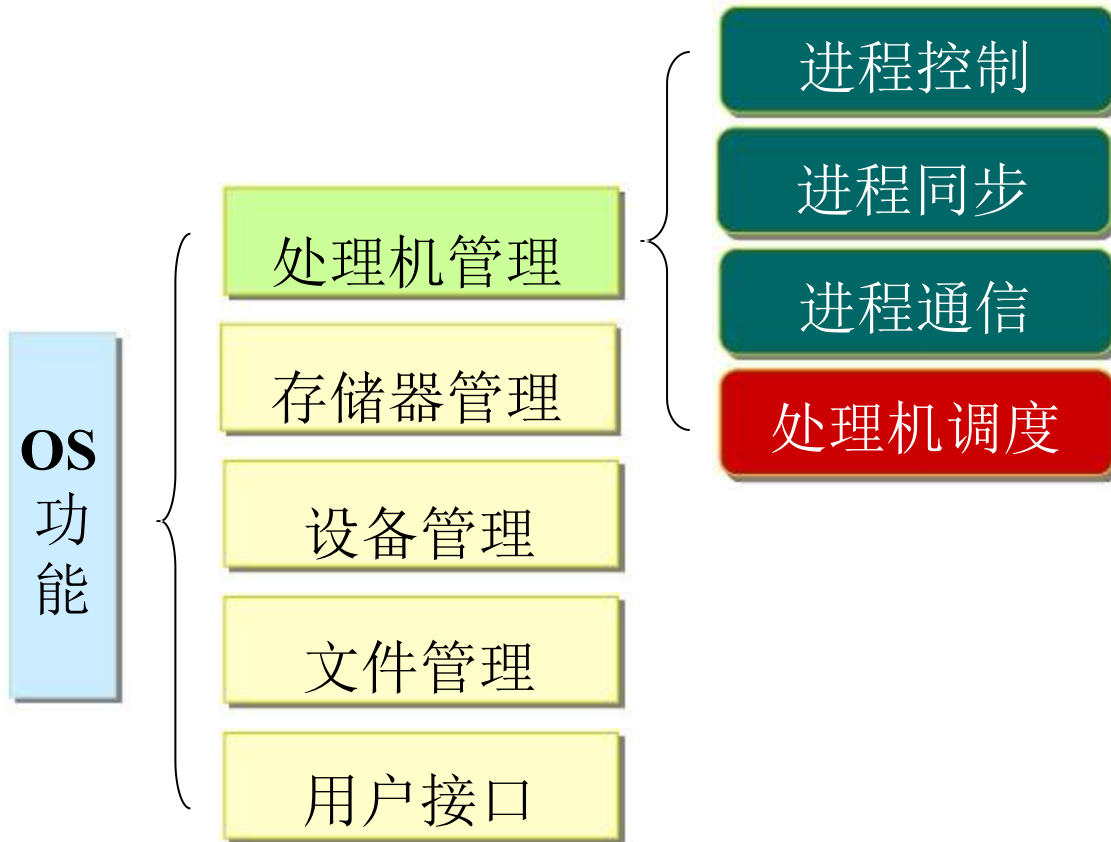




第3章 处理机调度与死锁 (6学时)



本章内容所处位置





本章主要内容

- ❖ 3.1 处理机调度的层次和调度算法的目标
- ❖ 3.2 作业与作业调度
- ❖ 3.3 进程调度
- ❖ 3.4 实时调度
- ❖ 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



调度的作用和级别

1. 调度的作用

处理机调度的主要目的就是分配处理机

▲ **调度**的功能是组织和维护就绪进程队列。包括确定调度算法、按调度算法组织和维护就绪进程队列。

▲ **分派**的功能是指当处理机空闲时，从就绪队列队首中移出一个PCB，并将该进程投入运行。

习惯上往往把上述功能统称为**进程调度**

● 调度程序还要关注CPU的利用效率



3.1 处理机调度的层次与调度算法的目标

❖ 3.1.1 处理机调度的层次

高级调度

低级调度

中级调度

❖ 3.1.2 处理机调度算法的目标

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

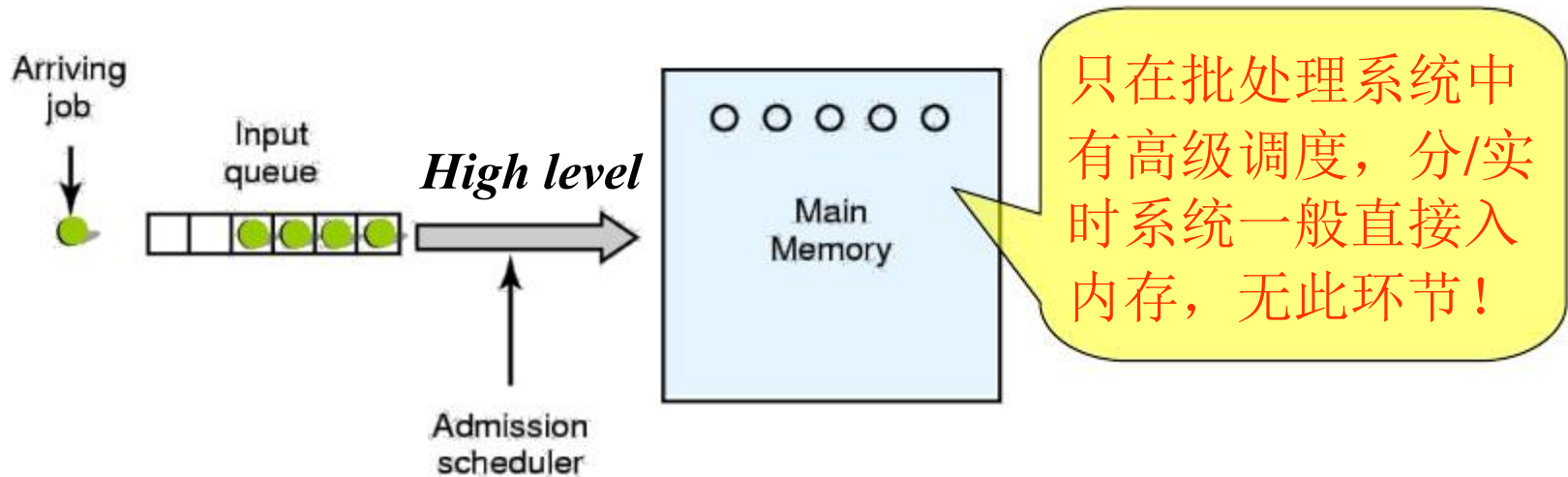




3.1.1 处理机调度的层次

❖ 1、高级调度（作业调度）

高级调度又称作业调度、长程调度、宏观调度，主要功能是根据某种算法将外存上处于后备队列中的作业调入内存，并创建进程、分配资源，然后将进程插入就绪队列，有时也称为接纳调度。

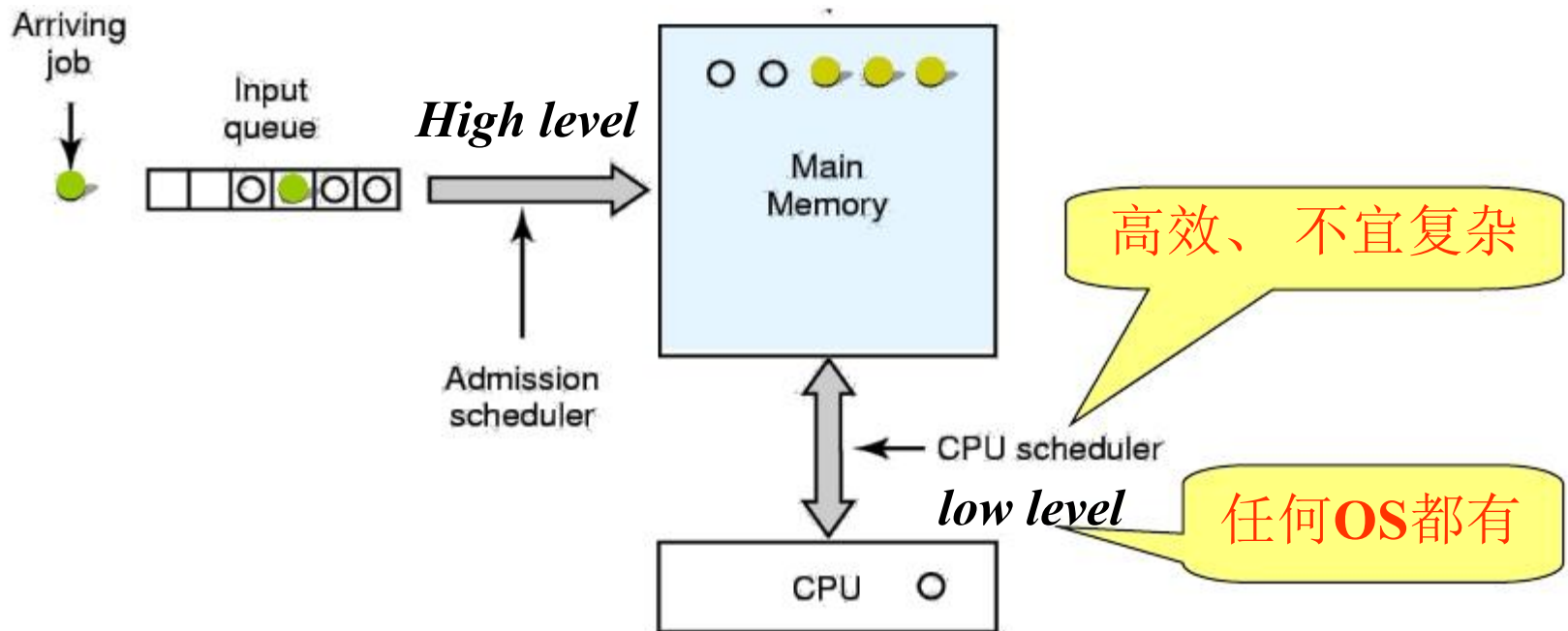




3.1.1 处理机调度的层次

❖ 2、低级调度（进程调度）

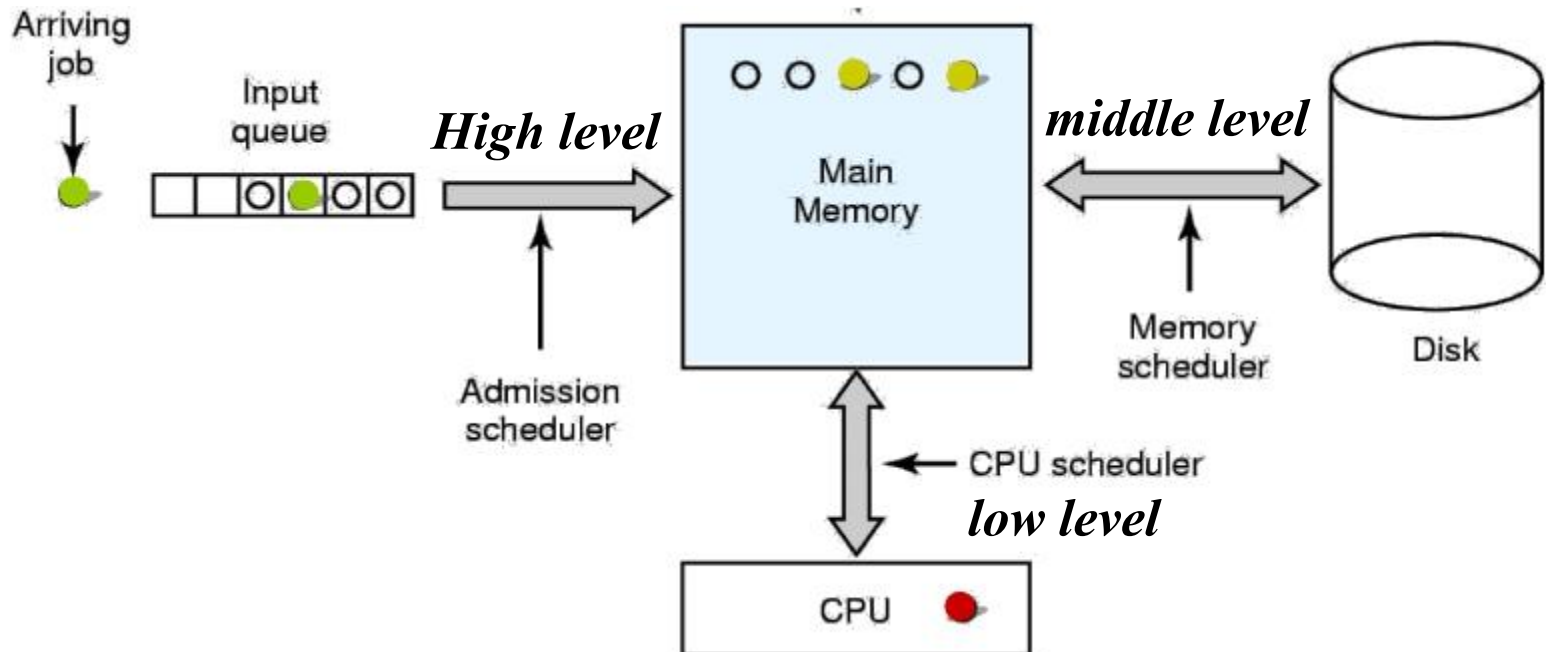
低级调度又称进程调度或短程调度，用来决定就绪队列中哪个进程应获得处理机，再由分派程序执行把处理机分配给该进程的具体操作（比如保存和设置**CPU现场**）。



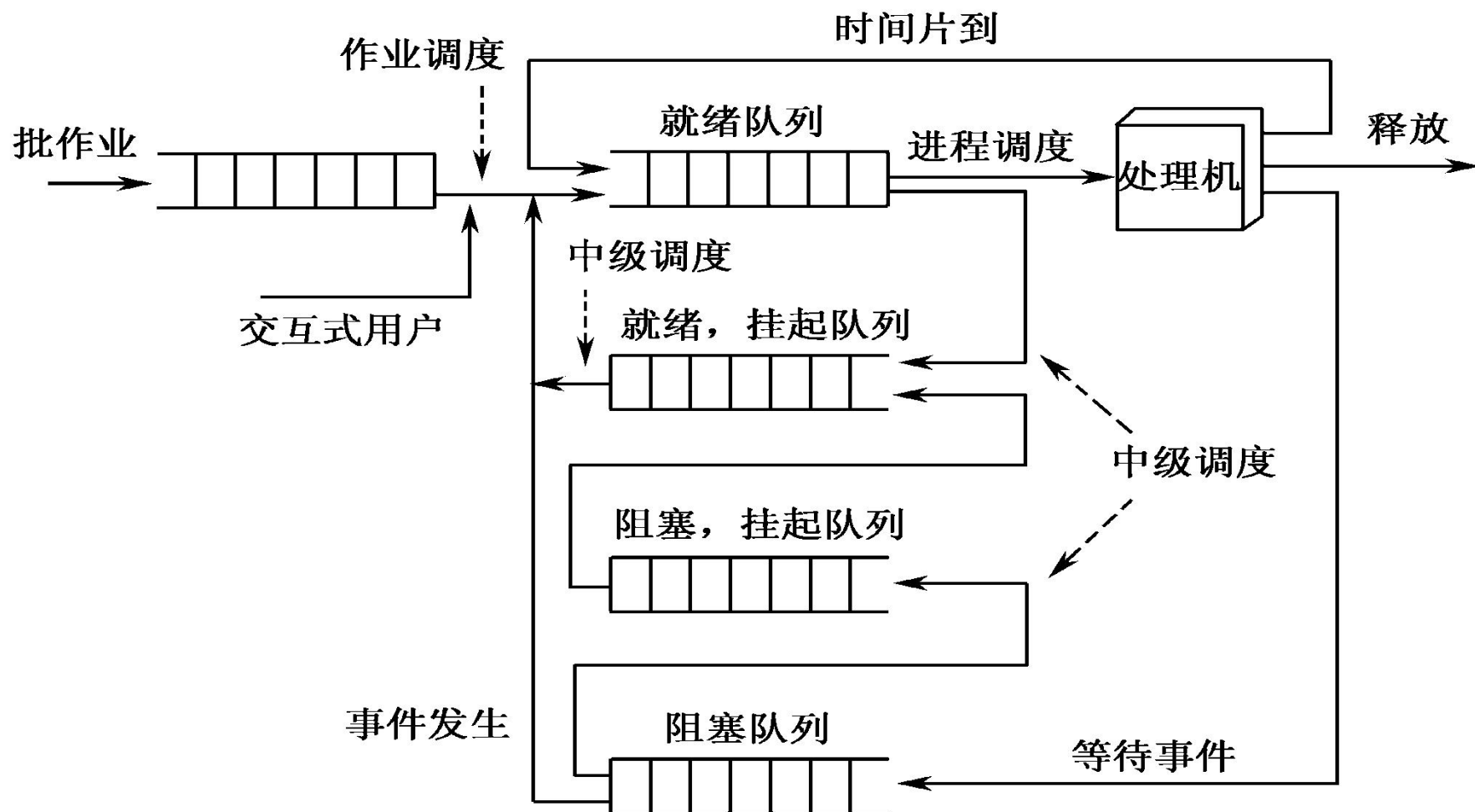
3.1.1 处理机调度的层次

❖ 3、中级调度

中级调度又称为中程调度，主要任务就是完成进程的部分或全部在内、外存间的交换，以提高内存利用率和系统吞吐量。（第四章再详细讲）



三级调度模型





三种调度小结

	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 处理机调度的层次与调度算法的目标

❖ 3.1.1 处理机调度的层次

高级调度

低级调度

中级调度

❖ 3.1.2 处理机调度算法的目标

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





3.1.2 处理机调度算法的目标

❖ 1、处理机调度的共同目标

(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：保持各类资源使用的平衡性

(4) 策略强制执行：对制订的策略，必须予以准确执行，即使会造成某些工作的延迟也要执行



3.1.2 处理机调度算法的目标

❖ 2、批处理系统的目标

(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时间：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。



