操作系统
第十二讲

张涛

Review

产生死锁的原因和必要条件

处理死锁的基本方法

死锁的预防

死锁的避免

Today we focus on …

死锁的避免——银行家算法

死锁的检测

死锁的解除

3.7.5 死锁的避免

- 在系统运行过程中，对进程发出的每一个系统能够满足的资源申请进行动态检查，并根据检查结果决定是否分配资源，若分配后系统可能发生死锁，则不予分配，否则予以分配
- 安全状态：
  - 如果存在一个由系统中所有进程构成的安全序列P1，...，Pn，则系统处于安全状态
  - 安全状态一定是没有死锁发生的
- 不安全状态不一定是死锁状态（不安全状态可能导致死锁）；

银行家算法

- **银行家算法**
  - 银行家拥有一笔周转资金
  - 客户要求分期贷款，如果客户能够得到各期贷款，就一定能够归还贷款，否则就一定不能归还贷款
  - 银行家应谨慎的贷款，防止出现坏帐
- **用银行家算法避免死锁**
  - 操作系统（银行家）
  - 操作系统管理的资源(周转资金)
  - 进程（要求贷款的客户）

利用银行家算法避免死锁

- n：系统中进程的总数；m：资源类总数
- 银行家算法中的数据结构
  - (1) 可利用资源向量Available。含有m个元素的数组，其中的每一个元素代表一类可利用的资源数目，其初始值是系统中所配置的该类全部可用资源的数目，其数值随该类资源的分配和回收而动态地改变。
  - (2) 最大需求矩阵Max。这是一个n×m的矩阵，它定义了系统中n个进程中的每一个进程对m类资源的最大需求。如果Max [i,j] =K，则表示进程i需要Rj类资源的最大数目为K。

- (3) 分配矩阵Allocation。一个n×m的矩阵，它定义了系统中每一类资源当前已分配给每一进程的资源数。如果Allocation［i,j］=K，则表示进程i当前已分得Rj类资源的数目为K。
- (4) 需求矩阵Need。一个n×m的矩阵，用以表示每一个进程尚需的各类资源数。如果Need［i,j］=K，则表示进程i还需要Rj类资源K个，方能完成其任务。

$$Need[i,j] = Max[i,j] - Allocation[i,j]$$

- (5)请求向量Requesti是进程Pi的请求向量，如果Requesti［j］=*K*，表示进程Pi需要*K*个Rj类型的资源。

Example

可利用资源向量Available：

A	B	C
5	2	3

最大需求矩阵Max：

	A	B	C
P1	5	6	2
P2	3	3	1
P3	4	2	5
P4	3	3	2

银行家算法

系统按下述步骤进行检查：

(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］；

(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，将本次的试探分配作废，恢复原来的资源分配状态，让进程P_i等待。

安全性算法

(1) 设置两个向量：

① 工作向量Work: 它表示系统可提供给进程继续运行所需的各类资源数目，它含有m个元素，在执行安全算法开始时，Work：=Available;

② Finish: 它表示系统是否有足够的资源分配给进程，使之运行完成。开始时先做Finish［i］：=false; 当有足够资源分配给进程时，再令Finish［i］：=true。

(2) 从进程集合中找到一个能满足下述条件的进程：

① Finish [i] =false;

② Need [i,j] ≤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，可顺利执行，直至完成，并释放出分配给它的资源，故应执行：

Work [j] ： =Work [i] +Allocation [i,j] ；

Finish [i] ： =true;

go to step 2;

(4) 如果所有进程的Finish [i] =true都满足， 则表示系统处于安全状态；否则，系统处于不安全状态。

一个安全性计算的实例

问：T0时刻是否为安全状态？

T_0 时刻资源分配表

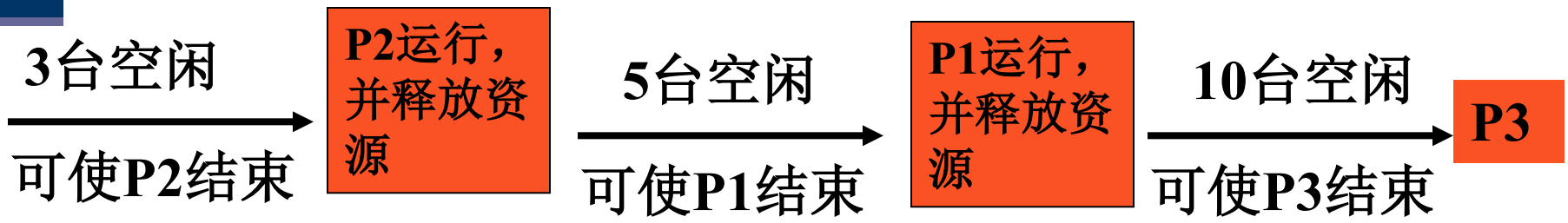
资源情况 / 进程	Claim			Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C	A	B	C
P_0	7	5	3	0	1	0	7	4	3	3	3	2
P_1	3	2	2	2	0	0	1	2	2			
P_2	9	0	2	3	0	2	6	0	0			
P_3	2	2	2	2	1	1	0	1	1			
P_4	4	3	3	0	0	2	4	3	1			

