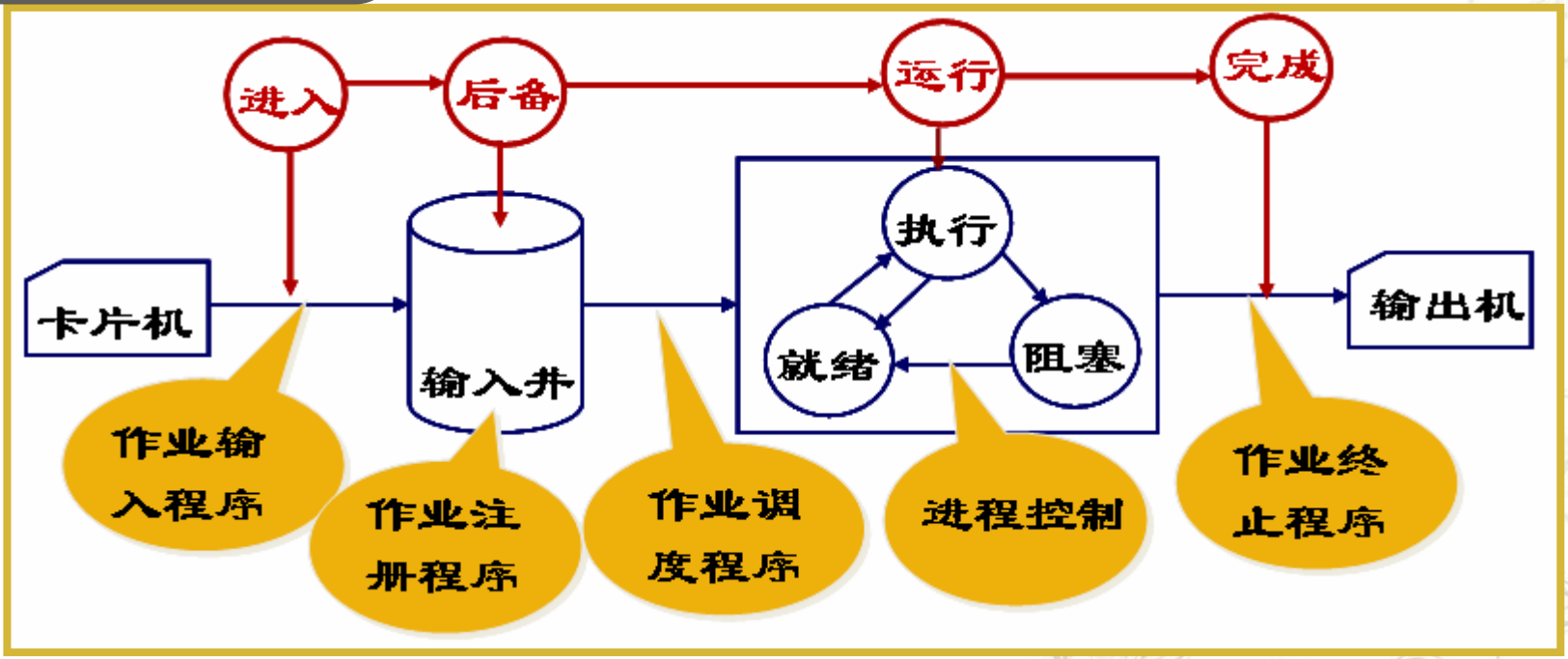
第五章 调度与死锁

§1 处理机调度的层次

一、作业状态



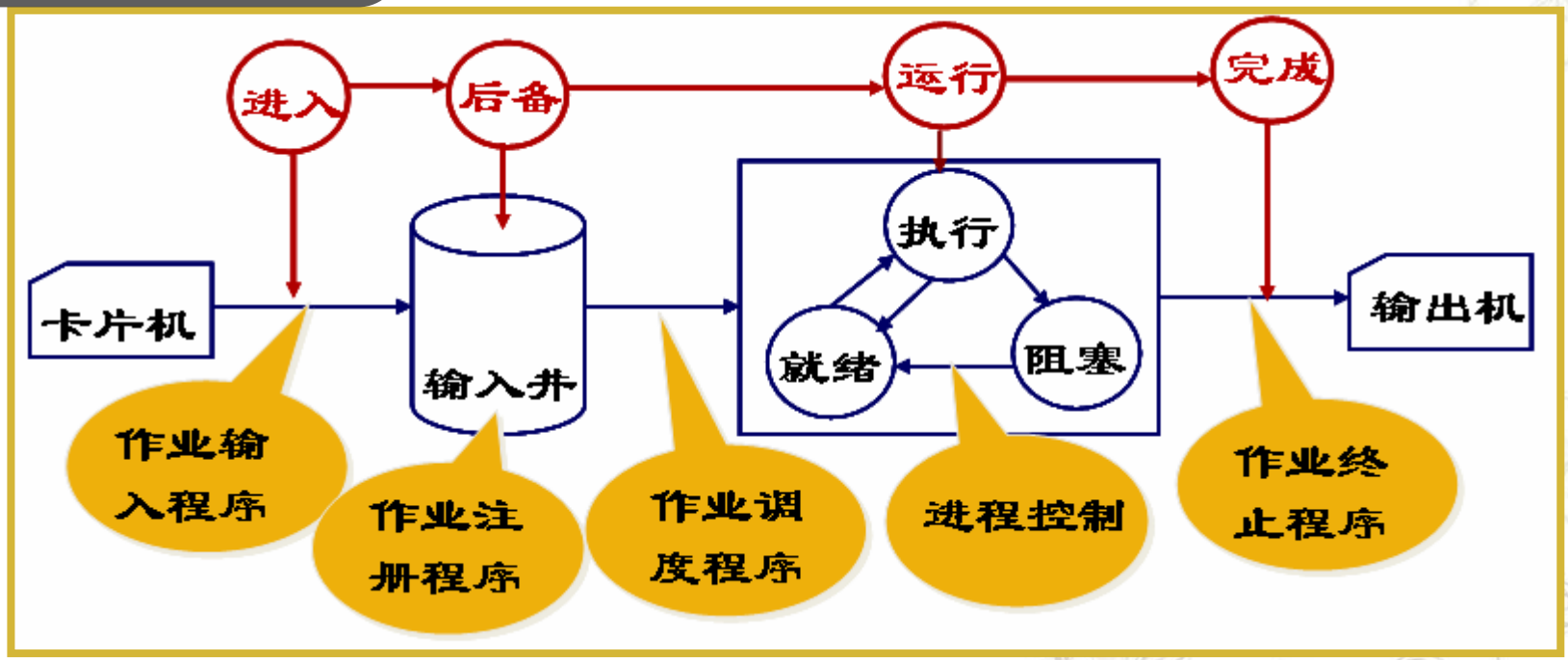
1.进入(提交)态：作业原始信息正由输入设备向输入井预输入，尚未全部到达输入井的状态。