3.1.2 处理机调度算法的目标

指标

平均周转时间

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

平均带权周转时间

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

其中： T_i 为第*i*个作业的周转时间， T_{si} 为系统实际给第*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。



3.1.2 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



3.1.2 处理机调度算法的目标

❖ 3、分时系统的目标

(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。



3.1.2 处理机调度算法的目标

❖ 4、实时系统的目标

(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。



本章主要内容

- ❖ 3.1 处理机调度的层次和调度算法的目标
- ❖ 3.2 作业与作业调度
- ❖ 3.3 进程调度
- ❖ 3.4 实时调度
- ❖ 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





3.2.1 批处理系统中的作业

❖ 1、作业、作业步和作业流

作业(Job): 由程序 + 数据 + 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。

在批处理系统中，是以作业为基本单位从外存调入内存的。

作业步(Job Step): 是指在作业运行期间，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 “连结装配”作业步；③ “运行”作业步。

作业流(Job Stream): 是将相关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。



3.2.1 批处理系统中的作业

❖ 2、作业控制块**JCB (Job Control Block)**

JCB是作业在系统中存在的标志，其中保存了批处理系统对作业进行管理和调度所需的全部信息。

JCB内容

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

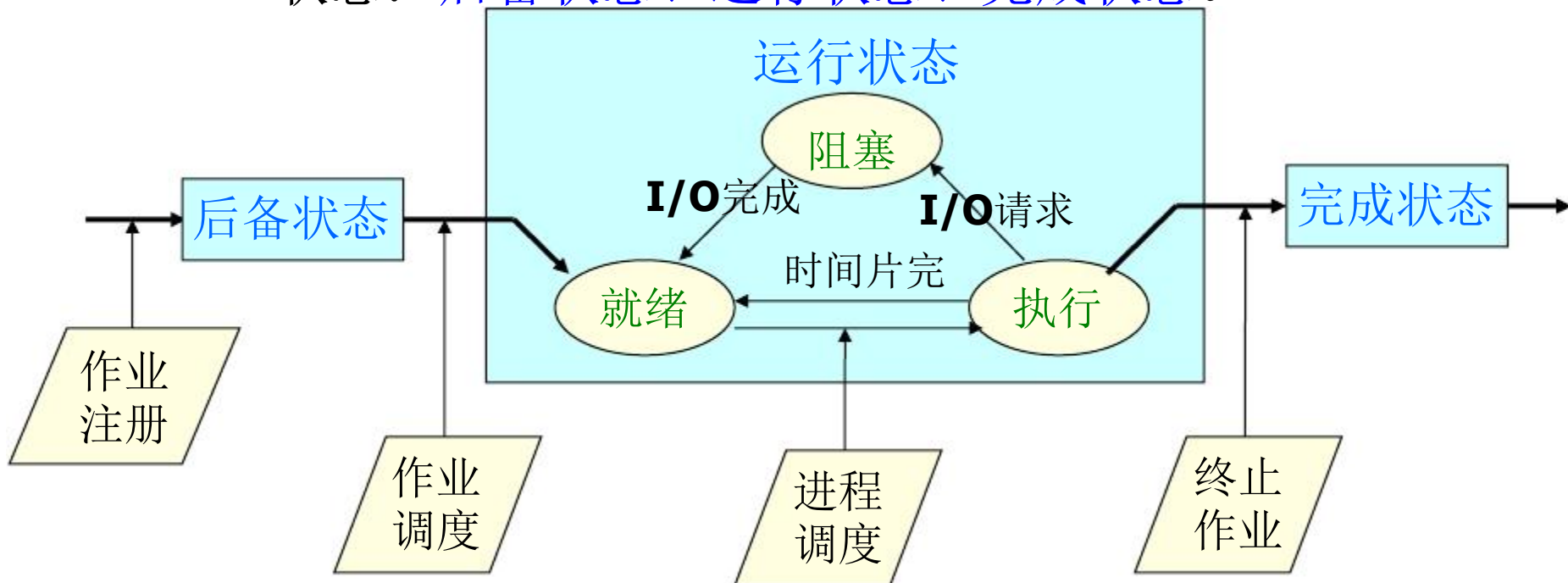
进入系统→建立**JCB** →插入相应后备队列→**作业调度**→作业运行→作业结束→回收资源→撤销**JCB** 。



3.2.1 批处理系统中的作业

❖ 3、作业运行的三个阶段与三种状态

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。





3.2.2 作业调度的主要任务

❖ 作业调度

主要功能

将外存后备队列中的作业调入内存，创建进程及**PCB**等，插入就绪队列。

调度决策

决定接纳作业数量（取决于多道程序度）

太多：有利于提高**CPU**的利用率与系统吞吐量，
但周转时间太长

太少：系统效率低

决定接纳哪些作业（取决于调度算法）

先来先服务、短作业优先、基于作业优先级等



3.2.3 作业调度算法

调度算法是指根据系统的资源分配策略所规定的资源分配算法。对于不同的系统和调度目标，通常采用不同的调度算法。

- ❖ 1、先来先服务调度算法
- ❖ 2、短作业（进程）优先调度算法
- ❖ 3、优先级调度算法

高响应比优先调度算法



上述所有算法也可用于进程调度



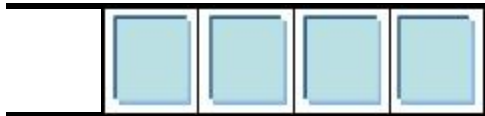
3.2.3 作业调度算法

❖ 1、先来先服务（FCFS）调度算法

应用范围与含义

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



3.2.3 作业调度算法

FCFS应用示例

作业名	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



3.2.3 作业调度算法

FCFS的优缺点

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。

概念

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。

I/O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。

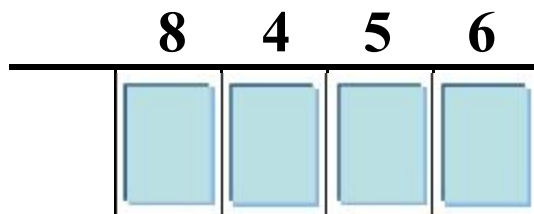


3.2.3 作业调度算法

❖2、短作业/进程优先（SJF/SPF）调度算法 应用范围与含义

作业调度：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。

进程调度：从就绪队列中选择估计运行时间最短的进程，为之分配处理机，使之投入运行。





3.2.3 作业调度算法

SJF/SPF应用示例

调度 算法	作业名	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

带权周转时间明显变短



3.2.3 作业调度算法

SJF/SPF的优点

能有效地降低作业的平均等待时间，提高系统吞吐量。

SJF/SPF的缺点

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/进程优先。

对长作业不利。

人机无法实现交互。

未考虑作业紧迫程度（**FCFS**也存在这个缺点）。



3.2.3 作业调度算法

❖3、优先级调度算法

应用范围与含义

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。

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 作业调度算法

高响应比优先（**HRRN**）调度算法

英文全称：**Highest Response Ratio Next**

优先级计算方式

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

 $= \frac{\text{周转时间}}{\text{要求服务时间}}$

$$= \frac{\text{响应时间}}{\text{要求服务时间}}$$

响应比 R_P



3.2.3 作业调度算法

进一步理解HRRN调度算法

$$R_P = 1 + \frac{\text{等待时间}}{\text{要求服务时间}}$$



长短兼顾

(1) 如作业等待时间相同，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。

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) 如处理时间相同，等待时间越长，响应比越高，即此时是按先来先服务来实现调度的。

缺点：每次调度之前都需要进行响应比的计算，**开销大**。



3.2.3 作业调度算法

例：FCFS/SJF/HPF调度四个作业

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





3.2.3 作业调度算法

1> FCFS算法调度结果

作业	进入时间	估计运行时间 (分钟)	开始时间	结束时间	周转时间 (分钟)	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 作业调度算法

2> SJF算法调度结果

作业	进入时间	估计运行时间 (分钟)	开始时间	结束时间	周转时间 (分钟)	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
作业平均周转时间 $T=95$ 作业带权平均周转时间 $W=3.25$					380	13

JOB1 >> JOB3 >> JOB4 >> JOB2



3.2.3 作业调度算法

3> HPF算法调度结果

作业	进入时间	估计运行 时间 (分钟)	开始时间	结束时间	周转时间 (分钟)	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
作业平均周转时间 $T = 102.5$ 作业带权平均周转时间 $W = 3.775$					410	15.1

JOB1 >> JOB3 >> JOB2 >> JOB4



本章主要内容

- ❖ 3.1 处理机调度的层次和调度算法的目标
- ❖ 3.2 作业与作业调度
- ❖ 3.3 进程调度
- ❖ 3.4 实时调度
- ❖ 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

对OS性能影响最大
的一种调度方式





3.3.1 进程调度的任务、机制和方式

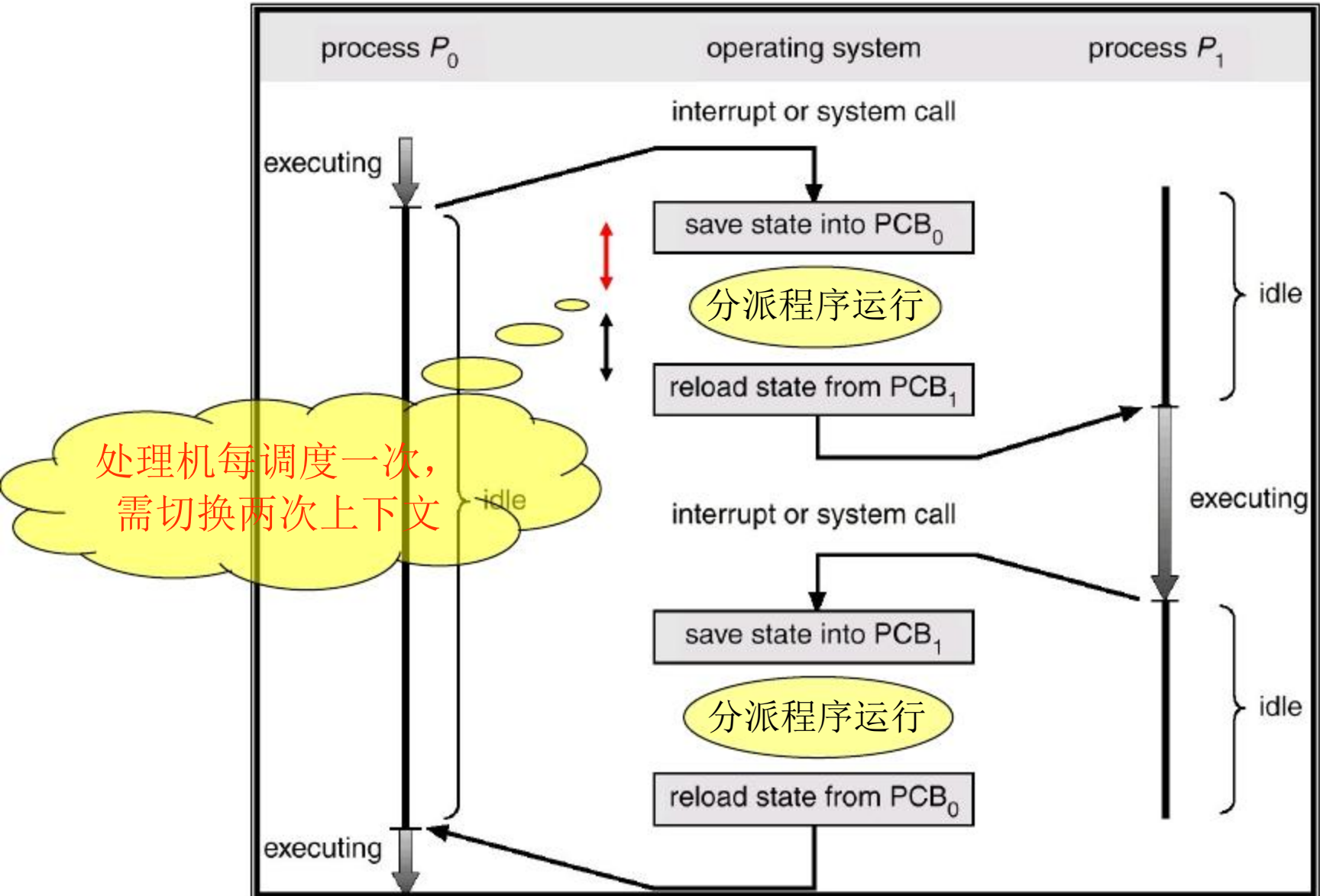
❖ 1、进程调度的任务

- 1> 保存处理机现场信息
- 2> 按某种算法选取进程
- 3> 把处理机分配给进程

❖ 2、进程调度的三个基本机制

- 1> 排队机制：对就绪进程排队
- 2> 分派机制：分派程序为选定进程分配**CPU**
- 3> 上下文切换（运行环境切换）机制：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

上下文切换机制示意图





3.3.1 进程调度的任务、机制和方式

❖ 3、进程调度方式

1> 非抢占式(Nonpreemptive Mode)

概念

非抢占式是指一旦把处理机分配给某进程，便让它一直运行下去，不允许某进程抢占已经分配出去的处理机，直至该进程完成自愿释放处理机，或发生某事件而被阻塞时，才再把处理机分配给其他进程。多用于批处理系统，不能用于分时和多数实时系统。

优缺点

简单、系统开销小，实时性差

进程调度的时机

进程正常终止或异常终止

进程因某种原因阻塞：**I/O**请求；**P**操作等



3.3.1 进程调度的任务、机制和方式

2> 抢占式(Preemptive Mode)

概念

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。

优缺点

公平、实时性好，系统开销大

抢占调度原则

优先权原则

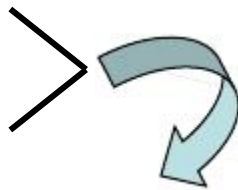
短进程优先原则

时间片原则



3.3.2 进程调度算法

- ❖ 1、时间片轮转调度算法
- ❖ 2、优先级调度算法
- ❖ 3、多队列调度算法
- ❖ 4、多级反馈队列调度算法
- ❖ 5、基于公平原则的调度算法





3.3.2 进程调度算法

❖ 1、时间片轮转 (RR) 调度算法

早期的分时
系统使用

英文全称: **Round Robin**

基本原理

把**CPU**划分成若干时间片;

按顺序将时间片分配给**FIFO**就绪队列中的每一个进程;

时间片用完时 (时钟中断请求), 即使进程未执行完毕, 系统也剥夺该进程的**CPU**, 将该进程排在就绪队列末尾, 同时系统选择队首进程运行。

时间片大小的确定

太大: 退化为**FCFS**; 无法满足短作业和交互式用户的需求

太小: 系统开销过大

较为可取的大小: 略大于一次典型交互所需要的时间



3.3.2 进程调度算法

❖ 2、优先级调度算法

❖ 优先级调度算法的类型

非抢占式优先权算法

一旦将处理机分配给就绪队列中优先级最高的进程后，该进程将一直接运行，直至完成或主动放弃**CPU**。

抢占式优先权算法

一旦有新的优先级更高的进程到来，将对当前进程的**CPU**抢占。常用于对实时性要求较高的系统中。



3.3.2 进程调度算法

❖ 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



3.3.2 进程调度算法

❖ 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。

优缺点

简单易行，系统开销小；

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。

例如：高响应比优先调度算法的优先级就是动态的。



3.3.2 进程调度算法

❖ 3、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。

多就绪队列调度算法

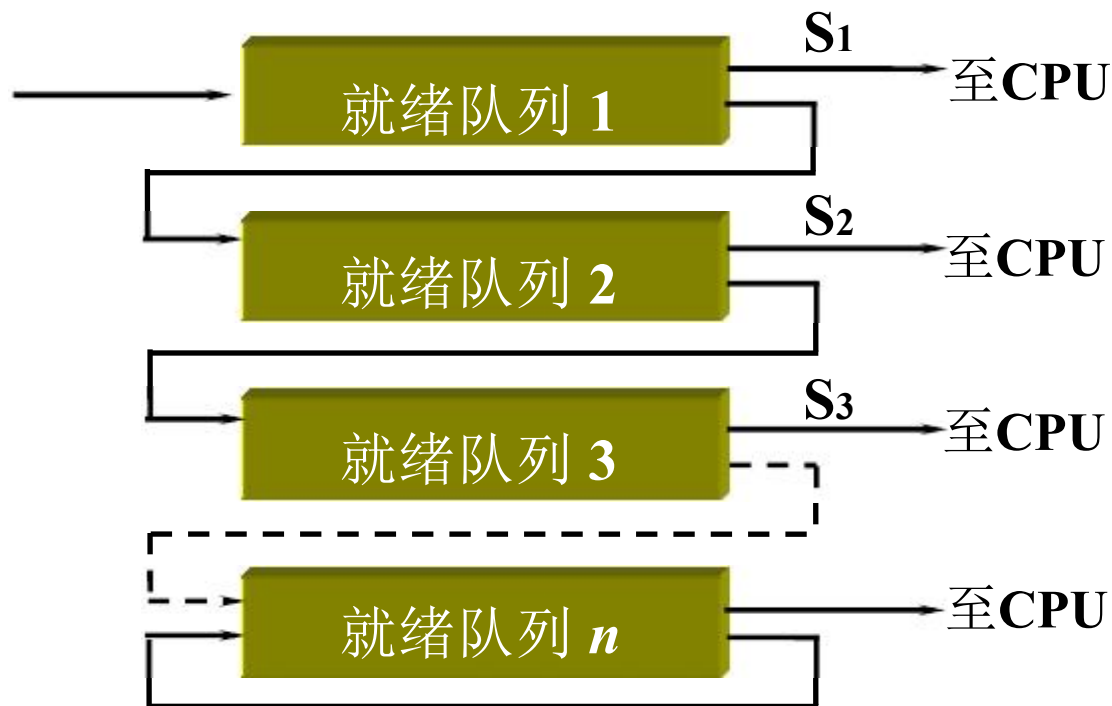
就是将就绪队列按进程的类型或性质拆分成为若干个，不同就绪队列可采用不同的调度算法，同一就绪队列的进程还可设置不同优先级，不同就绪队列也可设置不同的优先级。