资源情况 / 进程	Work			Need			Allocation			Work+Allocation			Finish
	A	B	C	A	B	C	A	B	C	A	B	C	
P_1	3	3	2	1	2	2	2	0	0	5	3	2	True
P_3	5	3	2	0	1	1	2	1	1	7	4	3	True
P_4	7	4	3	4	3	1	0	0	2	7	4	5	True
P_2	7	4	5	6	0	0	3	0	2	10	4	7	True
P_0	10	4	7	7	4	3	0	1	0	10	5	7	True

银行家算法—举例

- 假定系统有三个进程P1，P2，P3，共有12台磁带机。进程P1总共要求10台磁带机，P2和P3分别要求4台和9台。设在T0时刻进程P1，P2，P3已分别获得5，2，2台，尚有3台空余未分。

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



称 P2，P1，P3 为安全序列

银行家算法—举例

条件同前例，如果：P3提出申请2台资源，问是否可以满足要求？

假设法

解：先假设能够满足要求且分配资源，则系统状态如下：

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

只剩余 1 台资源，不能使任何一个进程执行结束，因而不存在安全序列，所以系统不安全，因此拒绝分配资源，撤消开始的假设。

银行家算法举例

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

	Max			Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C	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			

	Work	Need	Alloc	Work+alloc	Finish
	A B C	A B C	A B C	A B C	
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

T0时刻的安全性检查

- P1发出请求Request(1,0,2)，执行银行家算法

	Max			Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C	A	B	C
p0	7	5	3	0	1	0	7	4	3	3	3	2
p1	3	2	2	2 (3	0 0	0 2)	1 (0	2 2	2 0)	(2	3	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			

	Work A B C	Need A B C	Alloc 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

P1申请资源(1,0,2)时安全性检查(安全)

P1请求资源之后：

P4发出请求Request(3,3,0), 执行银行家算法

Available=2 3 0

不能通过算法第2步（ Request[i]≤Available ），所以P4等待。

P0请求资源：Request（0，2，0），执行银行家算法

	Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C
p0	0	3	0	7	2	3	2	1	0
p1	3	0	2	0	2	0			
p2	3	0	2	6	0	0			
p3	2	1	1	0	1	1			
p4	0	0	2	4	3	1			

为P0分配（0，2，0）后的情况（不安全）

- 练习：有三类资源A(17)、B(5)、C(20)。有5个进程P1—P5。T0时刻系统状态如下：

	最大需求	已分配
P1	5 5 9	2 1 2
P2	5 3 6	4 0 2
P3	4 0 11	4 0 5
P4	4 2 5	2 0 4
P5	4 2 4	3 1 4

问(1)、T0时刻是否为安全状态，若是，给出安全系列。

(2)、T0时刻，P2: Request(0,3,4)，能否分配，为什么？

(3)、在(2)的基础上P4：Request(2,0,1)，能否分配，为什么？

(4)、在(3)的基础上P1：Request(0,2,0)，能否分配，为什么？

解

	最大资源需求M			已分配资源数量U			尚需资源N		
	A	B	C	A	B	C	A	B	C
P1	5	5	9	2	1	2	3	4	7
P2	5	3	6	4	0	2	1	3	4
P3	4	0	11	4	0	5	0	0	6
P4	4	0	5	2	0	4	2	0	1
P5	4	2	4	3	1	4	1	1	0

剩余向量A＝（2，3，3）

1. 是否安全？

（2, 3, 3） → P4 （4, 3, 7） → P5 （7, 4, 11） → P1 （9, 5, 13） → P2 （13, 5, 15） → P3

（2, 0, 4）　（3, 1, 4）　（2, 1, 2）　（4, 0, 2）

安全序列：p4, p5, p1, p2, p3

思考： 安全序列是否唯一？

解：

	最大资源需求M			已分配资源数量U			尚需资源N		
	A	B	C	A	B	C	A	B	C
P1	5	5	9	2	1	2	3	4	7
P2	5	3	6	4	0	2	1	3	4
P3	4	0	11	4	0	5	0	0	6
P4	4	0	5	2	0	4	2	0	1
P5	4	2	4	3	1	4	1	1	0