——调用文件系统的后援存储空间分配程序为作业分配辅存；

——调用设备管理的Spooling输入程序预输入作业信息；

§1 处理机调度的层次

一、作业状态



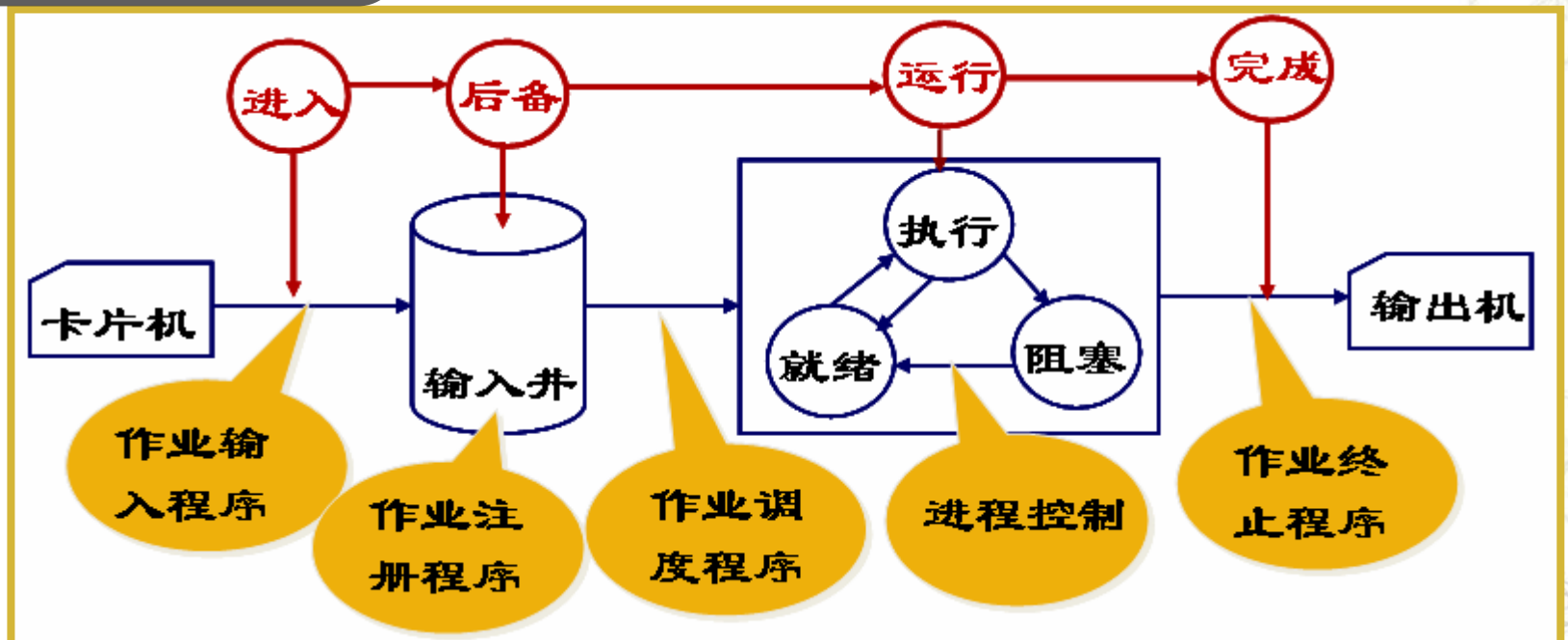
2.后备(收容)态：作业预输入结束但尚未被选中运行的状态。

——作业注册程序注册生成作业控制块JCB；

——作业注册程序将JCB插入后备作业队列；

§1 处理机调度的层次

一、作业状态



3.**运行态**：作业被作业调度选中，处于已进入主存宏观运行的状态。

——调用**主存分配程序**审核并分配主存资源；

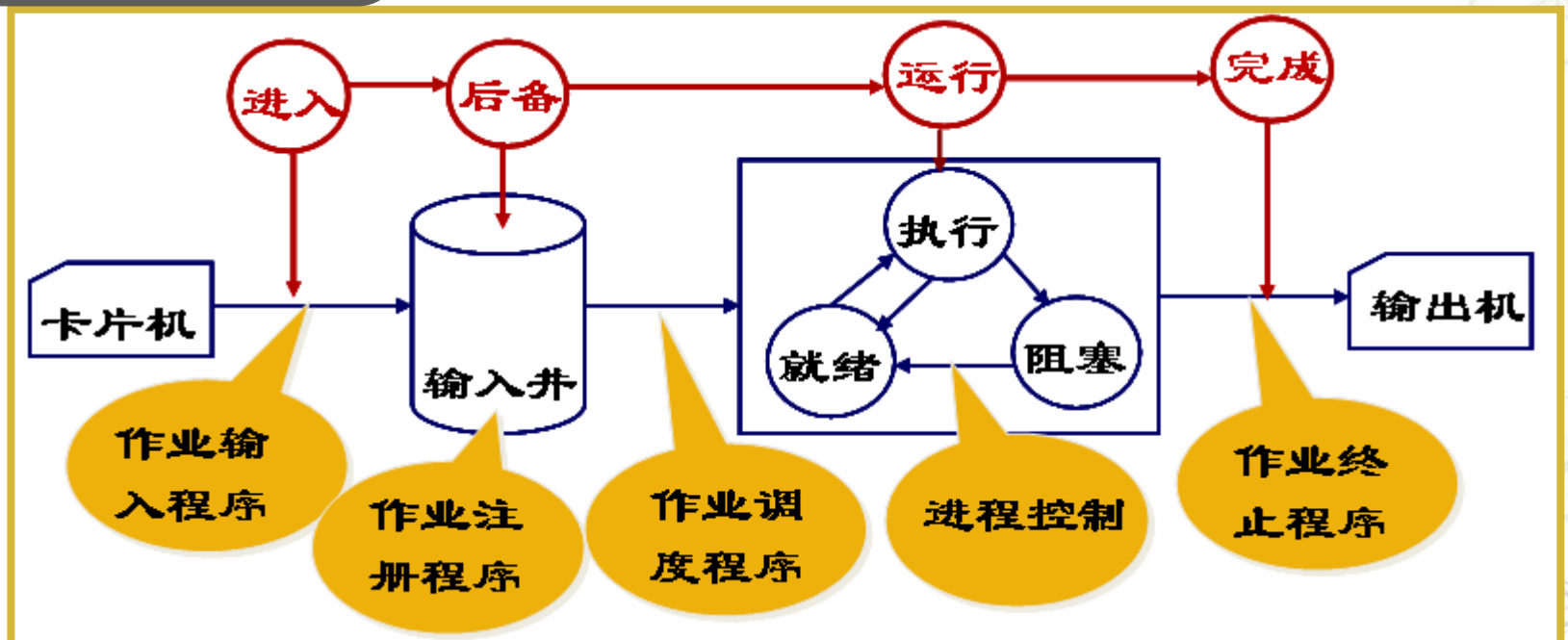
——调用**设备分配程序**审核并分配外设资源；

——调用**创建原语**为作业运行创建一个或一组并发进程；

——调用存储管理的**装入程序**装入作业体；

§1 处理机调度的层次

一、作业状态



4.**完成态**：作业运行完毕或因错误终止，正在回收资源、等待其结果输出的状态。

——调用**存储管理/设备管理**程序回收主存及外围设备；

——调用**Spooling输出程序**将结果以文件形式输出；

——**撤消其JCB**，从而将该作业从系统中注销；

§1 处理机调度的层次

二、三级调度概念

- **作业调度、高级调度、宏观调度、长程调度：**
  选择一或多个后备作业，为其分配必要资源，创建一组并发进程，并将其实体调入主存的过程。
- **交换调度、中级调度、滚进/出调度、中程调度：**
  将外存某些具备运行条件的进程调入内存；在内存不足时，将内存中某些暂时不能运行进程调出至外存的过程。
- **进程调度、低级调度、微观调度、短程调度：**
  决定获得CPU的就绪进程，并将CPU分派给它。

三、进程调度方式

- **非剥夺方式：** 除非CPU现行进程主动放弃CPU(阻塞、完成)，否则不得剥夺现行进程的CPU使用权。eg：Windows 3.x
- **剥夺方式：** 可以根据某种抢占原则(优先权原则、短进程原则、时间片原则)，剥夺现行进程的CPU。eg:Windows 95始

§1 处理机调度的层次

例：若某单处理机多进程系统中有多个就绪进程，

则下列关于处理机调度的叙述中错误的是：C。

A. 在进程结束时能进行处理机调度

B. 创建新进程后能进行处理机调度

C. 在进程处于临界区时不能进行处理机调度

D. 在系统调用完成并返回用户态时能进行处理机调度

§2 调度性能评价

设：n —— 作业流中作业数；

t_{si} —— 作业i提交时刻；

t_{Ri} —— 作业i实际所需运行时间；

t_{ci} —— 作业i完成时刻；

周转时间T_i —— 作业从提交系统到运行完成所需时间：

等待时间＋运行时间$=t_{ci}-t_{si}$。

平均周转时间$T=\sum T_i/n$

带权周转时间W_i —— 周转时间与作业运行时间之比：

$$W_i=T_i/t_{Ri}$$

平均带权周转时间$W=\sum W_i/n$

进程周转时间T：第一次进入就绪队列到从执行队列中完成

=总就绪时间+总cpu执行期+总阻塞时间

进程带权周转时间W：T/CPU服务时间

§3 调度算法

一、先来先服务(FCFS)

作业的后备队列按FCFS原则建立，例行选择队首作业调度运行

作业	t_{si}	t_{Ri}	开始时刻t_{bi}	t_{ci}	T_i	W_i
1	8.00	2.00	8.00	10.00	2.0	1
2	8.50	0.5	10.00	10.50	2.0	4
3	9.00	0.1	10.50	10.60	1.6	16
4	9.50	0.2	10.60	10.80	1.3	6.5
平均周转时间T=1.725(小时),带权平均周转W=6.875						

有利于长作业，不利于短作业

平均周转时间长

§3 调度算法

二、短作业优先(SJF)

例行选择最短运行时间的作业调度运行

作业	t_{si}	t_{Ri}	开始时刻t_{bi}	t_{ci}	T_i	W_i
1	8.00	2.00	8.00	10.00	2.0	1
2	8.50	0.5	10.30	10.80	2.3	4.6
3	9.00	0.1	10.00	10.10	1.1	11
4	9.50	0.2	10.10	10.30	0.8	4
平均周转时间T=1.55(小时),带权平均周转W=5.15						

吞吐量大

可能造成长作业的“饥饿”现象

§3 调度算法

二、短作业优先(SJF)

例 :设有4个作业，它们的到达时间、所需运行时间如下图所示，若采用先来先服务、短作业优先和静态优先权的调度算法，则平均周转时间分别为___、___、___。

作业号	到达时间	所需运行时间（小时）	优先数
1	0	2	4
2	1	5	9
3	2	8	1
4	3	3	8

§3 调度算法

二、短作业优先(SJF)

作业号	到达时间	运行时间	开始时刻	完成时刻	Ti
1	0	2	0	2	2
2	1	5	2	7	6
3	2	8	10	18	16
4	3	3	7	10	7
T=7.75					

作业号	到达时间	所需运行时间（小时）	优先数
1	0	2	4
2	1	5	9
3	2	8	1
4	3	3	8

§3 调度算法

三、最高响应比优先(HRN)

- 选择响应比最高的作业投入运行：

响应比R_p=响应时间/运行时间＝1＋等待时间/运行时间

—— 对等长作业，体现了FCFS的公平原则

—— 在相同的等待时间下，体现短作业优先原则

—— 改善了长作业迟迟无法运行的现象

作业	t_{si}	t_{Ri}	开始时刻t_{bi}	t_{ci}	T_i	W_i
1	8.00	2.00	8.00	10.00	2.0	1
2	8.50	0.5	10.10	10.60	2.1	4.2
3	9.00	0.1	10.00	10.10	1.1	11
4	9.50	0.2	10.60	10.80	1.3	6.5
平均周转时间T=1.625(小时),带权平均周转W=5.675						

§3 调度算法

三、最高响应比优先(HRN)

- 选择响应比最高的作业投入运行：

  响应比R_p=响应时间/运行时间＝1＋等待时间/运行时间

  —— 对等长作业，体现了FCFS的公平原则

  —— 在相同的等待时间下，体现短作业优先原则

  —— 改善了长作业迟迟无法运行的现象

例：一作业8:00到达系统，估计运行时间为1小时。若10:00开始执行该作业，其响应比是 C 。

A.2　　B.1　　C.3　　D.0.5

§3 调度算法

四、优先权法

用于作业调度时，从后备队列中选择一或若干个优先权最高的作业进入主存运行；用于进程调度时，把CPU分配给优先权最高的就绪进程。

— **静态优先数**：创建进程时给定,在进程整个运行期间不变

— **动态优先数**：创建时给定初值，其后可不断调整

例：有一个具有两道作业的批处理系统，作业调度采用短作业优先调度，进程调度采用以优先数为基础的抢占式调度，作业序列如图所示，其中作业优先数即进程优先数，数值越小优先级越高。