OS可根据进程所处队列安排相应的调度算法，满足不同用户的需求。



3.3.2 进程调度算法

❖4、多级反馈队列（FB）调度算法

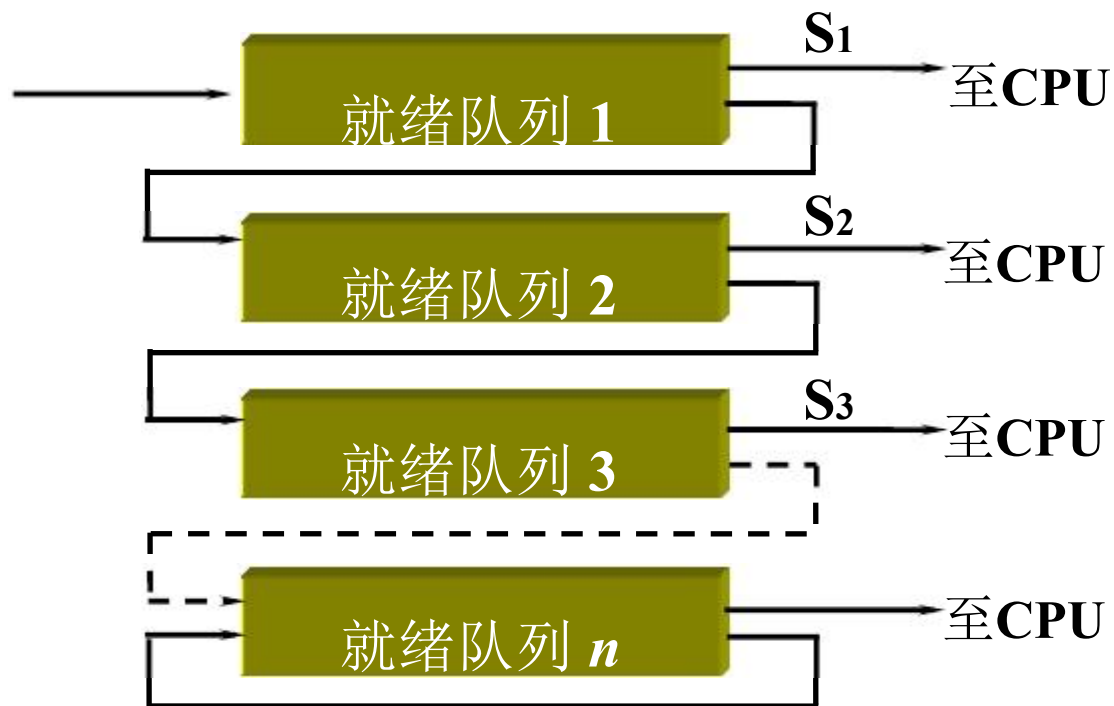


(时间片: $S_1 < S_2 < S_3$)

1> 分级: 将就绪队列分为N级, 每个就绪队列分配给不同的时间片, 队列级别越高, 时间片越短, 级别越低, 时间片越长。



3.3.2 进程调度算法

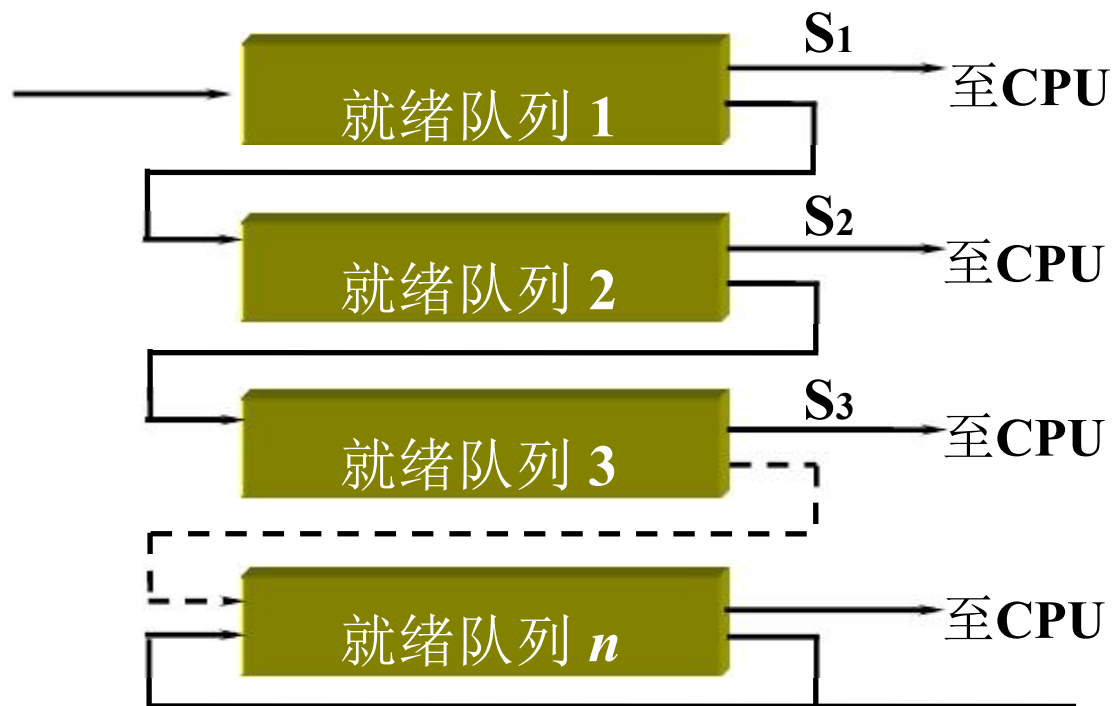


(时间片: $S_1 < S_2 < S_3$)

2> 调度: 进程第一次就绪时, 放入第一级队列队尾, 按FCFS原则等待调度。系统从第一级队列调度, 当第一级为空时, 系统转向第二级队列,



3.3.2 进程调度算法

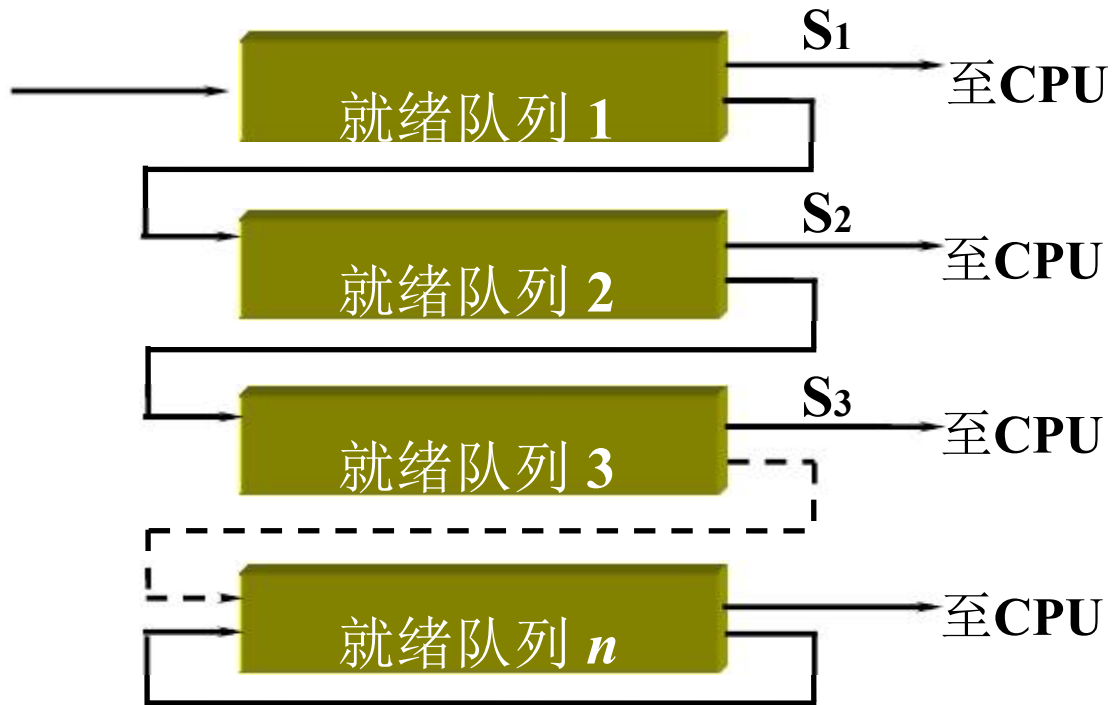


(时间片: $S_1 < S_2 < S_3$)

3> 运行: 当前进程用完一个时间片, 如运行完成, 则退出系统, 否则必须放弃CPU, 并插入下一级队列队尾; 阻塞进程被唤醒时, 进入原来的就绪队列。



3.3.2 进程调度算法



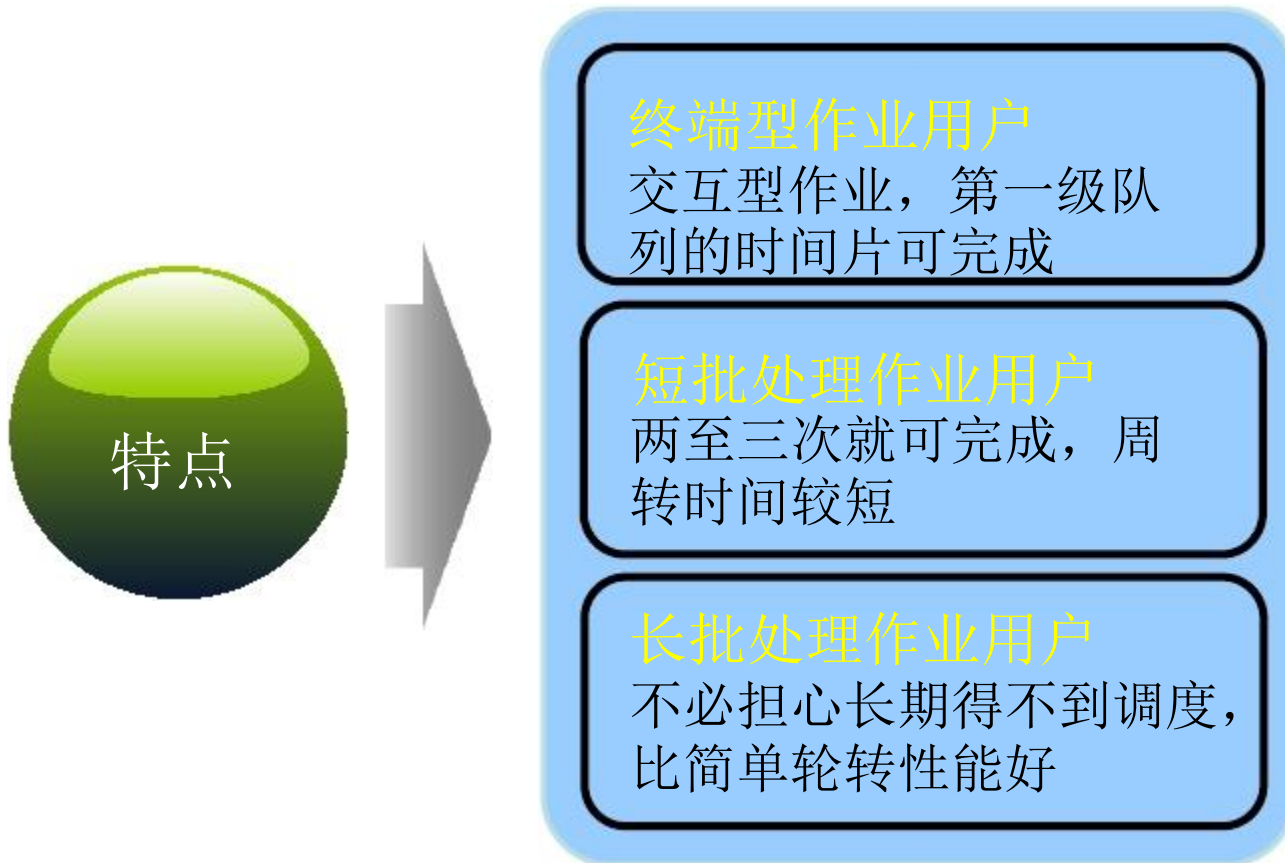
(时间片: $S_1 < S_2 < S_3$)

4> 抢占: 如果CPU正在处理第 i 级队列时, 有新进程加入第一级队列, 或者有新唤醒的进程比当前进程的队列级别高, 则新进程抢占当前进程的CPU, 而原来的当前进程插入第 i 级队列队尾。



3.3.2 进程调度算法

多级反馈队列调度算法性能





3.3.2 进程调度算法

❖ 5、基于公平原则的调度算法

5.1 保证调度算法

向用户做出的保证并不是优先运行，而是明确的性能保证，例如应保证同类型进程所获CPU时间一致
实施公平调度的算法应具有的功能

- (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：(1)/(2)
- (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



3.3.2 进程调度算法

5.2 公平分享调度算法

保证调度算法是**针对同类进程**而言的公平，如果不同用户的进程数量不一样，则变得不公平了。

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



3.3 调度算法小结

算法	选择函数	调度方式	吞吐量	响应时间	开销	对进程的影响	饿死
FCFS	max[w]	非抢占	不突出	可能很高，特别是当进程的执行时间差别很大时	小	对短进程不利；对I/O密集型进程不利	无
RR	常数	抢占	时间片太小，吞吐量会很低	为短进程提供好的响应时间	小	公平对待	无
SPF	min[s]	非抢占	高	提供好的响应时间	可能较高	对长进程不利	可能

注：w表示等待时间，s表示所需总服务时间，e表示已运行时间



3.3 调度算法小结

算法	选择函数	调度方式	吞吐量	响应时间	开销	对进程的影响	饿死
HRRN	$\max[R_p]$	非抢占	高	提供好的响应时间	可能较高	很好的满足	无
多级反馈	—	抢占	不突出	不突出	可能较高	受I/O限制的进程有利	可能



本章主要内容

- ❖ 3.1 处理机调度的层次和调度算法的目标
- ❖ 3.2 作业与作业调度
- ❖ 3.3 进程调度
- ❖ 3.4 实时调度
- ❖ 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





3.4 实时调度

- ❖ 3.4.1 实现实时调度的基本条件
- ❖ 3.4.2 实时调度算法的分类
- ❖ 3.4.3 常用实时调度算法
- ❖ 3.4.4 优先级倒置问题





3.4.1 实现实时调度的基本条件

❖ 实时系统实现实时调度的基本条件

在实时系统中，实时调度必须满足实时任务对截止时间的要求，因此实时系统应具备一定基本条件：

1> 应向调度程序提供实时任务的必要信息

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



3.4.1 实现实时调度的基本条件

2> 应具备较强的处理能力

(1) 单处理机系统必须具备的处理能力

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

其中： P_i 为周期时间， C_i 为处理时间（基于周期性实时任务）

(2) 多处理机系统必须具备的处理能力

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

其中： N 为处理机数量

例：有**6**个周期性的硬实时任务，处理时间都为**10ms**，周期时间都为**50ms**，判断实时系统是否可调度？



3.4.1 实现实时调度的基本条件

3> 广泛采用抢占调度方式

抢占式：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，实现时一般应使实
时任务较小，以便及时释放**CPU**。