剩余向量A＝（2，3，3）

2. 在T0时刻若进程P2请求资源（0，3，4），是否能实施资源分配？为什么？

解：R（0，3，4）＞A（2，3，3）

所以不能满足，不能分配

解

	最大资源需求M			已分配资源数量U			尚需资源N		
	A	B	C	A	B	C	A	B	C
P1	5	5	9	2	1	2	3	4	7
P2	5	3	6	4	0	2	1	3	4
P3	4	0	11	4	0	5	0	0	6
P4	4	0	5	2	0	4	2	0	1
P5	4	2	4	3	1	4	1	1	0

剩余向量A＝（2，3，3）

③ 在②的基础上，若进程P4请求资源（2，0，1），是否能实施资源分配？为什么？

解：因为R（2，0，1）<=N4(2, 0, 1) 且＜A（2，3，3）

假设可以满足，则N4=(0, 0, 0), A=（0，3，2），在此基础上，

（0, 3, 2） → P4 （4, 3, 7） → P5 （7, 4, 11） → P1 （9, 5, 13） → P2 （13, 5, 15） → P3

所以是安全的，因此可以实施分配

(4) P1： Request(0,2,0)

	Allocation	Need	Available
P1	2 3 2	3 2 7	0 1 2
P2	4 0 2	1 3 4	
P3	4 0 5	0 0 6	
P4	4 0 5	0 2 0	
P5	3 1 4	1 1 0	

0 1 2 已不能满足任何进程的需要，不能分配

Summary of Banker's Algorithm

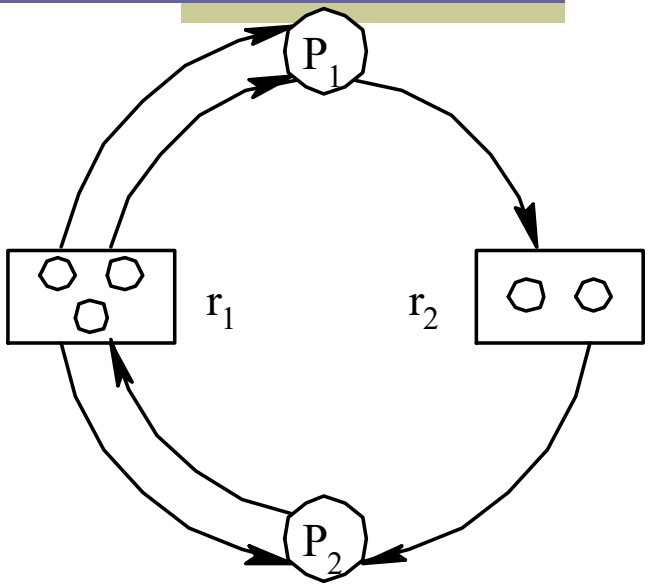
- 优点
  - 比死锁预防限制少
  - 无死锁检测方法中的资源剥夺，进程重启
- 缺点
  - 必须事先声明每个进程请求的最大资源
  - 考虑的进程必须是无关的，也就是说，它们执行的顺序没有任何同步要求的限制
  - 进程的数量保持固定不变，且分配的资源数目必须是固定的
  - 在占有资源时，进程不能退出