(1) 列出各作业进入主存时间及结束时间。

(2) 计算平均周转时间。

§3 调度算法

四、优先权法

作业	到达时间	估计运行时间	优先数
A	10:00	40分钟	5
B	10:20	30分钟	3
C	10:30	50分钟	4
D	10:50	20分钟	6

例：有一个具有两道作业的批处理系统，作业调度采用短作业优先调度，进程调度采用以优先数为基础的抢占式调度，作业序列如图所示，其中作业优先数即进程优先数，数值越小优先级越高。

(1) 列出各作业进入主存时间及结束时间。

(2) 计算平均周转时间。

§3 调度算法

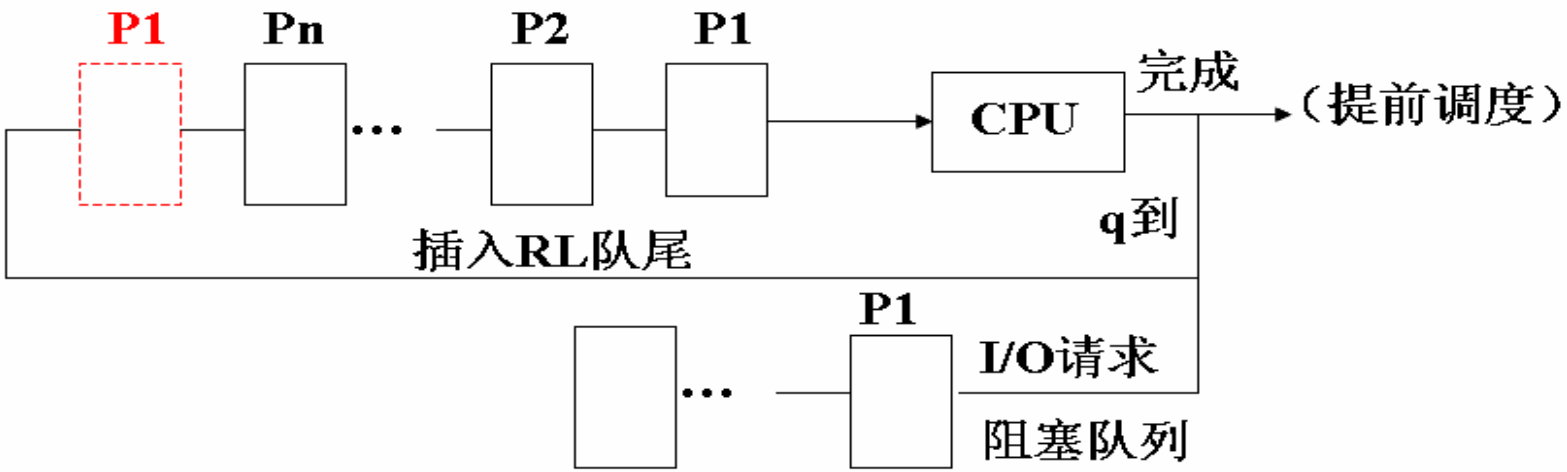
时刻	作业调度事件	进程调度事件	剩余运行时间	结束事件
10:00	选中A	选中A的进程	A: 40min	
10:20	选中B	A下台， B上台	A: 20min B: 30min	
10:30	C后备	B继续	A: 20min B: 20min	
10:50	选中D	选中A的进程	A: 20min D: 20min	B结束
11:10	选中C	选中C的进程	C: 50min D: 20min	A结束
12:00		选中D的进程	D: 20min	C结束
12:20				D结束

作业	到达时刻	运行时间	进入内存时间	结束时间	周转时间
A	10:00	40分钟	10:00	11:10	70min
B	10:20	30分钟	10:20	10:50	30min
C	10:30	50分钟	11:10	12:00	90min
D	10:50	20分钟	10:50	12:20	90min
平均周转时间=(70+30+90+90)/4=70min					

§3 调度算法

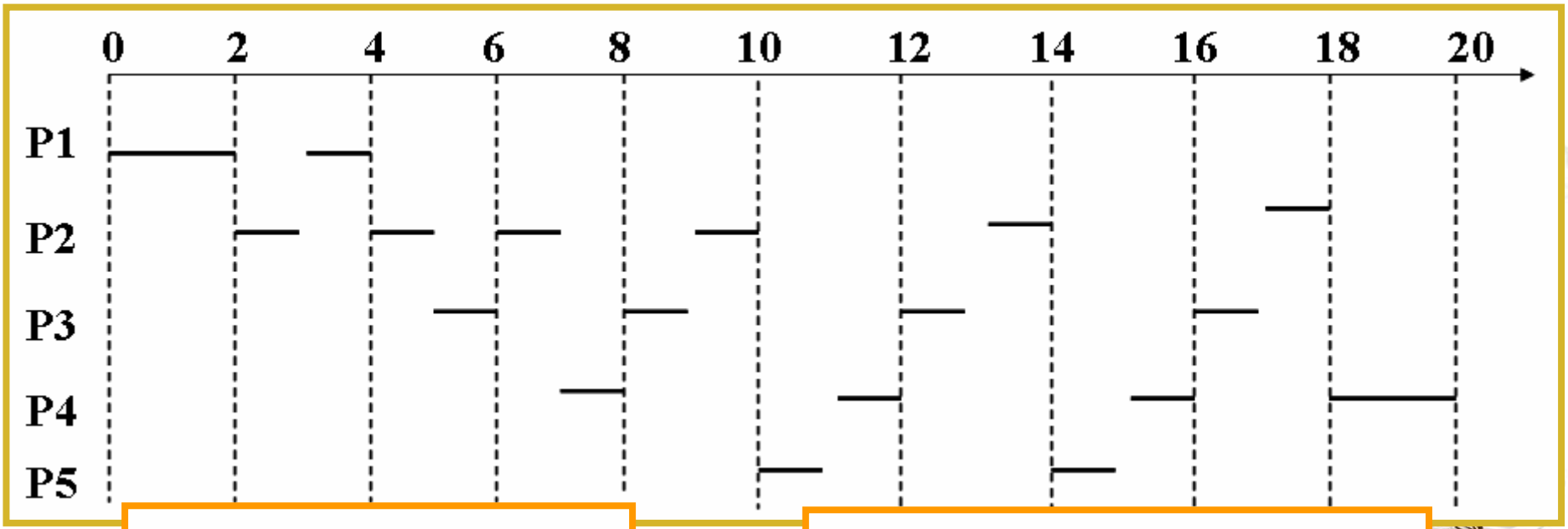
五、轮转法(RR)

- 将所有就绪进程按FCFS原则排列成一个就绪队列
- 将CPU分派给队首进程并使其获得一个时间片q的CPU时间
- 时间片到，将进程链入就绪队列尾部



例1：有5个进程如下图所示,采用RR调度,时间片长度为1个单位。

进程	就绪时刻	服务时间	结束时刻	周转时间	带权周转
P1	0	3			
P2	2	6			
P3	4	4			
P4	6	5			
P5	8	2			

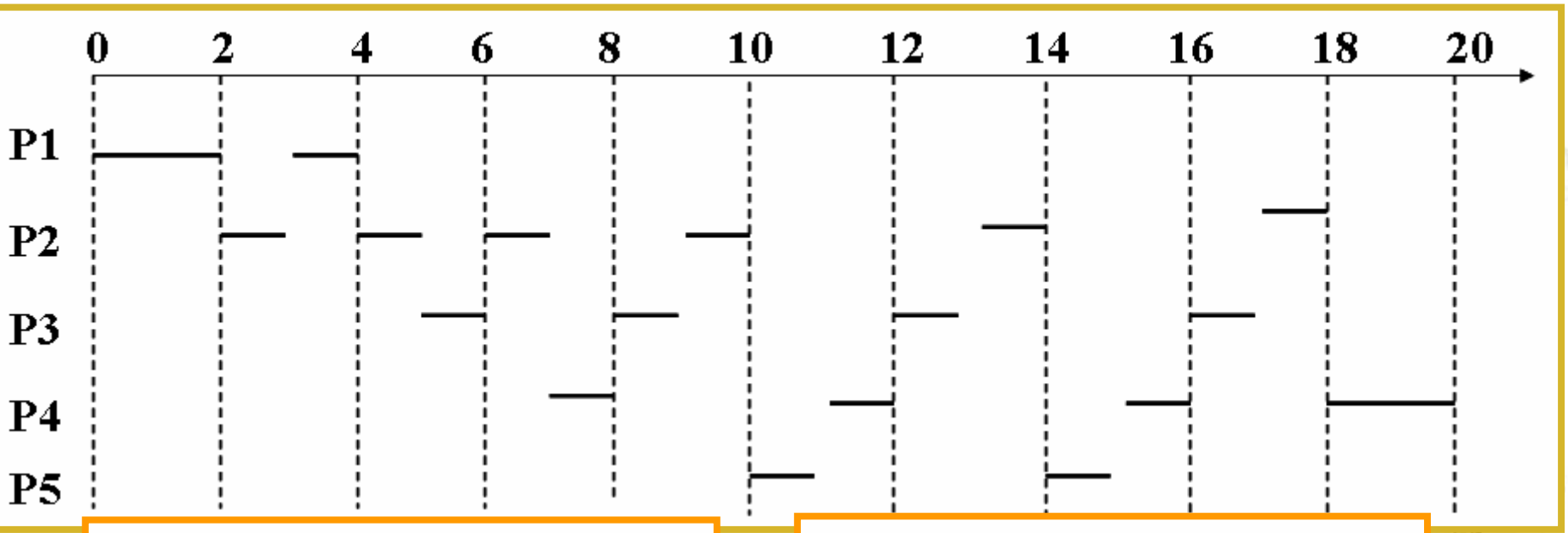


时刻2　P1——P2
(1)　(6)

时刻3　P2——P1
(5)　(1)