4> 具有快速切换机制

具有快速响应外部中断能力

具有快速分派任务能力



3.4 实时调度

- ❖ 3.4.1 实现实时调度的基本条件
- ❖ 3.4.2 实时调度算法的分类
- ❖ 3.4.3 常用实时调度算法
- ❖ 3.4.4 优先级倒置问题





3.4.2 实时调度算法的分类

❖ 实时调度算法分类

根据调度任务性质划分

硬实时调度算法

软实时调度算法

根据调度时间顺序划分

静态调度算法

动态调度算法

根据调度集中程度划分

集中式调度算法

分布式调度算法



3.4.2 实时调度算法的分类

根据进程调度方式划分

非抢占式调度算法（小型或软实时系统适用）

非抢占式轮转调度算法 秒级响应

非抢占式优先权调度算法 数百毫秒级响应

抢占式调度算法（硬实时系统适用）

时钟中断抢占式优先权调度算法 数十毫秒级响应

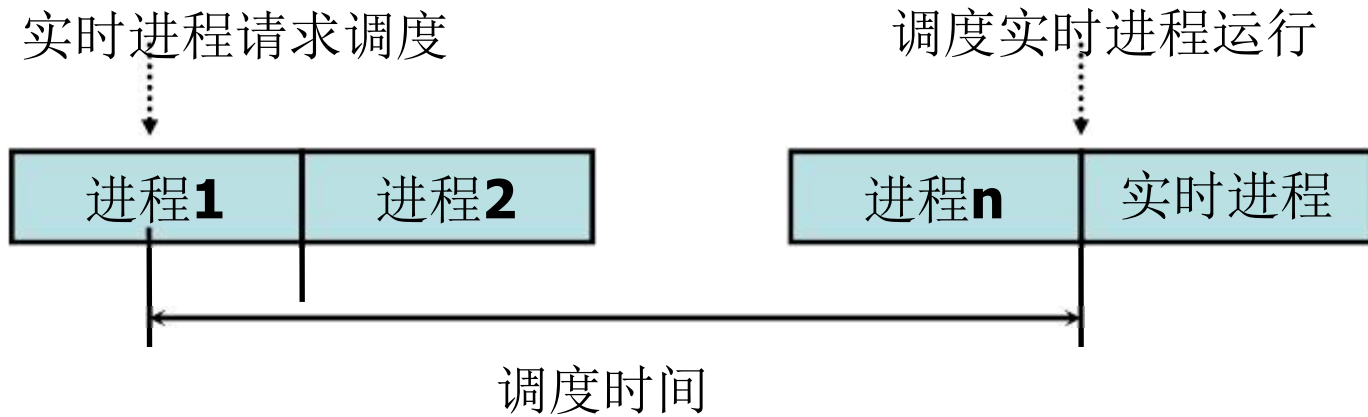
⌚ 基于时钟中断抢占点抢占

立即抢占式优先权调度算法 毫秒—微秒级

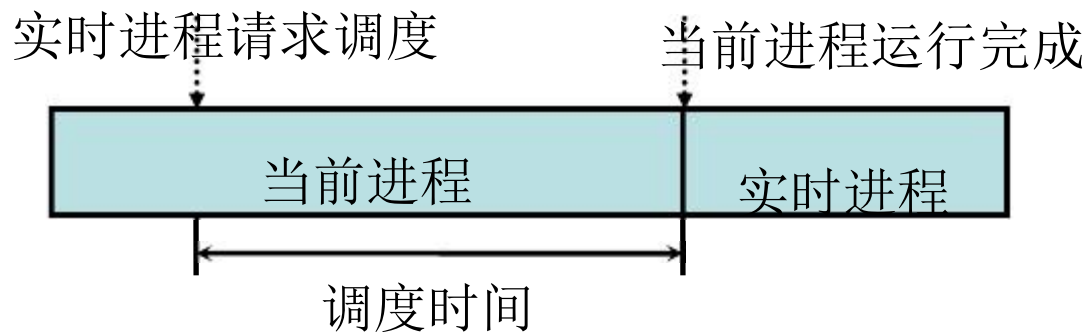
⌚ 只要不在临界区立即抢占（中断引发）



3.4.2 实时调度算法的分类



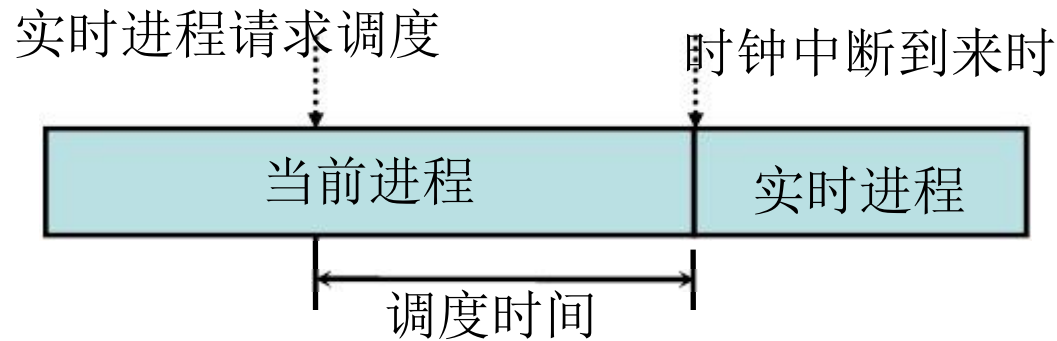
a> 非抢占式轮转调度



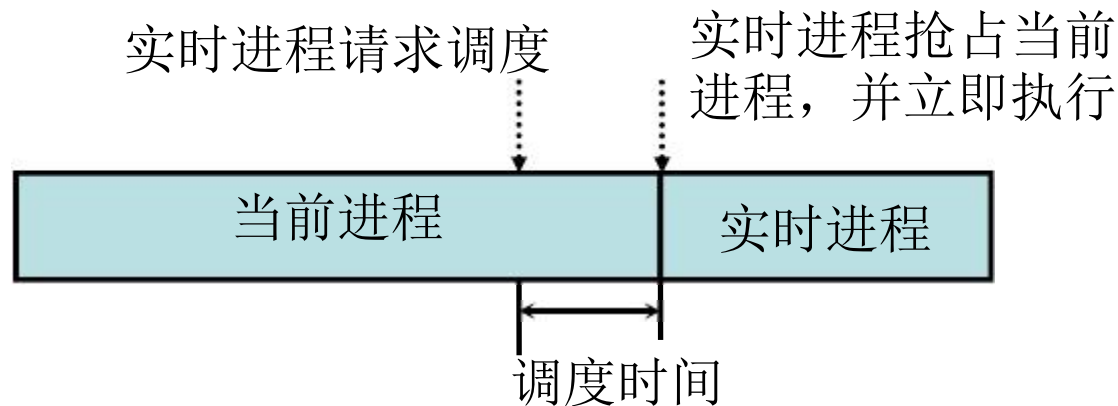
b> 非抢占式优先级调度



3.4.2 实时调度算法的分类



c> 基于时钟中断抢占的优先权调度



d> 立即抢占的优先权调度



3.4 实时调度

- ❖ 3.4.1 实现实时调度的基本条件
- ❖ 3.4.2 实时调度算法的分类
- ❖ 3.4.3 常用实时调度算法
- ❖ 3.4.4 优先级倒置问题





3.4.3 常用实时调度算法

❖1、最早截止时间优先（EDF）算法

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）



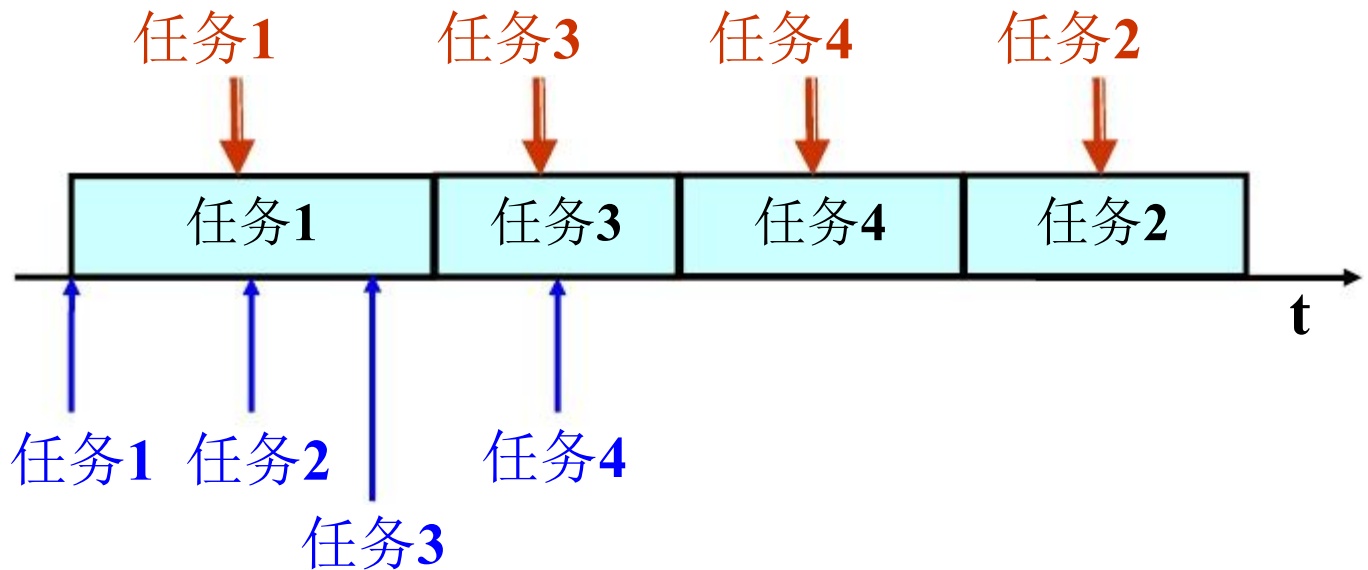
3.4.3 常用实时调度算法

例：EDF算法采用非抢占式调度实时任务

开始截止
止时间

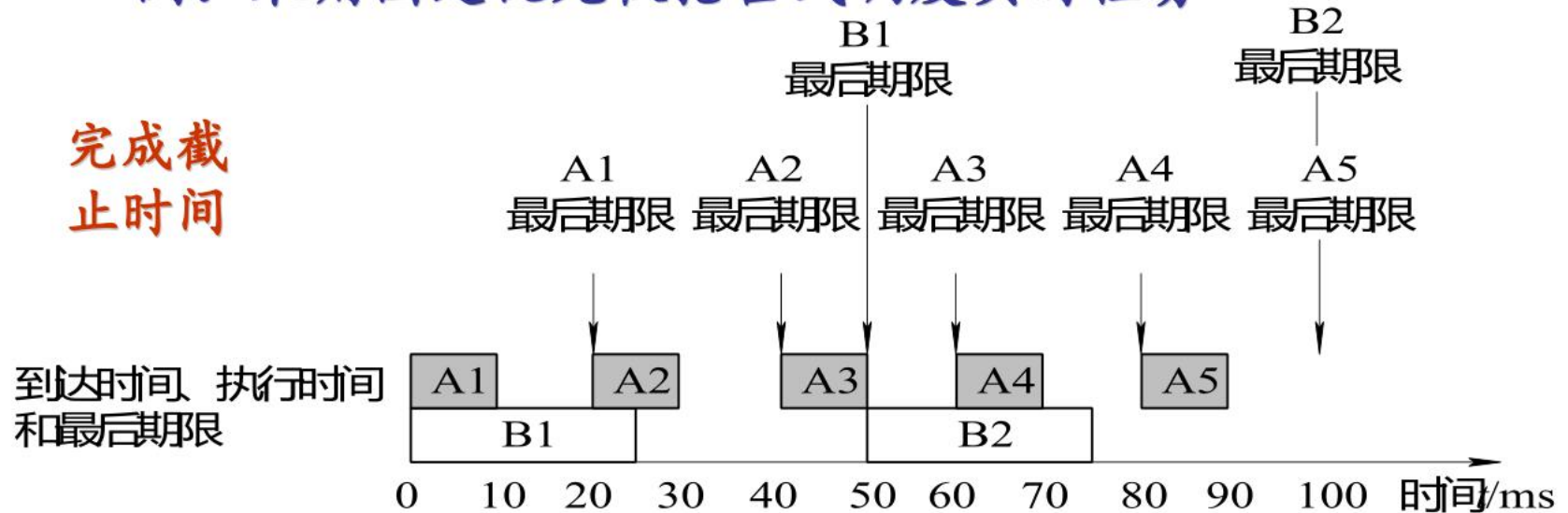
任务执行
时间

任务到
达时间

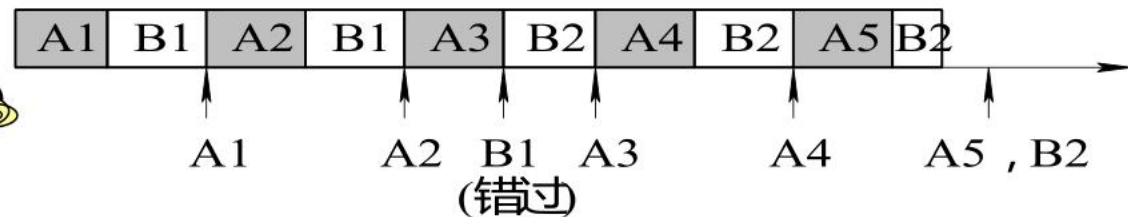


3.4.3 常用实时调度算法

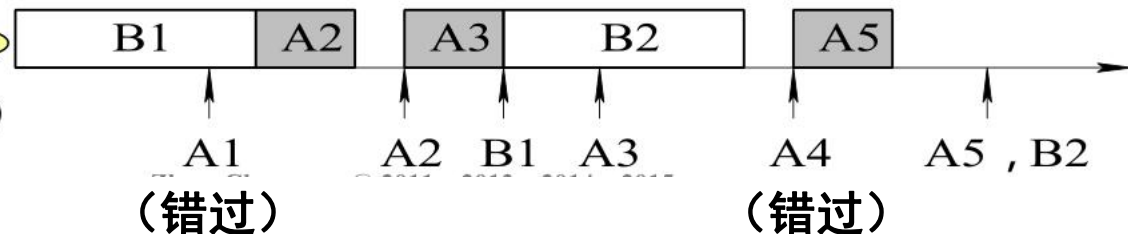
例：采用固定优先级抢占式调度实时任务



根据固定优先级调度(A优先)

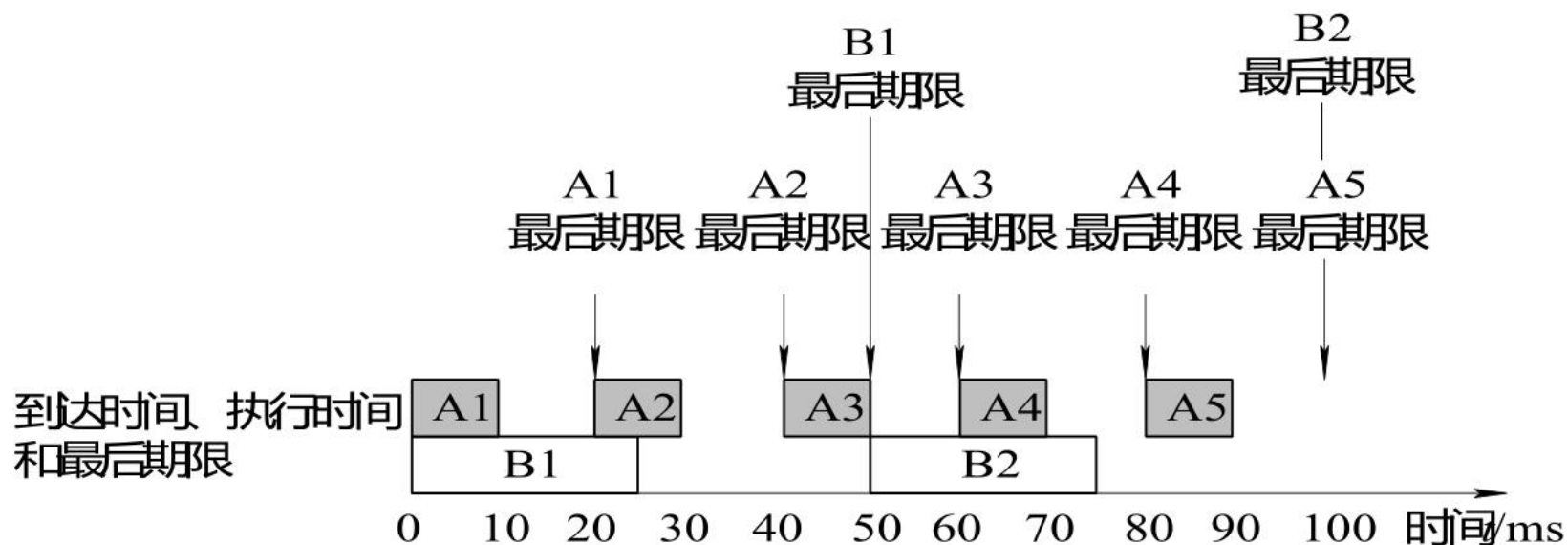


根据固定优先级调度(B优先)

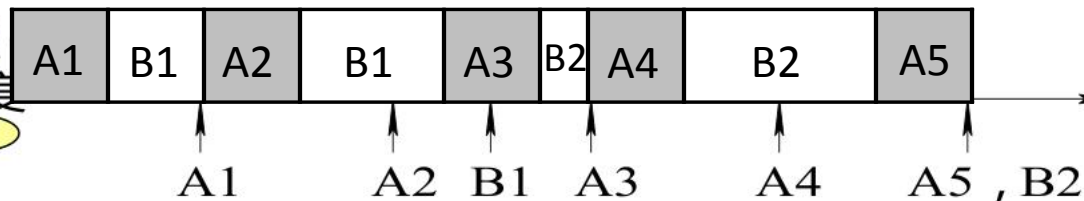


3.4.3 常用实时调度算法

例：EDF算法采用抢占式调度实时任务



根据最早完成
截止时间调度





3.4.3 常用实时调度算法

❖2、最低松弛度优先（LLF）算法

算法思想

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（松弛度越小），该任务的优先级就越高，使之优先执行。

松弛度计算

松弛度 = 截止完成时间 - 当前时间 - 剩余运行时间

LLF调度方式

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。

抢占时机：非高优先级抢占，而是松弛度为0抢占



3.4.3 常用实时调度算法

例：LLF算法调度周期性任务

假如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**20ms**执行一次，执行时间为**10ms**；任务**B**只要求每**50ms**执行一次，执行时间**25ms**。对任务**A**完成截止时间依次为**20、40、60、80、100 ...**；对任务**B**，完成截止时间依次为**50、100、150...**

