(mutex); …

3.3.5 优先级倒置问题

◆ 1、优先级倒置的形成



3.3.5 优先级倒置问题

◆ 2、优先级倒置问题解决方法

方法1：一旦一个进程进入临界区，不得对其抢占

缺点：如果临界区执行时间非常长，则效果不好。

方法2：采用动态优先级继承办法进行处理。

原理：高优先级进程阻塞后，其优先级被低优先级进程继承，从而使得低优先级进程可尽快执行完临界区，减少高优先级进程阻塞时间。

第3章 处理机调度与死锁

3.4 实例：Linux进程调度

◆Linux进程调度

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。

调度器类默认优先级：Stop_Task > Real_Time > Fair > Idle_Task

3.4 实例：Linux进程调度

◆Linux进程调度

1、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。

2、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。

处理机调度-小结

- 处理机调度的三个层次**
- 进程调度的两种方式**
- 处理机调度算法的目标
- 三种作业调度算法**

  FCFS、SJF、HRRN

- 三种进程调度算法**

  RR、FB、公平

- 两种常用实时调度算法(EDF、LLF)及优先级倒置**

本章（处理机部分）结束，
祝同学们学习愉快！