例1：有5个进程如下图所示,采用RR调度,时间片长度为1个单位。

进程	就绪时刻	服务时间	结束时刻	周转时间	带权周转
P1	0	3			
P2	2	6			
P3	4	4			
P4	6	5			
P5	8	2			

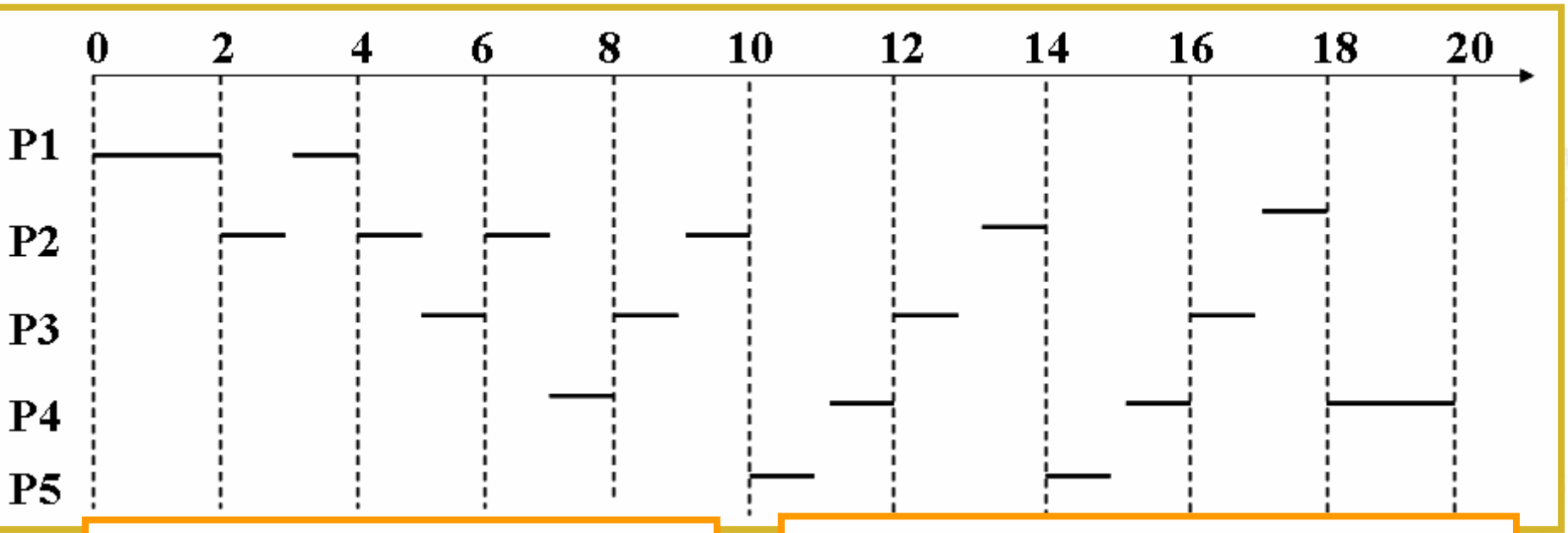


时刻4　P3——P2
(4)　(5)

时刻3　P2——P1
(5)　(1)

例1：有5个进程如下图所示,采用RR调度,时间片长度为1个单位。

进程	就绪时刻	服务时间	结束时刻	周转时间	带权周转
P1	0	3			
P2	2	6			
P3	4	4			
P4	6	5			
P5	8	2			

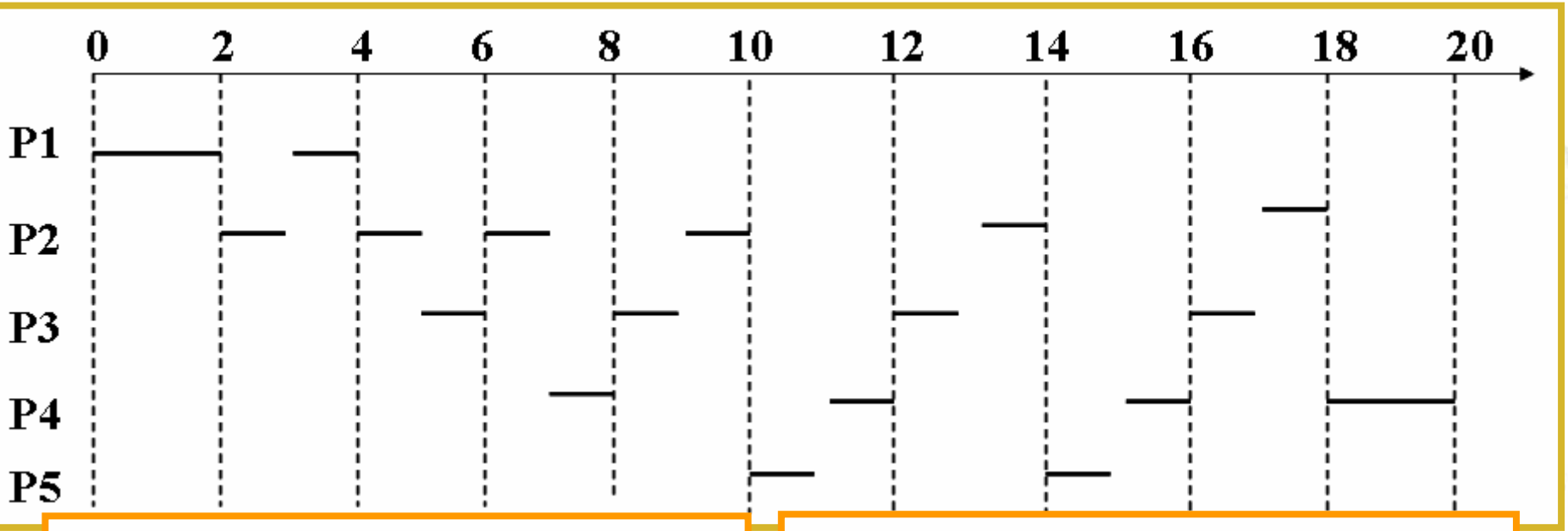


时刻4　P3——P2
(4)　(5)

时刻6　P3——P4——P2
(3)　(5)　(4)

例1：有5个进程如下图所示,采用RR调度,时间片长度为1个单位。

进程	就绪时刻	服务时间	结束时刻	周转时间	带权周转
P1	0	3			
P2	2	6			
P3	4	4			
P4	6	5			
P5	8	2			

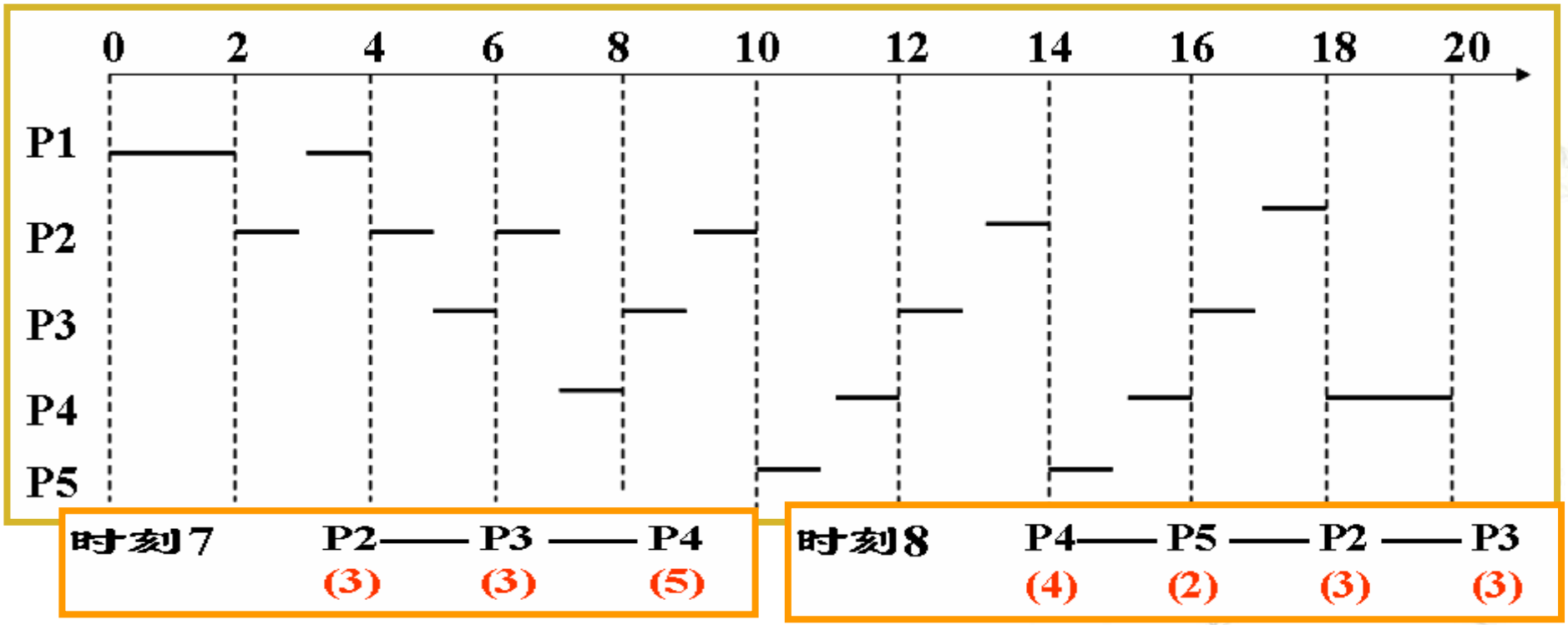


时刻7　P2——(3)　P3——(3)　P4(5)

时刻6　P3——(3)　P4——(5)　P2(4)