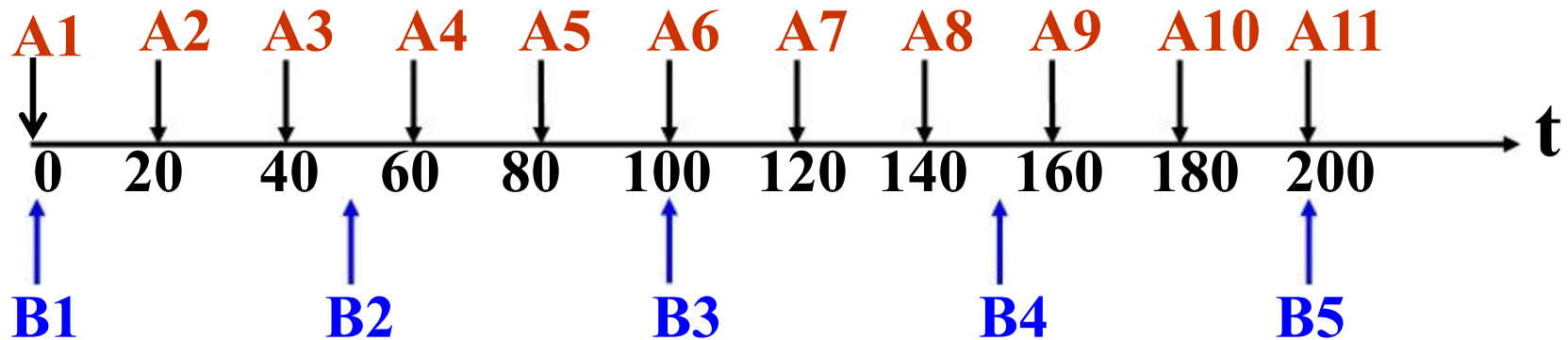


图3-11 A和B任务每次必须完成的时间点

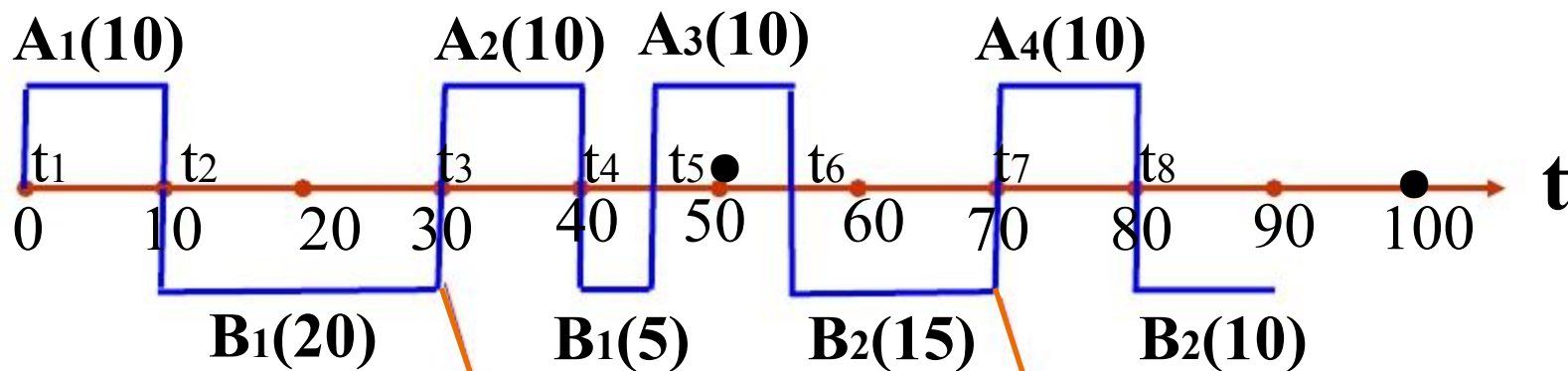
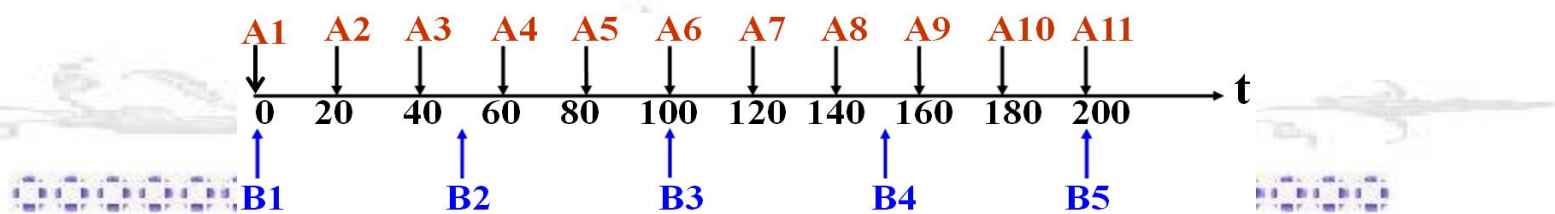


图3.12 利用LLF算法对两个周期性实时任务进行调度

	t=0	t=10	t=20	t=30	t=40	t=45	t=55	t=70	t=80	t=90
任务A ₁	10									
任务A ₂	-	-	10	0						
任务A ₃	-	-	-		10	5				
任务A ₄	-	-	-	-	-	-		0		
任务B ₁	25	15	15	15	5					
任务B ₂	-	-	-	-	-		20	20	10	



3.4 实时调度

- ❖ 3.4.1 实现实时调度的基本条件
- ❖ 3.4.2 实时调度算法的分类
- ❖ 3.4.3 常用实时调度算法
- ❖ 3.4.4 优先级倒置问题





3.4.4 优先级倒置问题

❖1、优先级倒置问题的形成

优先级倒置：指高优先级进程（或线程）被低优先级进程（或线程）延迟或阻塞。

例：已知三个进程P1、P2、P3，优先级 $P1 > P2 > P3$ ，P1与P3共享一个临界资源CS，各进程代码为：

P1 : P(mutex) CS V(mutex)

P2: ... Program2 ...

P3: P(mutex) CS V(mutex)

假设P3最先执行....



3.4.4 优先级倒置问题

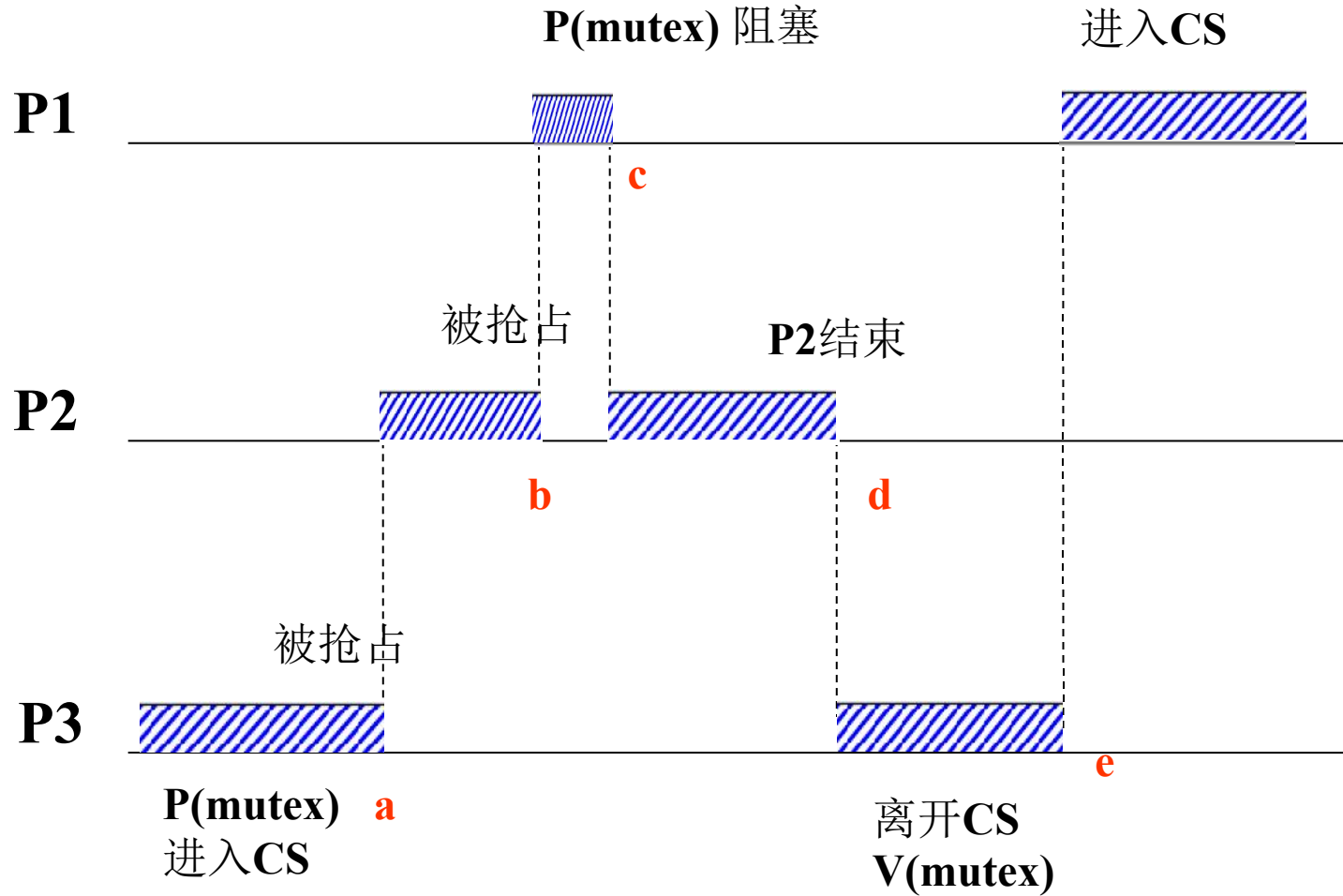


图3-10 优先级倒置示意图



3.4.4 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。



本章主要内容

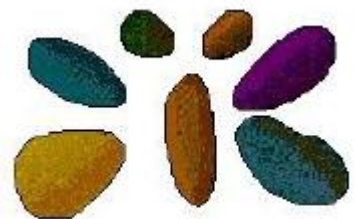
- ❖ 3.1 处理机调度的层次和调度算法的目标
- ❖ 3.2 作业与作业调度
- ❖ 3.3 进程调度
- ❖ 3.4 实时调度
- ❖ 3.5 死锁概述
- ❖ 3.6 预防死锁
- ❖ 3.7 避免死锁
- ❖ 3.8 死锁的检测与解除





3.5 死锁概述

- ❖ 3.5.1 资源问题
- ❖ 3.5.2 产生死锁的原因
- ❖ 3.5.3 产生死锁的必要条件
- ❖ 3.5.4 处理死锁的基本方法





3.5.1 资源问题

❖ 1、可重用性资源和消耗性资源

可重用性（永久性）资源

可重用性资源是指可重复的使用资源，如打印机。

性质：（1）每一类可重用性资源的**单元数目相对固定**，进程运行期间既不创建也不删除；（2）其每一个单元**只能分配给一个进程、不能共享**；（3）进程使用可重用性资源**须按顺序**：**a**、请求资源；**b**、使用资源；**c**、释放资源

可消耗性（临时性）资源

可消耗性资源是指由一个进程产生，被另一进程使用后就再也无用的资源，也称为临时性资源。

性质：（1）每一类资源的单元数目在进程运行期间可以不断变化；（2）可由进程不断创造，以增加某类资源的数量；（3）进程请求一定数量临时资源用完即弃。



3.5.1 资源问题

❖ 2、可抢占性资源和不可抢占性资源

可抢占性（可剥夺性）资源

可抢占性资源是指进程在获得这类资源后，该资源可再被其他进程或系统抢占，如**处理机**、**内存**等。

这类资源是不会引起死锁的。

不可抢占性（非剥夺性）资源

不可抢占性资源是指当系统把这类资源分配给某个进程后，再不能强行收回，只能在进程用完后自行释放，如**刻录机**、**磁带机**、**打印机**等。



3.5 死锁概述

- ❖ 3.5.1 资源问题
- ❖ 3.5.2 产生死锁的原因
- ❖ 3.5.3 产生死锁的必要条件
- ❖ 3.5.4 处理死锁的基本方法



3.5.2 产生死锁的原因

❖ 1、竞争不可抢占性资源可能引起死锁

系统中的不可抢占性资源由于数量有限而不能满足进程运行的需要，由于这类资源具有不可抢占性，进程在争夺这类资源时可能陷入死锁。

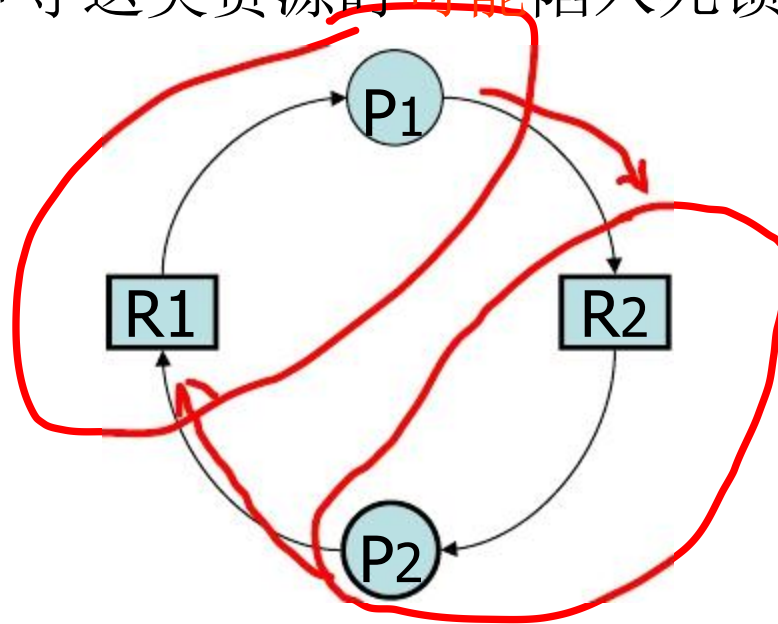
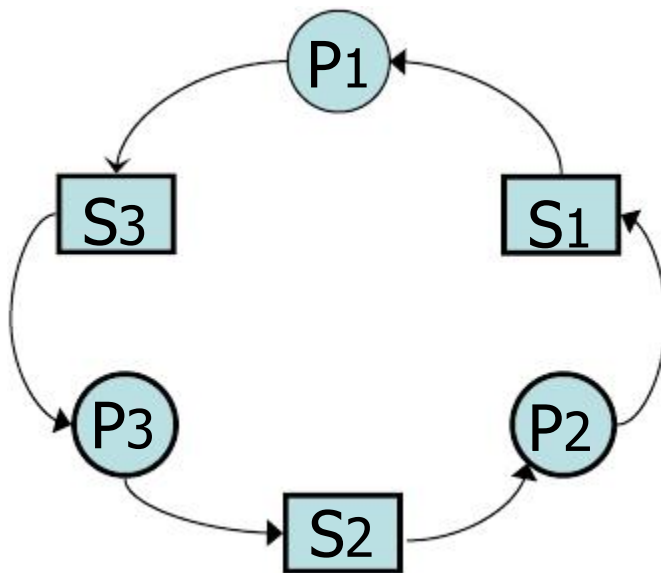


图3-13 I/O设备共享时的死锁情况

3.5.2 产生死锁的原因

❖ 2、竞争可消耗性资源可能引起死锁



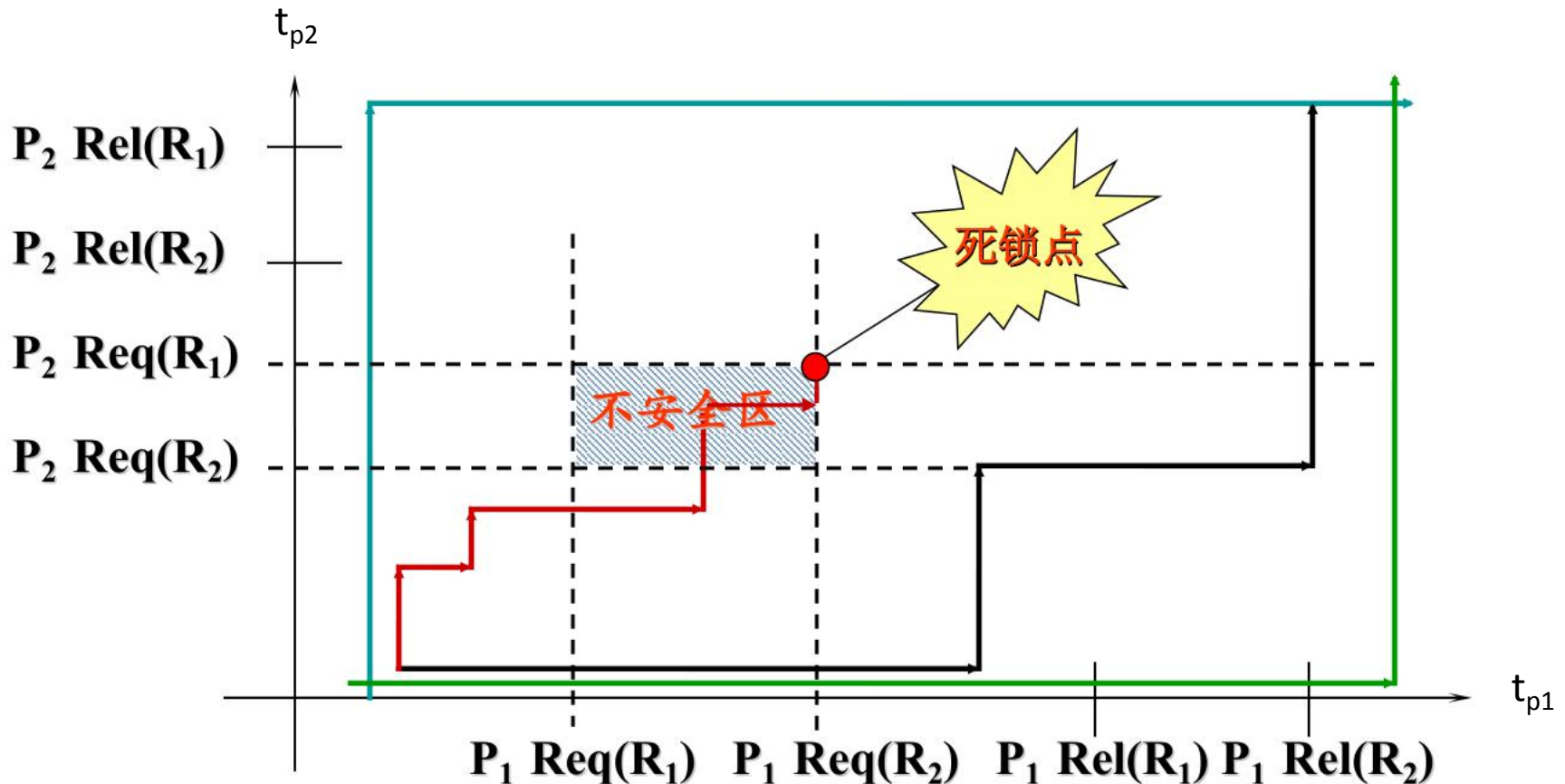
进程**P1**需要接收**S3**消息才发送**S1**消息；进程**P2**需要接收**S1**消息才发送**S2**消息；进程**P3**需要接收**S2**消息才发送**S3**消息，形成了一个环路。

图3-14 进程之间通信时的死锁



3.5.2 产生死锁的原因

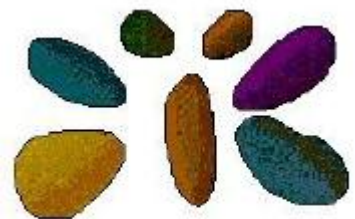
❖ 3、进程推进顺序不当引起死锁





3.5 死锁概述

- ❖ 3.5.1 资源问题
- ❖ 3.5.2 产生死锁的原因
- ❖ 3.5.3 产生死锁的必要条件
- ❖ 3.5.4 处理死锁的基本方法





3.5.3 产生死锁的必要条件

❖ 1、死锁的基本概念

死锁的定义

一组进程中每个进程都无限等待该组进程中另一进程所占有的资源，而处于的一种僵持局面，若无外力作用，该组进程中的所有进程均无法继续向前推进，这种现象称为**进程死锁**，这一组进程称为**死锁进程**。

死锁的理解

参与死锁的进程至少是**两个**；

参与死锁的进程至少有**两个已经占有资源**；

参与死锁的进程都在**等待资源**；

参与死锁的进程是当前系统中所有进程的子集。

死锁的危害

死锁的发生将会浪费大量系统资源，甚至导致系统崩溃。



3.5.2 产生死锁的必要条件

❖ 2、产生死锁的必要条件

四者必备

互斥条件

Mutual exclusion

不剥夺条件

no preemption

请求保持条件

hold and wait

环路条件

circular wait

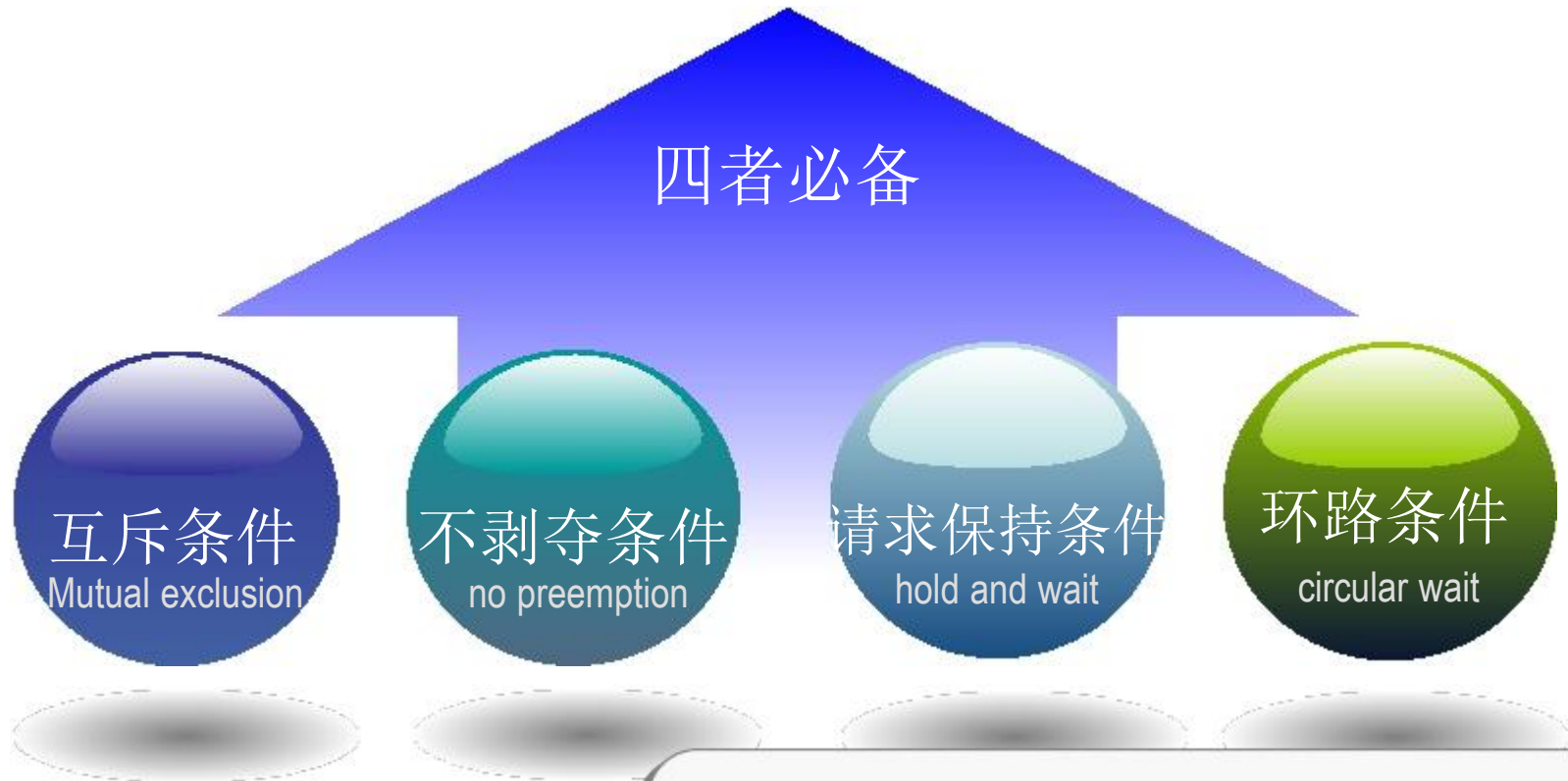
涉及的资源
是非共享的

不能剥夺进
程拥有资源

进程在请求一新
资源时继续占有
已分配的资源



3.5.2 产生死锁的必要条件



存在一种进程-资源的循环链，链中的每一个进程已获得的资源同时被链中的下一个进程所请求



3.5 死锁概述

- ❖ 3.5.1 资源问题
- ❖ 3.5.2 产生死锁的原因
- ❖ 3.5.3 产生死锁的必要条件
- ❖ 3.5.4 处理死锁的基本方法





3.5.4 处理死锁的基本方法

❖ 方法1：预防死锁(**deadlock prevention**)

方法思想

通过设置某些限制条件，去破坏死锁四个必要条件中的一个或多个，来防止死锁。

优缺点

较易实现，广泛使用。

由于所施加的限制条件往往太严格，可能导致系统资源利用率和系统吞吐量的降低。



3.5.4 处理死锁的基本方法

❖ 方法2：避免死锁(**deadlock avoidance**)

方法思想

不是事先采取限制去破坏产生死锁的条件，而是在资源的动态分配过程中，用某种方法去防止系统进入不安全状态，从而避免死锁的发生。

优缺点

事先只需要较弱的限制条件，可获得较高的资源利用率和系统吞吐量。

实现较难。



3.5.4 处理死锁的基本方法

❖ 方法3：检测死锁(**deadlock detection**)

方法思想

事先并不采取任何限制，也不检查系统是否进入不安全区，**允许死锁发生**；

通过检测机构及时检测出死锁的发生，并精确确定与死锁有关的进程和资源，然后采取适当措施，将系统中已发生的死锁清除掉。





3.5.4 处理死锁的基本方法

❖ 方法4：解除死锁(**deadlock recovery**)

方法思想

与检测死锁相配套，用于将进程从死锁状态解脱出来。

常用的方法是撤消或挂起一些进程，以回收一些资源，再将它们分配给处于阻塞状态的进程，使之转为就绪状态。

优缺点

可获得较好的资源利用率和系统吞吐量。

实现难度大。



3.5.4 处理死锁的基本方法

❖ 方法5：鸵鸟算法 (The Ostrich Algorithm)

方法思想

Pretend there is no problem

方法依据

Deadlocks occur very rarely

Cost of prevention is high

It is a trade off between convenience and correctness (可以理解为：平衡性能和复杂性)

实际应用

UNIX and Windows take this approach





本章主要内容

- ❖ 3.1 处理机调度的层次和调度算法的目标
- ❖ 3.2 作业与作业调度
- ❖ 3.3 进程调度
- ❖ 3.4 实时调度
- ❖ 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





3.6 预防死锁

❖ 预防死锁基本原理

方法思想

通过设置某些限制条件，去破坏死锁四个必要条件中的一个或多个，来防止死锁。

四大必要条件是否都可破坏？

互斥条件 —— 控制临界资源的访问，破坏“困难”

请求和保持条件

不剥夺条件

环路等待条件

可以破坏



3.6 预防死锁

❖ 1、摒弃“互斥”条件？

基本思想

互斥条件的含义

进程对所分配到的临界资源独自占有。

如何破坏？

破坏互斥条件是让进程不独自占有资源，但是，从资源本身来讲，要让多个进程同时使用是不现实的，正如多个进程同时使用打印机，会造成打印机使用混乱一样。因此，对大多数资源来讲，破坏互斥条件根本不现实。



3.6 预防死锁

❖ 2、摒弃“请求和保持”条件

全分配全释放

基本思想

请求和保持条件的含义

进程在请求一些资源的同时继续保持已有的资源。

如何破坏？

方法1（破坏请求）：进程在请求资源时必须一次性申请其在整个运行过程所需的全部资源，一旦获得了资源，以后不得再提出另外的资源请求（**保持时不请求**）

方法2（破坏保持）：分配资源时只要进程请求的资源有一种不能满足，就不给该进程分配任何资源，即该进程等待时不占有任何资源（**请求时不保持**）



3.6 预防死锁

优缺点

优点

简单、易于实现且安全。

缺点

浪费严重：进程一次性获得所需全部资源，其中**有些资源很少或很晚才使用**。如：当用户作业出错时才需要打印机输出错误信息，但采用静态分配法必须把打印机分配给该作业，并长期占用。采用该方法对系统来说是非常浪费的。

延迟运行：用户作业必须等待，直到所有资源满足才能运行。

难以提全：一个用户在作业运行之前可能提不出它的作业将要使用的全部资源。



3.6 预防死锁

❖3、摒弃“不剥夺”条件

基本思想

不剥夺条件的含义

进程占有的资源只有等该进程用完后由其自行释放。

如何破坏？

一个已拥有某些资源的进程，若它再提出新资源要求而不能立即得到满足时，它必须主动释放其已拥有的全部资源，以后需要时再重新申请。

与摒弃“请求和保持”条件不同，此处进程只要拥有部分资源就运行，是一种动态资源分配。



3.6 预防死锁

优缺点

优点

资源采用动态分配方式，资源利用率较高。

缺点

- (1) 实现比较复杂
- (2) 运行代价高昂（可能会造成以前的工作失效、两次运行的信息不连续、资源的反复申请和释放（系统开销增大）、进程执行的推迟）。



3.6 预防死锁

❖4、摒弃“环路等待”条件

基本思想

环路等待条件的含义

进程、资源形成了一个环路。

如何破坏？

系统将所有资源按类型进行线性排序，并赋予不同的序号，**所有进程对资源的请求必须严格按照资源序号递增的次序提出。**

例如：系统中有下列设备：输入机（1），打印机（2），穿孔机（3），磁带机（4），磁盘（5）。有一进程要先后使用输入机、磁盘、打印机，则它申请设备时要按输入机、打印机、磁盘的顺序申请。

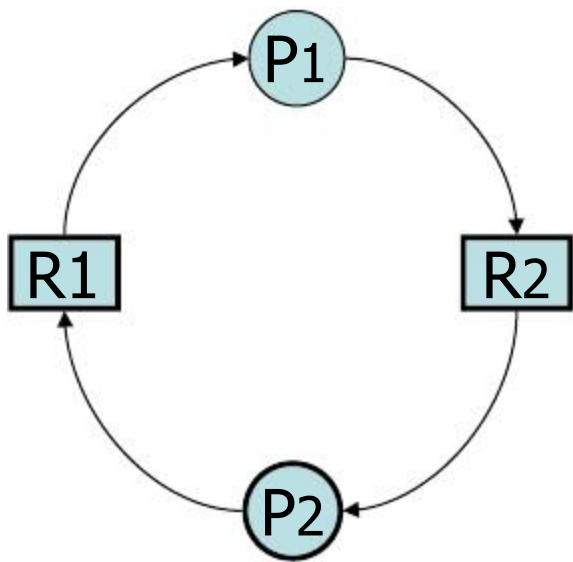


3.6 预防死锁

思考

为什么对所有资源进行线性排序，按照资源序号递增的次序分配，就可以解决死锁问题呢？

例：如图所示，请证明采用摒弃环路等待条件的办法可以预防死锁的发生。



证明：采用**反证法**，假设采用摒弃环路等待条件依旧出现死锁。由图可知，**P1**进程占有**R1**申请**R2**，则说明**R1**的序号小于**R2**的序号；同理，**P2**进程占有**R2**申请**R1**，说明**R2**的序号小于**R1**的序号，这是一个矛盾的结果，所以假设不成立，证毕。



3.6 预防死锁

优缺点

优点

同前两种方法相比，其资源利用率和系统吞吐量有较明显的改善。

缺点

- (1) 进程实际需要资源的顺序不一定与资源的编号一致，因此仍会造成资源浪费。
- (2) 资源的序号必须相对稳定，从而限制了新设备的增加。
- (3) 按规定次序申请资源限制了用户编程的自主性。



3.6 预防死锁

❖ 小结

预防死锁通过设置某些限制条件以破坏死锁四个必要条件中的一个或多个，来防止死锁。

- (1) 互斥条件一般不允许破坏。
- (2) 破坏请求和保持条件的资源分配算法是一种静态资源分配。
- (3) 破坏不可剥夺条件和环路条件是动态资源分配算法。
- (4) 破坏环路条件是相对优秀的一种预防死锁算法。



本章主要内容

- ❖ 3.1 处理机调度的层次和调度算法的目标
- ❖ 3.2 作业与作业调度
- ❖ 3.3 进程调度
- ❖ 3.4 实时调度
- ❖ 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

不破坏产生死锁的必要条件，而是在资源动态分配过程中，防止系统进入不安全状态



3.7 避免死锁

- ❖ 3.7.1 系统安全状态（避免死锁）
- ❖ 3.7.2 利用银行家算法避免死锁





3.7.1 系统安全状态

寻找安全系列!!