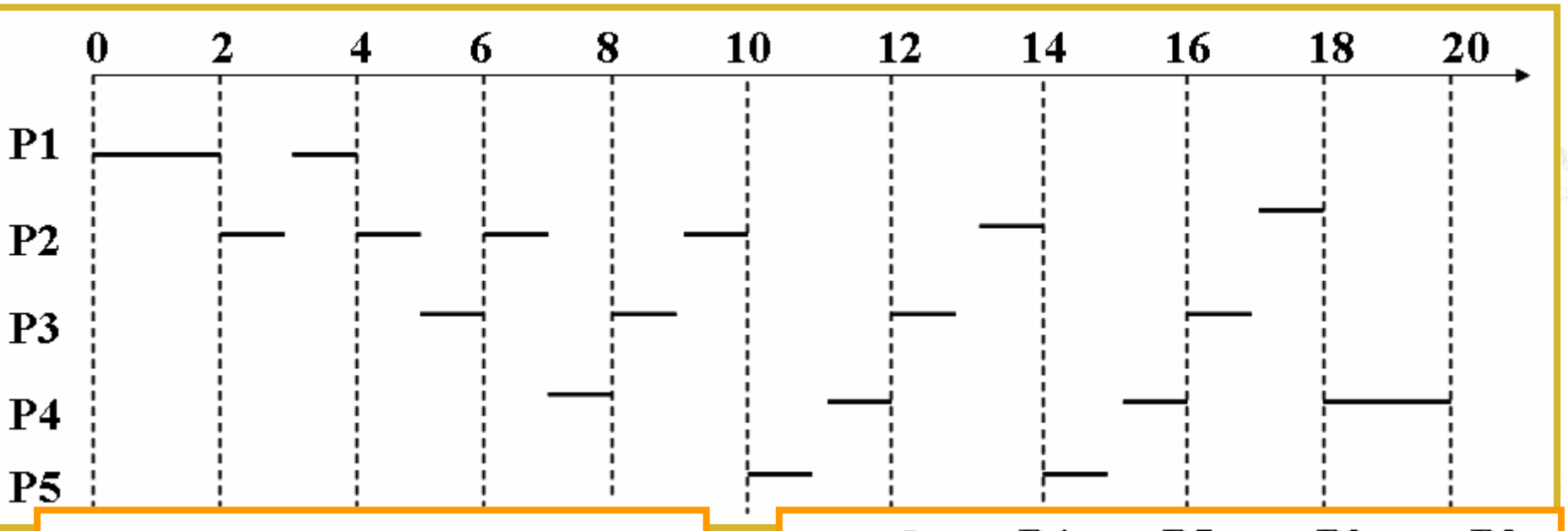
例1：有5个进程如下图所示,采用RR调度,时间片长度为1个单位。

进程	就绪时刻	服务时间	结束时刻	周转时间	带权周转
P1	0	3			
P2	2	6			
P3	4	4			
P4	6	5			
P5	8	2			



例1：有5个进程如下图所示,采用RR调度,时间片长度为1个单位。

进程	就绪时刻	服务时间	结束时刻	周转时间	带权周转
P1	0	3			
P2	2	6			
P3	4	4			
P4	6	5			
P5	8	2			

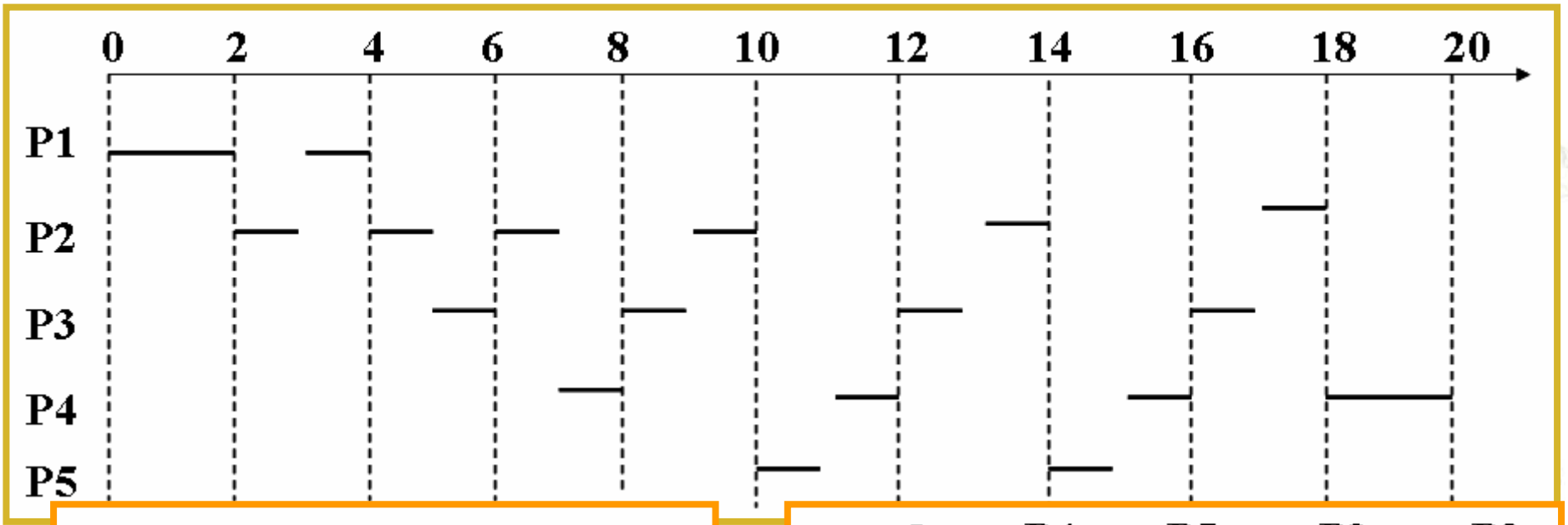


时刻16　P4——P2——P3
(2)　(1)　(1)

时刻8　P4——P5——P2——P3
(4)　(2)　(3)　(3)

例1：有5个进程如下图所示,采用RR调度,时间片长度为1个单位。

进程	就绪时刻	服务时间	结束时刻	周转时间	带权周转
P1	0	3	4	4-0=4	4/3=1.33
P2	2	6	18	18-2=16	16/6=2.67
P3	4	4	17	17-4=13	13/4=3.25
P4	6	5	20	20-6=14	14/5=2.80
P5	8	2	15	15-8=7	7/2=3.50



§3 调度算法

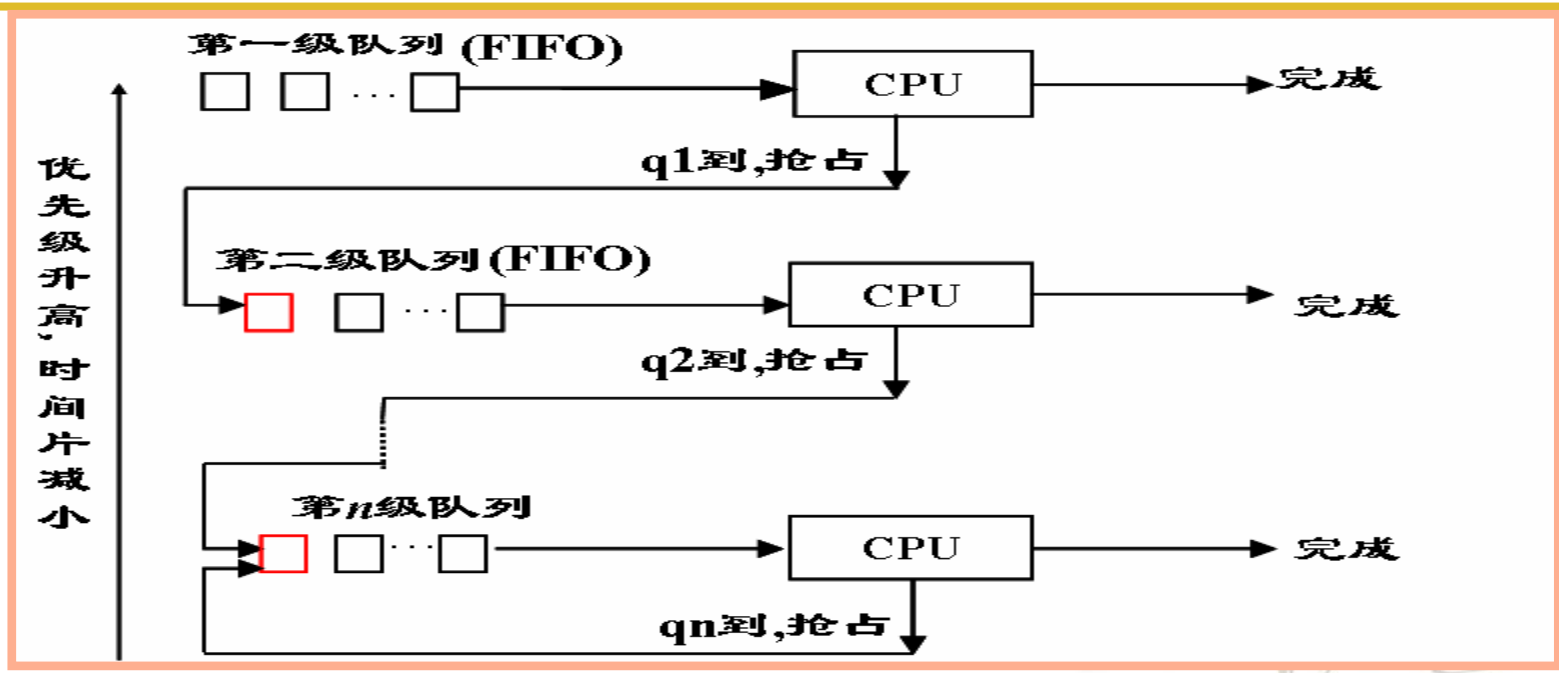
例2：（1）进程调度算法采用固定时间片轮转法，时间片过大，就会使轮转法转化为 FCFS 调度算法。

（2）判断：进程调度采用时间片轮转法，且就绪队列按优先权法建立，就可使得进程正比优先数向前推进。（ F ）

§3 调度算法

六、多级反馈队列调度

- 将就绪进程按优先权不同链接成多个RL。
- 规定各队列进程获得的时间片反比于优先权。
- 调度时机到来，调度最高优先级、非空队列队首进程，只有高级队列为空(完成或阻塞)，才调度低一级队列中进程。
- 当时间片到，强迫进程释放CPU并反馈至下一级队列。



§3 调度算法

六、多级反馈队列调度

- 将就绪进程按优先权不同链接成多个RL。
- 规定各队列进程获得的时间片反比于优先权。
- 调度时机到来，调度最高优先级、非空队列队首进程，只有高级队列为空(完成或阻塞)，才调度低一级队列中进程。
- 当时间片到，强迫进程释放CPU并反馈至下一级队列。

特点与效果

- 对于交互式短进程，在高级别队列上运行，可获满意响应时间
- I/O型和计算型进程，可设置在不同q的队列中，避免无谓CPU切换
- 进程越长，优先级下降，保证了系统吞吐量
- 可改善批量型大型作业的进程迟迟得不到处理的现象

§3 调度算法

例1:下列选项中,降低进程优先权级的合理时机是 A 。

A.进程的时间片用完　　B.进程刚完成I/O，进入就绪队列

C. 进程长期处于就绪队列　D.进程从就绪状态转为运行状态

例2：下列选项中,满足短任务优先且不会发生饥饿现象的调度算法是 B 。

A.先来先服务　　B.高响应比优先

C. 时间片轮转　　D.非抢占式短任务优先

例3：在多道批处理系统中，为充分利用各种资源，作业调度程序应选择的作业类型是： D 。

A. 适应于内存分配的　　B.计算量大的

C. I/O量大的　　D. 计算型和I/O型搭配的

§3 调度算法

例4 ：在单道批处理系统中，有5个待运行作业，它们的估计运行时间分别是9,6,3,5和x。当3<x<5时，采用那种次序运行各作业能得到最短的平均周转时间 3,x,5,6,9 ？平均周转时间是多少 (51+4*x)/5 ？

【分析】在各类作业调度算法中，SJF可获得最短平均周转时间。因此有：

当0<x<3时，运行次序应为：x,3,5,6,9

当3<x<5时，运行次序应为：3,x,5,6,9

当5<x<6时，运行次序应为：3,5, x,6,9

当6<x<9时，运行次序应为：3,5,6, x,9

当x>9时，运行次序应为：3,5,6,9,x

平均周转时间=[3+(3+x)+(3+x+5)+(3+x+5+6)+(3+x+5+6+9)]/5
=(51+4*x)/5

§4 死锁的充要条件

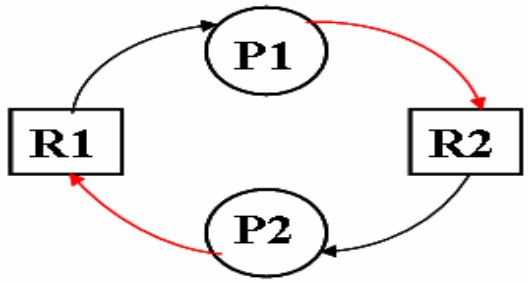
一、死锁产生的原因

死锁定义： 如果一个进程集合中的每个进程都在等待只能由该集合中其它进程才能引发的事件，而无限期陷入僵持的局面，此时的系统状态称为死锁状态。这组进程称为死锁进程，它们所占据的资源称为卷入死锁的资源。

原因
(充分条件)

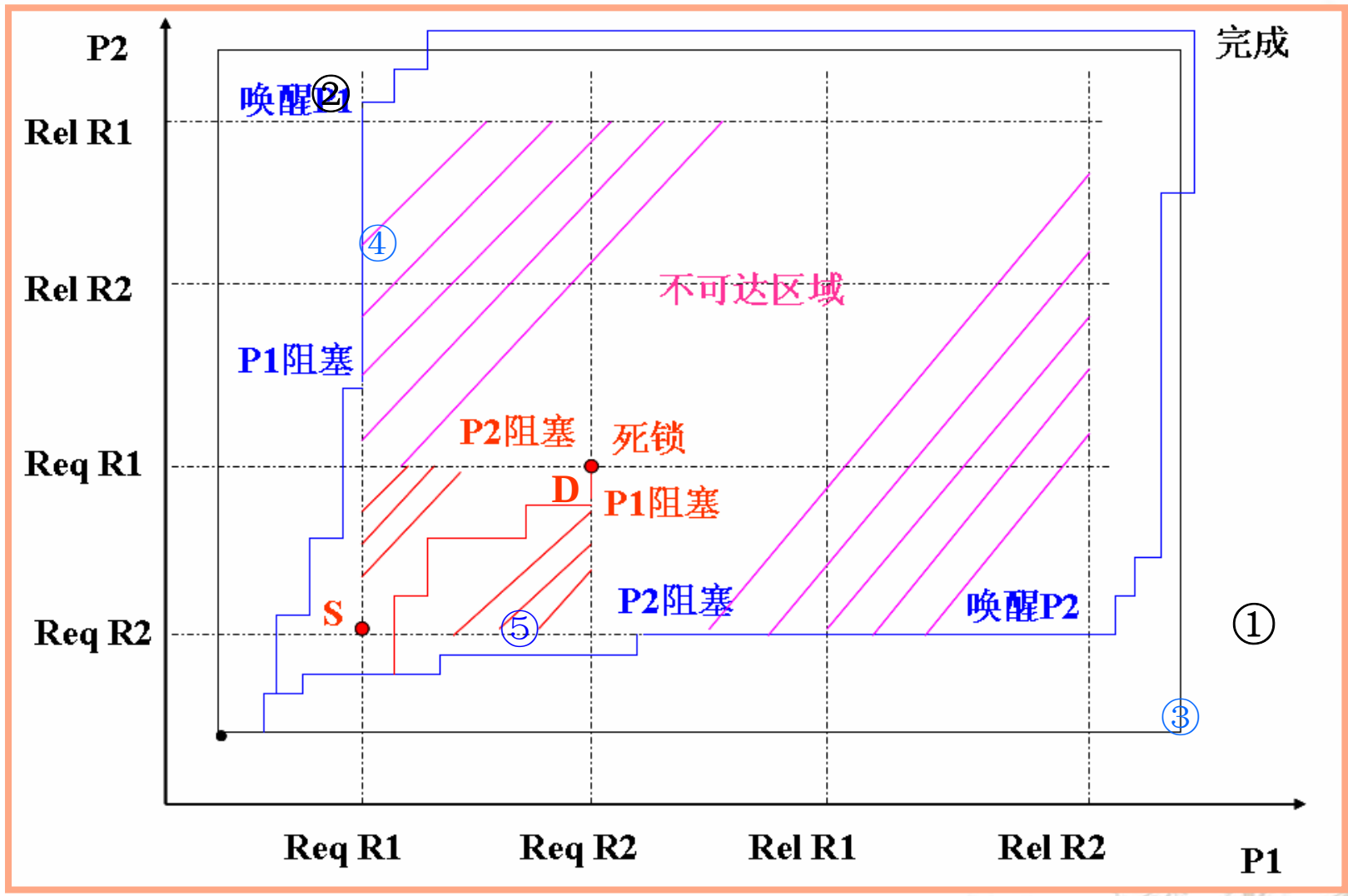
- 并发进程共享数量不足的资源
- 进程推进顺序不当

例：系统中有并发进程P1、P2，它们共享打印机R1一台，读卡机R2一台。试模拟系统并发情况。



go

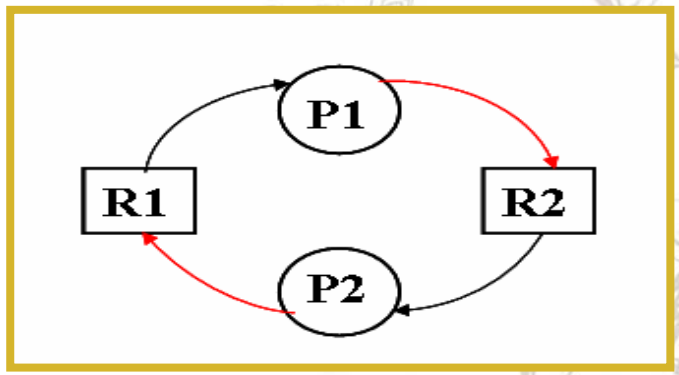
§4 死锁的充要条件


P2
Rel R1
Rel R2
Req R1
Req R2
完成
唤醒P1
②
④
不可达区域
P1阻塞
P2阻塞
死锁
D
P1阻塞
S
⑤
P2阻塞
唤醒P2
①
③
Req R1
Req R2
Rel R1
Rel R2
P1

§4 死锁的充要条件

二、死锁必要条件

1. **互斥条件:** 系统中存在临界资源,进程应互斥使用
2. **请求和保持条件:** 进程因请求资源得不到而阻塞时，保持已占有资源不予释放
3. **不剥夺条件:** 已占用资源只能由属主自行释放，不能由系统予以剥夺
4. **环路条件:** 环中任一进程均等待环中上一进程所占有的资源，形成循环等待



§4 死锁的充要条件

二、死锁必要条件

1. **互斥条件**:系统中存在临界资源,进程应互斥使用
2. **请求和保持条件**：进程因请求资源得不到而阻塞时，保持已占有资源不予释放
3. **不剥夺条件**：已占用资源只能由属主自行释放，不能由系统予以剥夺
4. **环路条件**：环中任一进程均等待环中上一进程所占有的资源，形成循环等待

死锁对策
- **死锁预防**:通过破坏必要条件从而杜绝死锁发生。
- **死锁避免**:避免进入不安全状态从而避免死锁。
- **死锁检测**:通过检测算法判定死锁是否存在。
- **死锁解除**:将系统从死锁态回复至死锁前的某个状态。

§4 死锁的充要条件

例1：①当系统中可用资源不满足当前进程的资源需求时，就一定会产生死锁(F)。

② 死锁是指系统中的全部进程都处于阻塞状态(F)。

③在用P、V操作解决进程同步和互斥时，一定要正确安排P和V操作的顺序，否则会引起死锁(T)。

例2：在 C 情况下，系统出现死锁。

A.进程进入临界区 B. 有多个封锁的进程同时存在

C. 进程因竞争资源而无休止地相互等待对方释放已占有资源

D. 资源数大大小于进程数或进程同时申请的资源数大大超过资源总数

§4 死锁的充要条件

例3：什么是饥饿？死锁与饥饿的主要区别是什么？

死锁指进程竞争数量不足的CR，在推进顺序不当时所导致的一种与时间有关的错误。**体现为：**多个进程无限期地等待永不发生的条件，如无外力干预均不能向前推进。当死锁发生时，一定有资源被无限期地占用而得不到释放。