❖ 避免死锁基本原理

在系统运行过程中，对进程提出的每一个（系统能够满足的）资源申请进行安全性检查，如果安全，则予以分配，否则不予分配。

❖ 安全状态与不安全状态

概念：如果系统能按某种进程顺序 $\langle \underline{P_1}, \underline{P_2}, \dots, \underline{P_n} \rangle$ 为每个进程依次分配其所需的资源，直至满足每个进程对资源的最大需求，使每个进程都可顺利完成，称此时系统处于安全状态。

概念：进程顺序 $\langle \underline{P_1}, \underline{P_2}, \dots, \underline{P_n} \rangle$ 称为安全序列。

概念：若系统中不存在这样一个安全序列称此时系统处于不安全状态。




3.7.1 系统安全状态

❖ 安全状态应用举例

系统有三个进程P1、P2、P3，12台磁带机，P1、P2、P3共要求10、4、9台，在T₀时刻，三个进程分别获得5、2、2台，尚有3台空闲，请问此时系统是否处于安全状态？

进程	最大需求	已分配	还需	可用
P ₁	10	5	5	3 5 10 12
P ₂	4	2	2	
P ₃	9	2	7	



因在T₀时刻存在一个安全序列<P₂,P₁,P₃>，故系统此刻处于安全状态



3.7.1 系统安全状态

如果不按安全序列分配资源，则系统可能会由安全状态进入不安全状态。

如在 T_0 以后， P_3 要求1台磁带机，若系统先分给它一台，则系统进入不安全状态

进程	最大需求	已分配	还需	可用
P_1	10	5	5	2
P_2	4	2	2	
P_3	9	3	6	

进入不安全状态就一定会发生死锁吗？

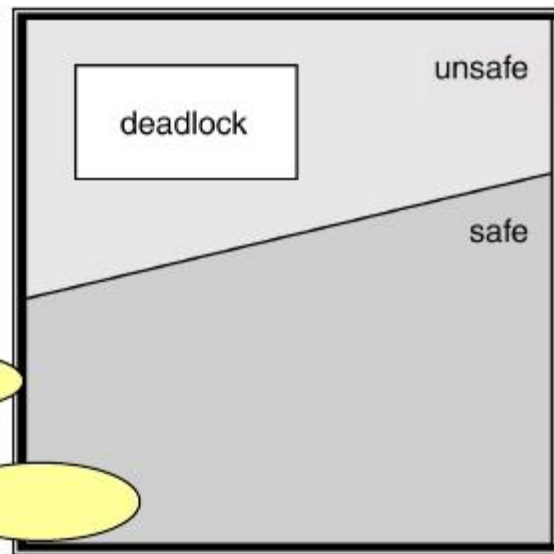


3.7.1 系统安全状态

❖ 注意

(1) 不安全状态 \neq 死锁

处于不安全状态的系统
不一定会发生死锁(因为
安全性检查中使用的最大资源
需求量 \mathbf{max} 是进程执行前提供
的,而在实际运行过程中,进
程需要的最大资源量可能小于
 \mathbf{max})



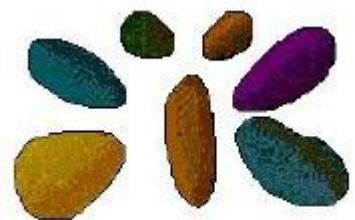
(2) 处于安全状态的系统一 定不会发生死锁

例如,一进程对应的程序中有一段
进行异常处理的代码需要 \mathbf{n} 个某种资
源,若该进程在运行过程中没有发生
异常就不需要调用这段代码,此
时它将不会请求这 \mathbf{n} 个某种资源。



3.7 避免死锁

- ❖ 3.7.1 系统安全状态（避免死锁）
- ❖ 3.7.2 利用银行家算法避免死锁





3.7.2 利用银行家算法避免死锁

Dijkstra提出，
最有代表性的
避免死锁
算法

❖ 1、算法中的数据结构

可用资源向量

$\text{available}[j]$ 表示系统现有 R_j 类可用资源的数量

资源最大需求矩阵

$\text{max}[i,j]$ 表示进程 i 对 R_j 类资源的最大需求的数量

资源分配矩阵

$\text{allocation}[i,j]$ 表示进程 i 现已共分得 R_j 类资源的数量

资源需求矩阵

$\text{need}[i,j]$ 表示进程 i 还需要 R_j 类资源的数量

$$\text{need}[i,j] = \text{max}[i,j] - \text{alloc}[i,j]$$



3.7.2 利用银行家算法避免死锁

资源请求向量

request_i[j]表示现阶段**进程i**请求**R_j**类资源的数量

工作向量

work[j]

表示现阶段**系统**可向进程提供的**R_j**类资源的数量

布尔向量

finish[i]取值为**true**或**false**

表示系统是否有足够的资源分配给进程**i**



3.7.2 利用银行家算法避免死锁

❖ 2、银行家算法

当进程 P_i 发出资源请求 $request_i[j]$ 后，系统按下述步骤进行检查、分配：

(1)如果 $request_i[j] \leq need[i,j]$ ，便转向步骤2；否则认为出错，因为它所需要的资源数已超过它所宣布的最大值。

(2)如果 $request_i[j] \leq available[j]$ ，便转向步骤(3)；否则，表示尚无足够资源， P_i 须等待。

(3)系统试探着把资源预分配给进程 P_i ，并修改下面数据结构中的数值：

$available[j] = available[j] - request_i[j];$

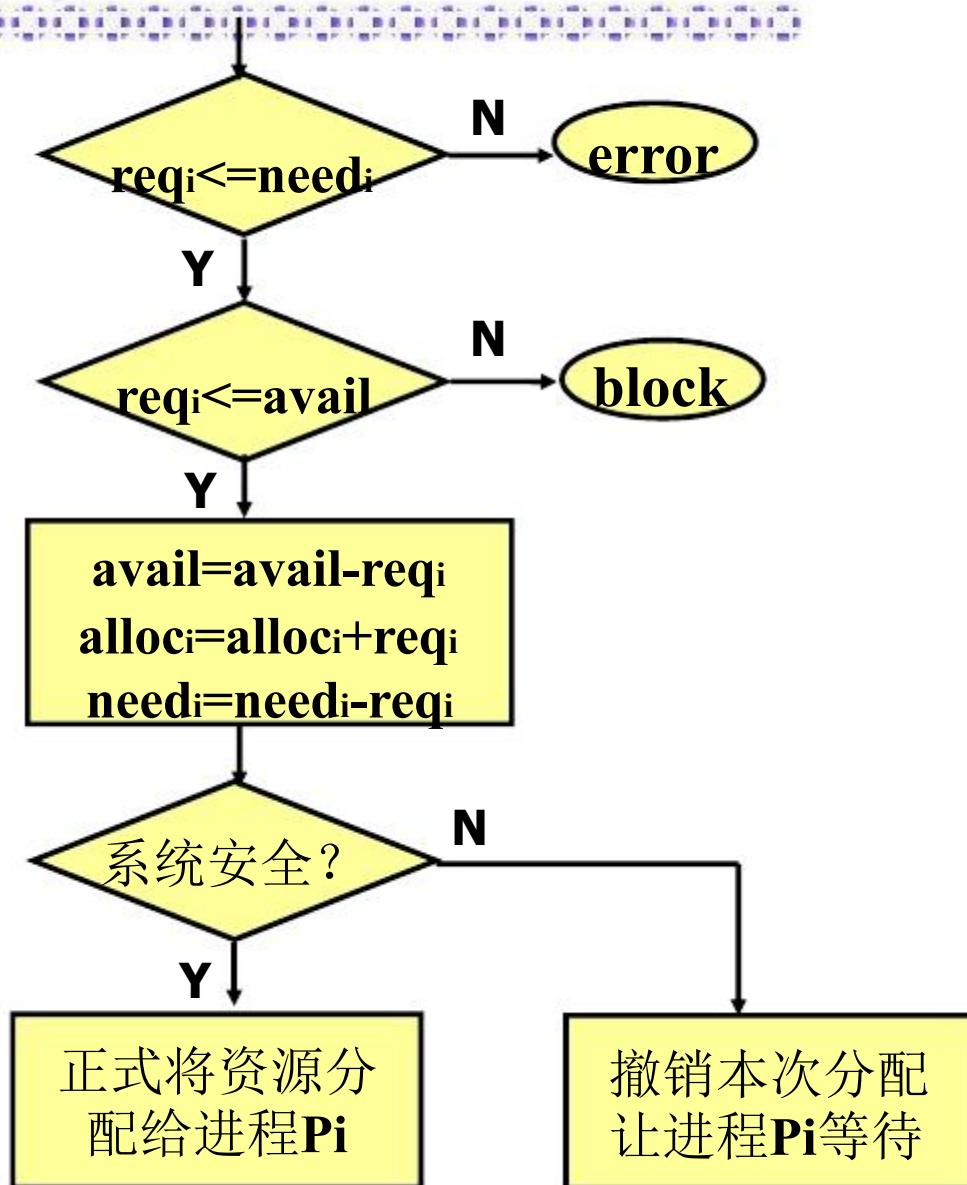
$allocation[i,j] = allocation[i,j] + request_i[j];$

$need[i,j] = need[i,j] - request_i[j];$



3.7.2 利用银行家算法避免死锁

(4) 系统执行安全性算法，检查此次资源分配后，系统是否处于安全状态：若安全，才正式将资源分配给进程 P_i ，以完成本次分配；否则，将本次的预分配作废，恢复原来的资源分配状态，让进程 P_i 等待。





3.7.2 利用银行家算法避免死锁

❖ 3、安全性算法 (这个算法嵌套在银行家算法内部)

(1) 初始化 $\text{work} = \text{available}$ 、 $\text{finish}[i] = \text{false}$;

(2) 从进程集合中找到一个能满足下述条件的进程 P_i :

① $\text{finish}[i] = \text{false}$; ② $\text{need}[i, j] \leq \text{work}[j]$; 若找到, 执行步骤(3), 否则, 执行步骤(4)。

(3) 当进程 P_i 获得资源后, 便可顺利执行直至完成, 并释放出分配给它的资源, 故应执行:

$\text{work}[j] = \text{work}[j] + \text{allocation}[i, j]$; $\text{finish}[i] = \text{true}$; 转向步骤(2)

(4) 如果所有进程的 $\text{finish}[i] = \text{true}$, 则表示系统处于安全状态; 否则, 系统处于不安全状态。

寻找安全系列!!



3.7.2 利用银行家算法避免死锁

❖ 4、银行家算法应用示例

假定系统中有五个进程 {P0, P1, P2, P3, P4} 和三类资源 {A, B, C}, 各种资源的总量分别为10、5、7, 在T₀时刻的资源分配情况如图所示。

进程	max A B C	allocation A B C	need A B C	available A B C
P0	7 5 3	0 1 0	7 4 3	3 3 2
P1	3 2 2	2 0 0	1 2 2	
P2	9 0 2	3 0 2	6 0 0	
P3	2 2 2	2 1 1	0 1 1	
P4	4 3 3	0 0 2	4 3 1	

进程	need A B C	available A B C
P0	7 4 3	3 3 2
P1	1 2 2	
P2	6 0 0	
P3	0 1 1	
P4	4 3 1	

❖ 问题1: T₀时刻是否安全? (用安全性算法检查)

进程	work A B C	need A B C	allocation A B C	work+alloc A B C	finish
P1	3 3 2	1 2 2	2 0 0	5 3 2	true
P3	5 3 2	0 1 1	2 1 1	7 4 3	true
P4	7 4 3	4 3 1	0 0 2	7 4 5	true
P2	7 4 5	6 0 0	3 0 2	10 4 7	true
P0	10 4 7	7 4 3	0 1 0	10 5 7	true

安全序列不一定是唯一的

安全



3.7.2 利用银行家算法避免死锁

❖ **问题2:** P1发出资源请求request₁(1,0,2), 可否分配 ?

1> 利用银行家算法检查, 可以预分配

进程	max A B C	allocation A B C	need A B C	available A B C
P0	7 5 3	0 1 0	7 4 3	3 3 2 2 3 0
P1	3 2 2	2 0 0 3 0 2	1 2 2 0 2 0	
P2	9 0 2	3 0 2	6 0 0	
P3	2 2 2	2 1 1	0 1 1	
P4	4 3 3	0 0 2	4 3 1	

进程	need A B C	available A B C
P0	7 4 3	2 3 0
P1	0 2 0	
P2	6 0 0	
P3	0 1 1	
P4	4 3 1	

2> 利用安全性算法检查，可以正式分配？

进程	work A B C	need A B C	allocation A B C	work+alloc A B C	finish
P1	2 3 0	0 2 0	3 0 2	5 3 2	true
P3	5 3 2	0 1 1	2 1 1	7 4 3	true
P4	7 4 3	4 3 1	0 0 2	7 4 5	true
P0	7 4 5	7 4 3	0 1 0	7 5 5	true
P2	7 5 5	6 0 0	3 0 2	10 5 7	true





3.7.2 利用银行家算法避免死锁

❖ **问题3:** P1请求的资源得到分配后，接着P4发出资源请求 $\text{request}_4(3,3,0)$ ，可否分配？

1> 利用银行家算法检查，算法终止

进程	max A B C	allocation A B C	need A B C	available A B C
P0	7 5 3	0 1 0	7 4 3	2 3 0
P1	3 2 2	3 0 2	0 2 0	
P2	9 0 2	3 0 2	6 0 0	
P3	2 2 2	2 1 1	0 1 1	
P4	4 3 3	0 0 2	4 3 1	

可用资源不足
以满足需求



3.7.2 利用银行家算法避免死锁

❖ **问题4:** 在P4之后, P0发出资源请求requesto(0,2,0), 可否分配?

1> 利用银行家算法检查, 可以预分配

进程	max A B C	allocation A B C	need A B C	available A B C
P0	7 5 3	0 1 0 0 3 0	7 4 3 7 2 3	2 3 0 2 1 0
P1	3 2 2	3 0 2	0 2 0	
P2	9 0 2	3 0 2	6 0 0	
P3	2 2 2	2 1 1	0 1 1	
P4	4 3 3	0 0 2	4 3 1	



3.7.2 利用银行家算法避免死锁

2> 利用安全性算法检查，不能满足任何进程需要，需要撤消上一阶段对P0的预分配。

进程	work A B C	need A B C	allocation A B C	work+alloc A B C	finish
P0	2 1 0	7 2 3	0 3 0	————	flase
P1	2 1 0	0 2 0	3 0 2	————	flase
P2	2 1 0	6 0 0	3 0 2	————	flase
P3	2 1 0	0 1 1	2 1 1	————	flase
P4	2 1 0	4 3 1	0 0 2	————	flase

不安全



本章主要内容

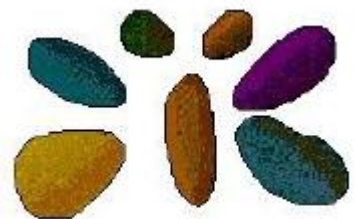
- ❖ 3.1 处理机调度的层次和调度算法的目标
- ❖ 3.2 作业与作业调度
- ❖ 3.3 进程调度
- ❖ 3.4 实时调度
- ❖ 3.5 死锁概述
- ❖ 3.6 预防死锁
- ❖ 3.7 避免死锁
- ❖ 3.8 死锁的检测与解除





3.8 死锁的检测与解除

- ❖ 3.8.1 死锁的检测
- ❖ 3.8.2 死锁的解除





3.8.1 死锁的检测

❖ 1、死锁检测基本原理

允许死锁发生，操作系统不断监视系统进展情况，判断死锁是否发生。

一旦死锁发生则采取专门的措施，解除死锁并以最小的代价恢复操作系统运行。

❖ 2、死锁检测时机

定时检测

当进程阻塞时检测死锁（其缺点是系统的开销大）

系统资源利用率下降时检测死锁



3.8.1 死锁的检测

❖ 3、资源分配图

$$G = (V, E)$$

结点集 $V = P \cup R$

进程结点集 $P = \{P_1, P_2, \dots, P_n\}$

资源结点集 $R = \{r_1, r_2, \dots, r_m\}$

边集 E -- 其元素为有序二元组 $\langle P_i, r_j \rangle$ 或 $\langle r_j, P_i \rangle$

请求边（申请边） $\langle P_i, r_j \rangle$

分配边 $\langle r_j, P_i \rangle$

资源类与资源实例

系统中类型相同的资源构成一个资源类；

每个资源类中包含若干个同种资源，称为资源实例。



3.8.1 死锁的检测

结点的表示

资源结点

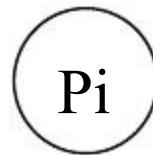
资源类用方框表示

资源实例用方框中的黑圆点（圈）表示



进程结点

用含有进程名的圆圈表示





3.8.1 死锁的检测

资源分配图示例

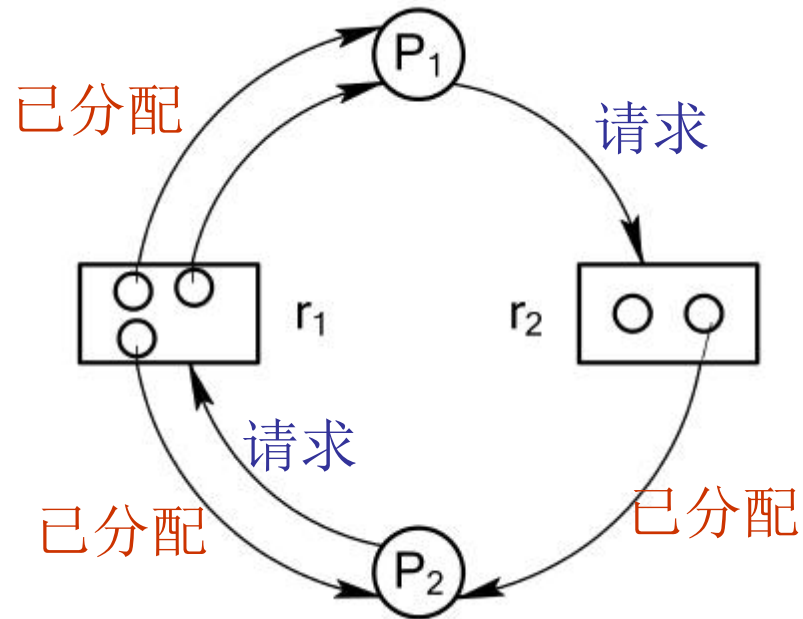


图3-20 每类资源有多个时的情况



3.8.1 死锁的检测

资源分配图化简（进程结点孤立化）

- 1> 从分配图中找一个非孤立、非阻塞（其所有请求边可转化为分配边，转化后该结点只有分配边）的进程结点，去掉其请求边和分配边，将其变为孤立结点；
- 2> 再把相应的资源分配给等待该资源的进程，将这些进程中的相应请求边变为分配边；
- 3> 重复以上步骤，若所有进程都可成为孤立结点，称该图是可完全简化的，否则称该图是不可完全简化的。