饥饿(饿死)并不表示系统死锁，资源也都能在有限时间内被释放，但仍存在申请者无限期得不到资源、其推进被无限期延迟的现象：如短进程优先调度算法中，如果不断有短进程进入系统，将导致长进程因得不到CPU而处于饥饿状态。

主要区别：①处于饥饿状态的进程可以只有一个，而处于死锁状态的进程却必须大于或等于两个。

②处于饥饿状态的可能是作业、就绪进程或阻塞进程，而处于死锁状态的必定是阻塞进程。

§5 死锁预防

——破坏必要条件法

1.破坏互斥条件 —— 摒弃互斥条件，不可行！

——若不能保证CR的互斥性，可能导致与时间有关的错误

2.破坏不剥夺条件 —— 摒弃不剥夺条件，不可行！

- 一个进程在申请新的资源不能满足而变为阻塞状态之前，必须释放已占有的全部资源，以后需要时再重新申请

——可能造成前后两次信息不连贯

——为保护放弃资源时的现场及以后再恢复，需付出很大代价

——资源反复申请与释放，系统开销增大

§5 死锁预防

3.破坏请求和保持条件 — 采用静态资源分配法，可行！

- 创建进程时审核并分配进程所需的全部资源
  - —— 进程并发过程中不会再请求资源,系统死锁无关
  - ——降低了资源利用率低,降低了进程并发度
  - ——不允许进程动态提出新的资源需求,是静态资源分配法

4.破坏环路条件 —采用动态资源分配法，可行！

- 将各类资源排列成线性序号，规定进程对资源的请求必须按序号递增次序进行
  - ——任何时刻总有一个占据最高序号的进程可推进
  - ——实现了资源的动态分配，改善了资源利用率
  - ——按序号，可能造成部分资源闲置，也增加了编程困难
  - ——序号的相对稳定性，使增设新设备不便

输入机＝1
磁带机＝2
磁盘机＝3
打印机＝4
穿孔机＝5
：

§5 死锁预防

例1：一进程在获得资源后，只能在使用完资源后由自己释放，这属于死锁必要条件的 C 。

A.互斥条件 B.请求和释放条件

C.不剥夺条件 D.环路等待条件

例2：破坏死锁4个必要条件中的请求和保持条件可用 B 方法。

A.Spooling B.资源静态分配

C.资源动态分配 D.撤销进程

例3：当系统采用资源有序分配方法预防死锁时，它破坏了产生死锁的必要条件中的 环路 条件。

§6 死锁避免

—避免进入不安全状态

一. 安全与不安全状态

设S为系统某个资源请求与分配的状态,该状态并发着n个进程：

对于状态S，若至少存在一种并发序列<$P_{i1},P_{i2}...P_{in}$>，按照这种序列 n个进程均能推进完成，则称S态为安全状态，<$P_{i1},P_{i2}...P_{in}$>为安全序列。

对于状态S，若找不到任何安全序列，则称S为不安全状态。

go

§6 死锁避免

—避免进入不安全状态

例1：系统中有12台磁带机，进程P1共要求10台，P2、P3分别要求4台和9台，某时刻系统进入S1态，试判别S1状态是否是安全态。

S1→S2

	最大需求	已分配	尚需	可用
P1	10	5	5	3→1
P2	4	2	2	
P3	9	2→4	7→5	

∵在S1态存在着安全序列<P2,P1,P3>，∴S1态是安全状态。

例2：在S1态，P3再请求2个单位的磁带机并给予分配，系统由S1态转入S2态，试判别S2是否是安全态。

∵在S2态找不到安全序列，∴S2态是不安全状态。

Back

§6 死锁避免

二、 银行家算法

设m为系统中资源种类， n为系统中并发进程数：

1. 数据结构

- 可用资源向量Available(m):某时刻系统各类资源的空闲数目
- 最大需求矩阵Max(n,m):表示各进程生命期中对各类资源的最大需求数量
- 分配矩阵Allocation(n,m):表示各进程当前已经获得的各类资源数量
- 需求矩阵Need(n,m):表示各进程推进完成对各类资源尚需求数量

§6 死锁避免

二、 银行家算法

2. 算法思想

设S_0状态下， P_i发出资源请求向量:$Request_i(r_1,r_2,...r_m)$;

- 若$Request_i>Need_i$ —— 出错
- 若$Request_i>Available$ —— 阻塞
- 预分配资源， 系统由S_0态进入S_1态
- 执行安全性算法(safe),判别S_1态安全性:若安全,正式分配资源;否则,取消预分配,阻塞P_i进程,恢复至原来状态

$Available=Available-Request_i$

$Allocation_i=Allocation_i+Request_i$

$Need_i=Need_i-Request_i$

§6 死锁避免

二、 银行家算法

3. safe算法

- 工作向量work(m)

— 系统可提供给进程的各类资源数量， $work_{初}$=Available

- 完成向量Finish(n)

— 表示n个进程能否推进完成， $Finish_{初}$ = 全假

- 从进程集合中找出P_i，其满足Finish(i)=false&$Need_i \leqslant work$
- 假定P_i完成,则有： Finish(i)=true且$work=work+Allocation_i$
- 若有限步后Finish全真，则S为安全态，否则为不安全态。

§6 死锁避免

例：设系统中有{P0,P1,P2,P3,P4}进程及资源{A,B,C}，初启时有Available=(10,5,7),T0时刻资源分配如下图所示:(1)试判T_0时刻的安全性(2)设P_1发出Request(1,0,2)能否分配？(3)设P_4发出Request(3,3,0)能否分配？(4)设P_0发出Request(0,2,0)能否分配？

	Max			Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C	A	B	C
P_0	7	5	3	0	1	0	7	4	3	3	3	2
P_1	3	2	2	2	0	0	1	2	2			
P_2	9	0	2	3	0	2	6	0	0			
P_3	2	2	2	2	1	1	0	1	1			
P_4	4	3	3	0	0	2	4	3	1			

§6 死锁避免

例：设系统中有{P0,P1,P2,P3,P4}进程及资源{A,B,C}，初启时有Available=(10,5,7),T0时刻资源分配如下图所示:(1)试判T_0时刻的安全性(2)设P_1发出Request(1,0,2)能否分配？(3)设P_4发出Request(3,3,0)能否分配？(4)设P_0发出Request(0,2,0)能否分配？

	Max			Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C	A	B	C
P_0	7	5	3	0	1	0	7	4	3	2	3	0
P_1	3	2	2	3	0	2	0	2	0			
P_2	9	0	2	3	0	2	6	0	0			
P_3	2	2	2	2	1	1	0	1	1			
P_4	4	3	3	0	0	2	4	3	1			

§6 死锁避免

例：设系统中有{P0,P1,P2,P3,P4}进程及资源{A,B,C}，初启时有Available=(10,5,7),T0时刻资源分配如下图所示:(1)试判T_0时刻的安全性(2)设P_1发出Request(1,0,2)能否分配？(3)设P_4发出Request(3,3,0)能否分配？(4)设P_0发出Request(0,2,0)能否分配？

	Max			Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C	A	B	C
P_0	7	5	3	0	3	0	7	2	3	2	1	0
P_1	3	2	2	3	0	2	0	2	0			
P_2	9	0	2	3	0	2	6	0	0			
P_3	2	2	2	2	1	1	0	1	1			
P_4	4	3	3	0	0	2	4	3	1			

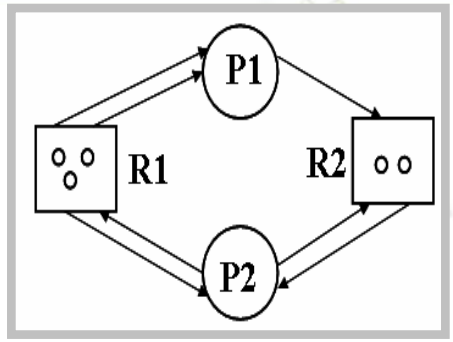
§6 死锁避免

4. 银行家算法特点

- 系统一直工作在安全状态；
- 算法允许互斥条件、请求和保持条件、不剥夺条件成立，相比于死锁预防对策，其对资源的限制条件放松，资源利用率提高；
- 算法要求各类资源数量固定不变，但资源可能损坏；
- 算法要求进程数固定不变，这在多道系统中是难以做到的；
- 算法保证用户要求在有限时间内得到响应，但实时用户要求更好的响应速度；
- 算法要求用户预先说明最大资源需求量，对用户不便甚至困难。

§7 死锁检测

一、资源分配图



是一组结点集N和有向边集E构成的对偶<N,E>：

- 用圆形结点P={P_1,P_2,...P_n}表示进程结点；
- 用方形结点R={R_1,R_2,...R_m}表示资源结点，用小圆圈表示资源实例；

则有向图中N=PUR

- E中任一边e∈E都链接着P中和R中一个结点：
- 若e=(P_i,R_j) —表示P_i申请一个单位的R_j资源；
- 若e=(R_j,P_i) —表示P_i获得一个单位的R_j资源；
- 若R_j资源有w_j个 (w_j≥1),则分配给P_i的R_j资源数目用 |(R_j,P_i)|表示;进程P_i申请R_j资源的数目用|(P_i,R_j)|表示。

§7 死锁检测

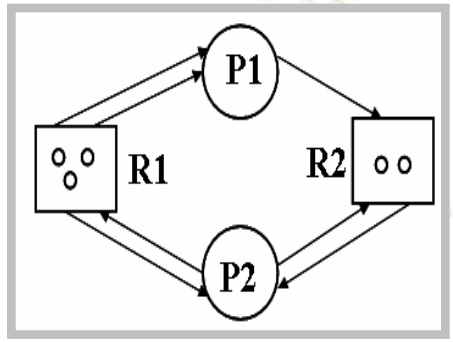
二、资源分配图化简

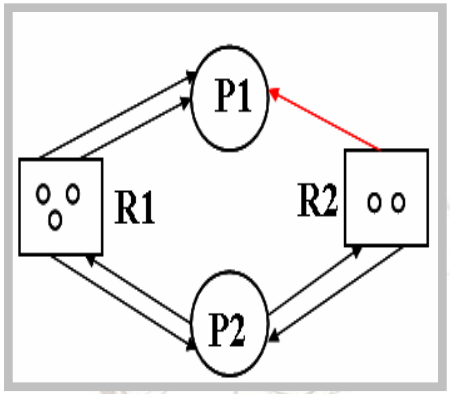
(1) 从进程结点集中找到既非孤立又非阻塞的结点P_i;