3.8.1 死锁的检测

资源分配图化简示例

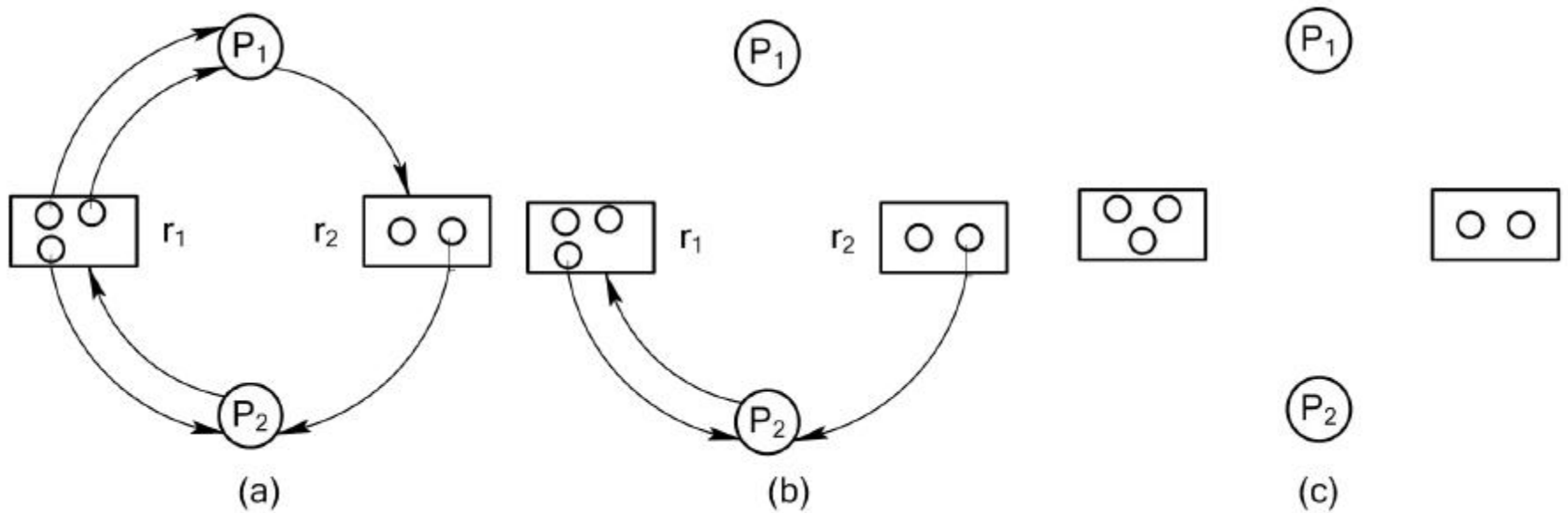


图3-21 资源分配图的简化



3.8.1 死锁的检测

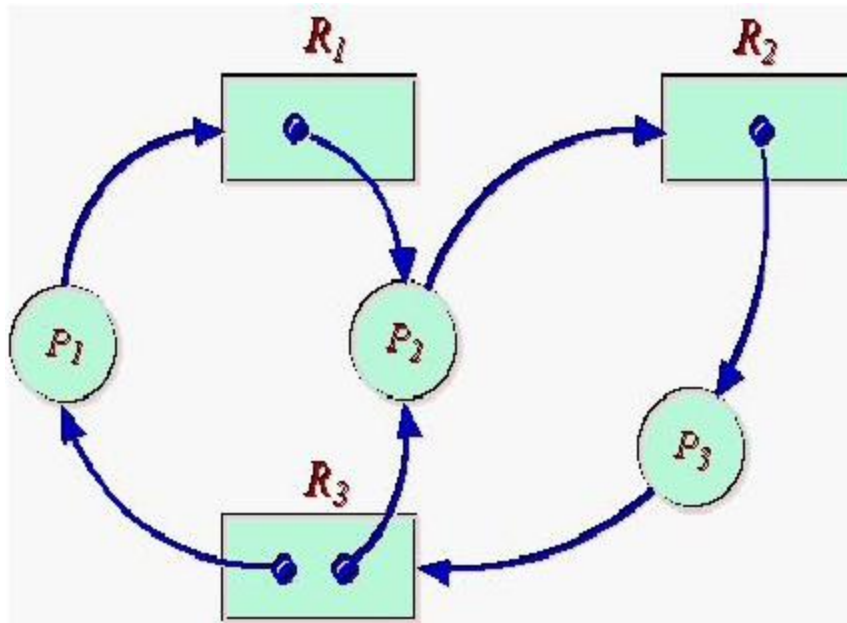
❖4、死锁定理

(1) **死锁的必要条件**：死锁状态一定有环路；如果资源分配图中没有环路，则系统中没有死锁，如果图中存在环路，则系统中可能存在死锁。

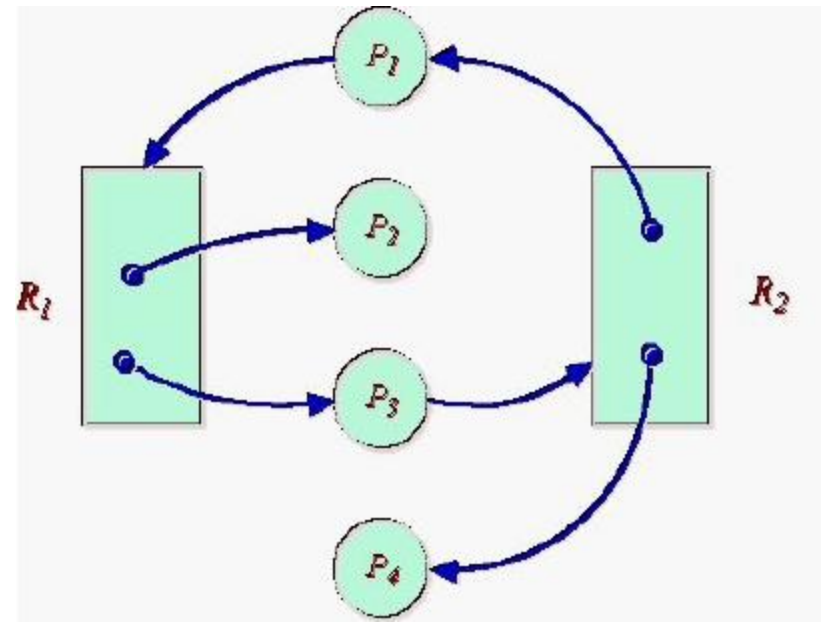
(2) **死锁的充分条件（死锁定理）**：当且仅当其资源分配图不可完全简化。

(3) **死锁的充要条件**：如果每个资源类中只包含一个资源实例，则环路是死锁存在的充分必要条件。

3.8.1 死锁的检测



有环有死锁



有环无死锁



3.8.1 死锁的检测

❖ 5、死锁检测（资源分配图化简）算法描述

Work:= available

P:={ p_i | $\text{allocation}_i=0 \cap \text{request}_i=0$ } /*存放孤立进程点*/

For all $p_i \notin P$ do

Begin

For all $\text{request}_i \leq \text{work}$ do

Begin

work:=work+ allocation_i

P=P \cup p_i

End

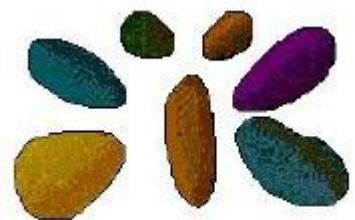
End

Deadlock:= \neg (P=={ $p_1 \dots p_n$ }) /*Deadlock为true时死锁*/



3.8 死锁的检测与解除

- ❖ 3.8.1 死锁的检测
- ❖ 3.8.2 死锁的解除（了解）





3.8.2 死锁的解除

❖ 1、死锁解除常用方法

重启OS

以前工作全部作废，损失可能很大

进程回退

根据系统保存的检查点**checkpoint**，让所有进程回退再重新启动，直到解除死锁。

抢占（剥夺）资源

从其它进程抢占（剥夺）资源给死锁进程

撤消进程

按一定策略撤消死锁进程（全部撤消、逐个撤消、...）
应使撤消进程所付出的代价尽可能小



3.8.2 死锁的解除

❖ 2、撤消进程的方法

全部撤消

以前工作全部作废，损失可能很大

逐个撤消

应使撤消进程所付出的代价尽可能小，**选取被撤消进程的参考因素**有：

进程的优先级大小

进程已执行多长时间，还要多长时间

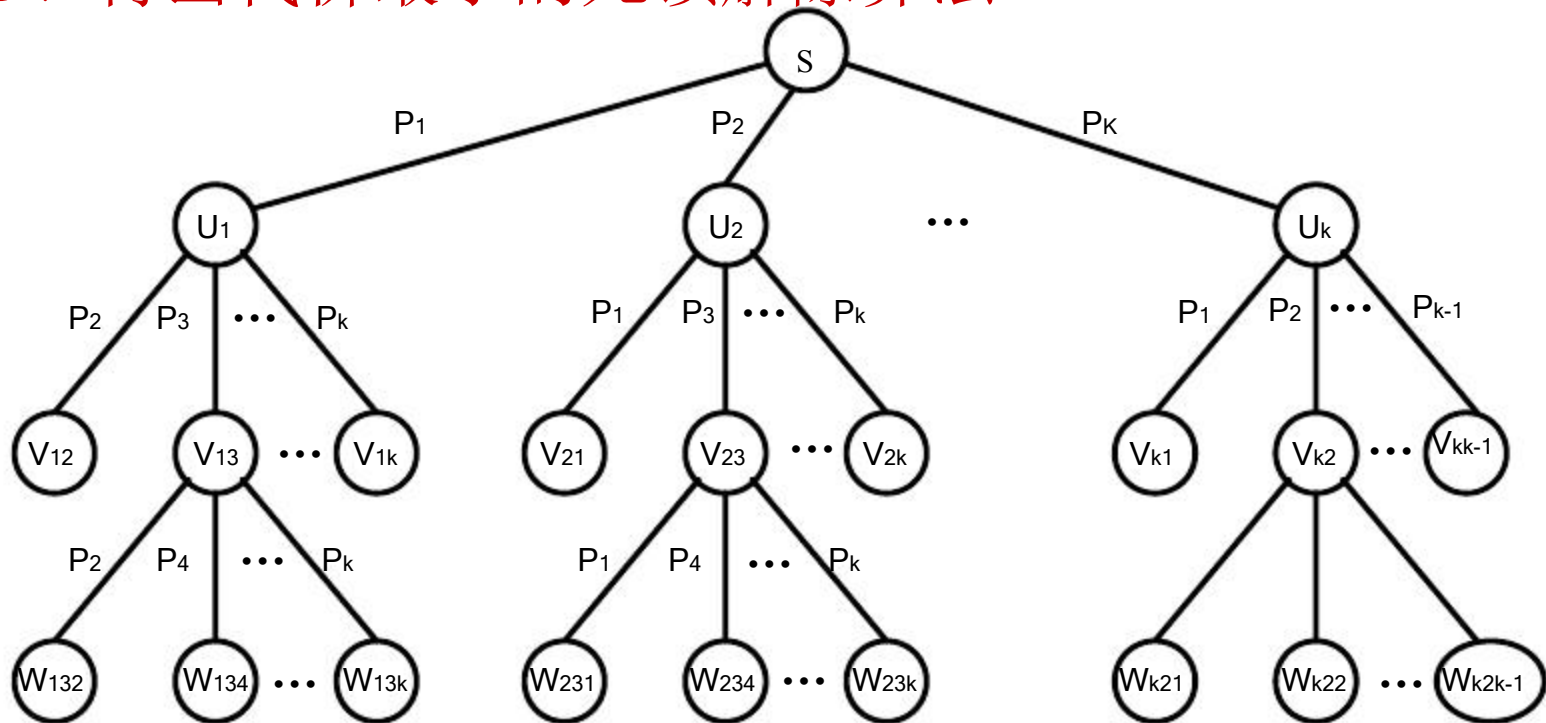
进程已使用多少资源，还需多少资源

进程是交互式的还是批处理式的



3.8.2 死锁的解除

❖ 3、付出代价最小的死锁解除算法



■ 撤消策略

- ✎ 1> 按树的广度优先搜索方法撤消：代价可能太大
- ✎ 2> 按树的最短路径（最小代价）优先方法撤消：比较好



3.8.2 死锁的解除

❖ 补充：用经验公式判断死锁

问题1：设系统仅有一类数量为 M 的独占型资源，系统中 N 个进程竞争该类资源，其中各进程对该类资源的最大需求均为 W 。当 M ， N ， W 分别取下列值时，试判断下列哪些情况会发生死锁？

- (1) $M=2$ ， $N=2$ ， $W=2$;
- (2) $M=3$ ， $N=2$ ， $W=2$;
- (3) $M=3$ ， $N=2$ ， $W=3$;
- (4) $M=5$ ， $N=3$ ， $W=2$;
- (5) $M=6$ ， $N=3$ ， $W=3$;



3.8.2 死锁的解除

死锁判断的经验公式：

$$x = \begin{cases} 1 & M \leq N \\ 1 + \frac{M-1}{N} & M > N \end{cases}$$

若 $W \leq x$ ，则不会发生死锁；若 $W > x$ ，则可能导致死锁。

3.8.2 死锁的解除

(1) $M=2, N=2, W=2;$

(2) $M=3, N=2, W=2;$

(3) $M=3, N=2, W=3;$

(4) $M=5, N=3, W=2;$

(5) $M=6, N=3, W=3;$

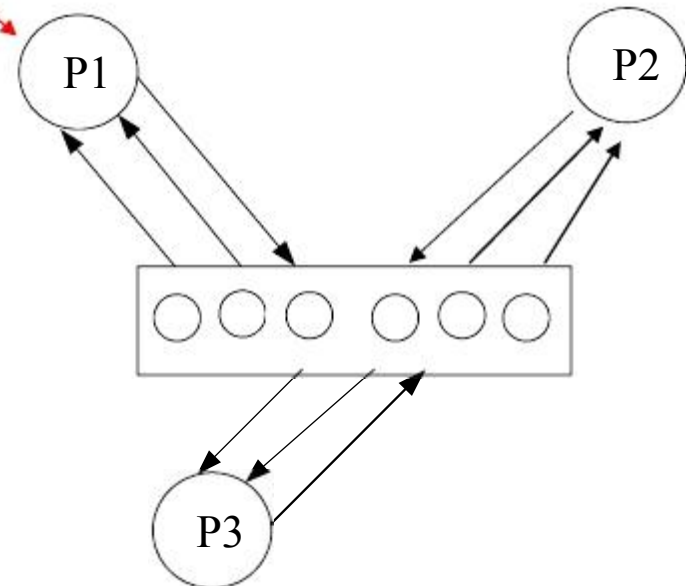
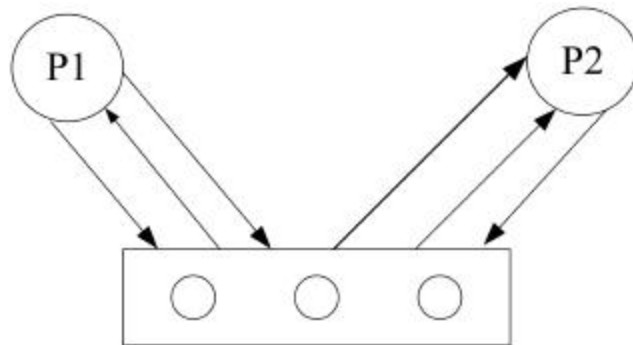
$x=1, x < W$, 可能会死锁

$x=2; x=W$, 不会死锁

$x=2; x < W$, 可能会死锁

$x=7/3; x > W$, 不会死锁

$x=8/3; x < W$, 可能会死锁





3.8.2 死锁的解除

问题2：一台计算机有**10**台磁带机被**n**个进程竞争，每个进程最多需要**3**台磁带机，那么**n**最多为_____时，系统没有死锁的危险？

解：依题意， **$W=3$** ， **$M=10$** ， **$x=9/n+1$**

由经验公式如要不死锁，则需 **$x \geq W$**

即： **$9/n+1 \geq 3$**

$n \leq 4.5$

故 **$n_{max}=4$**



本章小结

❖ 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**

❖ 产生死锁的三类原因

❖ 死锁的四大必要条件**

❖ 死锁的预防*





本章小结

❖ 死锁的避免（银行家算法）**

❖ 死锁的检测**

死锁定理、资源分配图化简、死锁检测算法

❖ 死锁的解除

❖ 重要概念

作业、作业步、作业流、非抢占方式、抢占方式、周转时间、带权周转时间、静态优先权、动态优先权、响应比、松弛度、安全状态、死锁定理



本章作业

❖ 要求:

一定要做在作业本上

❖ 交作业日期:

❖ 作业内容:

操作系统第3章网络在线测试

<http://jxpt.hainu.edu.cn/meol/homepage/communication/>



本章作业

- ❖ **补充题1:** 假设一个系统中有**5**个进程，它们到达的时间和服务时间如图所示，忽略**I/O**及其它开销时间，若分别按先来先服务(**FCFS**)、非抢占式及抢占式短进程优先(**SPF**)、高响应比优先(**HRRN**)、时间片轮转(**RR**)、多级反馈队列调度算法(**FB**，第*i*级队列的时间片= 2^{i-1})进行**CPU**调度，请给出各进程的完成时间、周转时间、带权周转时间、平均周转时间和平均带权周转时间。

进程	到达时间	服务时间
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2



本章作业

- ❖ **补充题2:** 对下面的**5**个非周期性实时任务，按最早开始截止时间优先调度算法应如何进行**CPU**调度？

进程	到达时间	执行时间	开始截止时间
A	10	20	110
B	20	20	20
C	40	20	50
D	50	20	90
E	60	20	70

- ❖ **补充题3:** 若有**3**个周期性实时任务**A**、**B**、**C**分别要求每**20ms**、**50ms**、**50ms**执行一次，每次执行时间分别为**10ms**、**10ms**、**15ms**，请用最低松弛度优先算法对它们进行调度。



本章课程结束！谢谢大家！