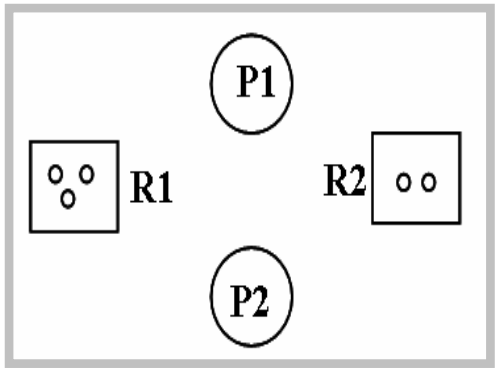
(2) 假设顺利，则P_i推进完成成为孤立结点;

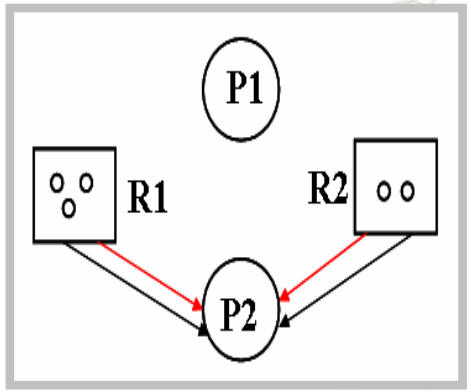
(3) P_i释放的资源使得P_j变为孤立结点;

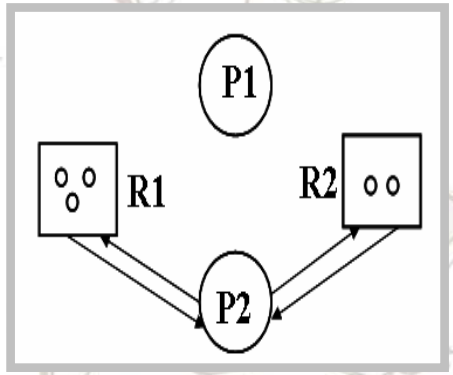
(4) 若一系列化简步骤后能消去图中所有边，则称该图是可完全化简的；若有向图不能通过任何进程予以化简，则称该图是不可完全化简的











§7 死锁检测

二、资源分配图化简

(1) 从进程结点集中找到既非孤立又非阻塞的结点P_i;

(2) 假设顺利，则P_i推进完成成为孤立结点;

(3) P_i释放的资源使得P_j变为孤立结点;

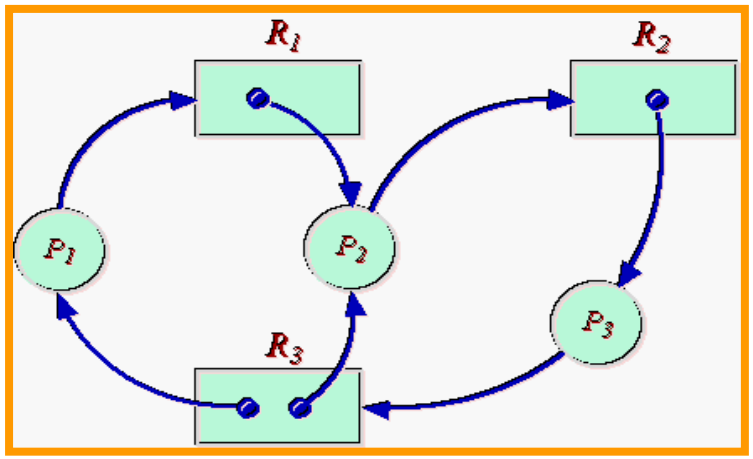
(4) 若一系列化简步骤后能消去图中所有边，则称该图是可完全化简的；若有向图不能通过任何进程予以化简，则称该图是不可完全化简的

死锁定理： S状态为死锁状态的充分条件是——当且仅当S状态的资源分配图是不可完全化简的。

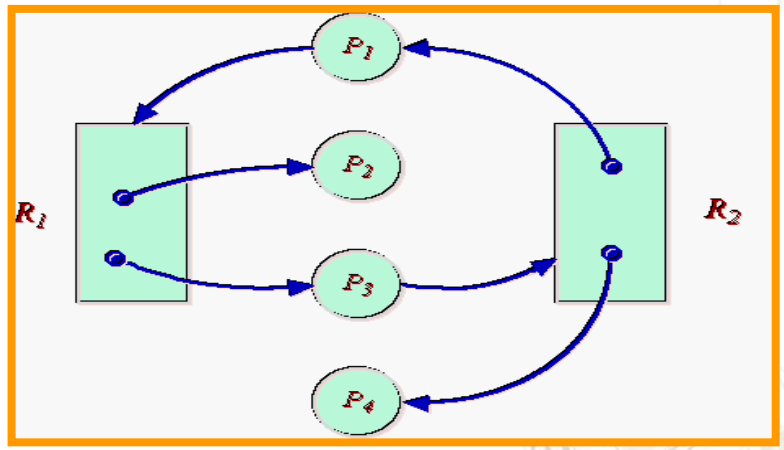
引理： 一个给定的资源分配图的全部化简序列导致同一不可化简图。

§7 死锁检测

例1：当资源分配图中出现环路时,系统一定发生了死锁。（ F ）



有环路有死锁



有环路无死锁

环路是死锁的必要条件，而非死锁的充分条件
当资源实例均为1时，环路是死锁的充分必要条件

例2:某系统中有11台打印机，N个进程共享，每个进程要求3台。当N的取值不超过 5 时，系统不会死锁。

§7 死锁检测

例3:系统中仅有一个资源类，其中共有3个资源实例，使用此类资源的进程共有3个，每个进程至少请求1个资源，它们所需资源最大量的总和为x，则发生死锁的必要条件是 x≥6 。

例4:3个进程共享4个同类资源,这些资源的分配与释放只能一次一个。已知每一个进程最多需要两个该类资源,则该系统 D 。

A.有某进程可能永远得不到该类资源　B.必然有死锁

C.进程请求该类资源立刻能得到　D.必然无死锁

【公式】设m为某类资源数，n为共享该资源的并发进程数，k为每个进程对该类资源的需求数目

死锁无关：$m-n\times(k-1)\geqslant 1$

例5:某计算机系统有8台打印机,k个进程竞争使用,每个进程最多需要3台打印机。该系统可能会发生死锁的k的最小值是 C 。

A. 2　B. 3　C. 4　D.5

§7 死锁检测

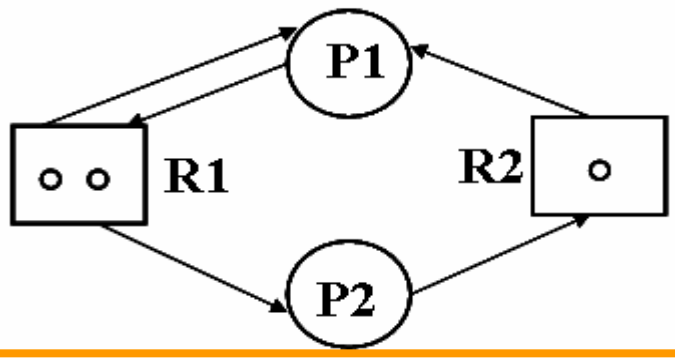
例6：假定某系统有R1和R2两类可再使用资源（其中R1有两个单位，R2有一个单位），它们被进程P1和P2所共享，且已知两个进程均以下列顺序使用两类资源：

→申请R1→申请R2→申请R1→释放R1→释放R2→释放R1 →

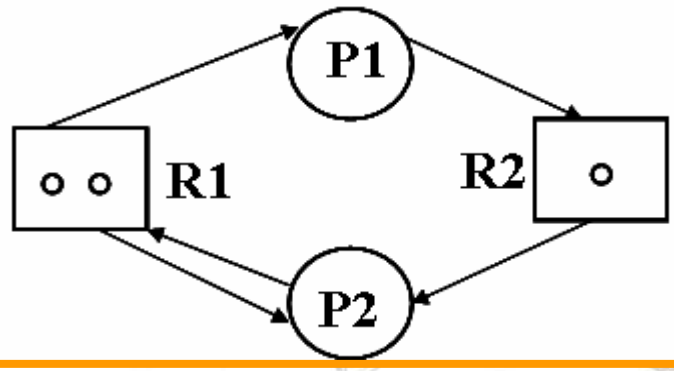
试求出系统运行过程中可能到达的死锁点，并画出死锁点的资源分配图。

当P1、P2均执行完第1步资源请求,即都申请到1个R1资源时,系统进入不安全状态。随着P1或P2任一申请到R2资源时,系统资源分配完毕。随后P1和P2的任何资源请求都无法满足，系统进入死锁状态。

①P1获得R1、R2，P2获得R1



②P2获得R1、R2，P1获得R1



§8 死锁解除

1.撤消进程法 —— 释放资源法

- 终止所有进程重启OS —— 鸵鸟算法
- 终止所有死锁进程
- 一次终止一个死锁进程直至死锁解除

2.挂起进程法 ——资源剥夺法

例：对待死锁，一般应考虑死锁的预防、避免、检测和解除四个问题。典型的银行家算法是属于 死锁避免，破坏环路条件是属于 死锁预防，而剥夺资源是属于 死锁解除的基本方法。其中，在银行家算法中，允许进程动态地申请资源，但系统在进行资源分配时，应先计算资源分配的 安全性；若此次分配不会导致系统进入 不安全状态，便将资源分配给它，否则进程 阻塞。