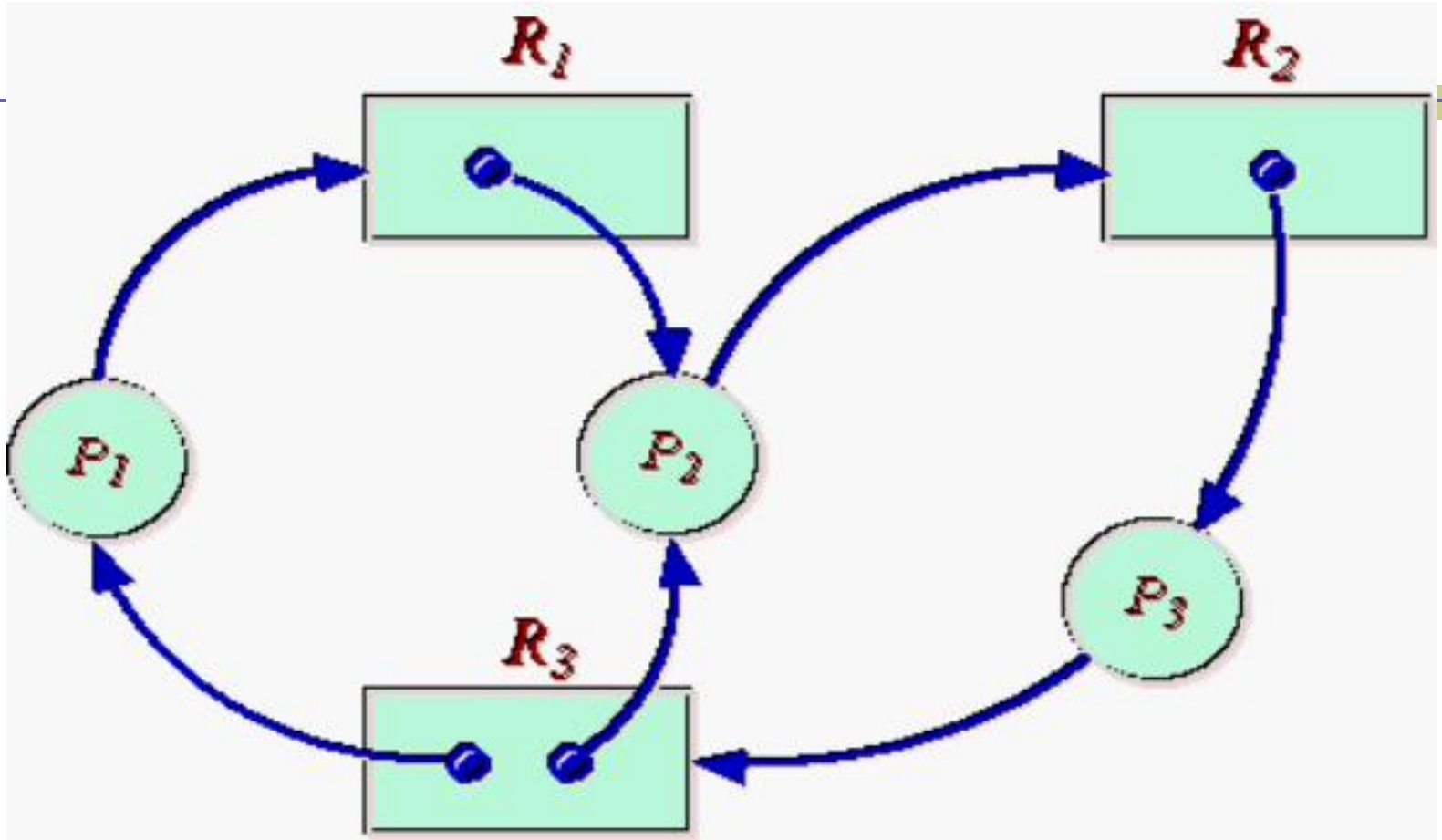
3.7.6 死锁的检测

- 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
- 检测时机：
  - 当进程等待时检测死锁（缺点是系统的开销大）
  - 定时检测
  - 系统资源利用率下降时检测死锁

资源分配图
Resource Allocation Graph

- 二元组G=（V，E）
  - V：结点集，分为P，R两部分
    - P={p1,p2,…,pn}
    - R={r1,r2,…,rm}
  - E：边的集合，其元素为有序二元组
    - (pi,rj)或(rj,pi)
- 表示法
  - 资源类（资源的不同类型），用方框表示
  - 资源实例（每个资源类中），用方框中的黑圆点（圈）表示
  - 进程，用圆圈中加进程名表示
- 分配边：资源实例→进程的一条有向边
- 申请边：进程→资源类的一条有向边




R_1
R_2
P_1
P_2
P_3
R_3

例：系统有两台宽行打印机和一台图形显示器，进程P_1请求一台宽打，则有的资源分配图1。进程P_1分配到一台宽打，并请示一台图形显示器，进程P_2已分到一台图显，并请求一台宽打，则有其分配图2。

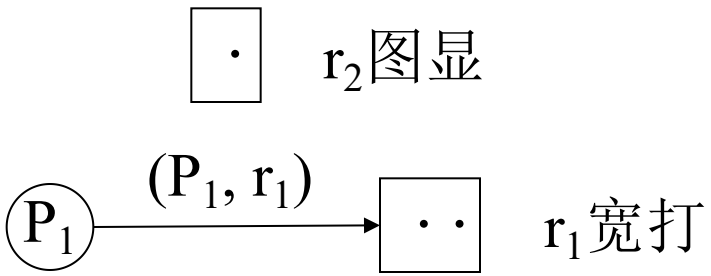


图1

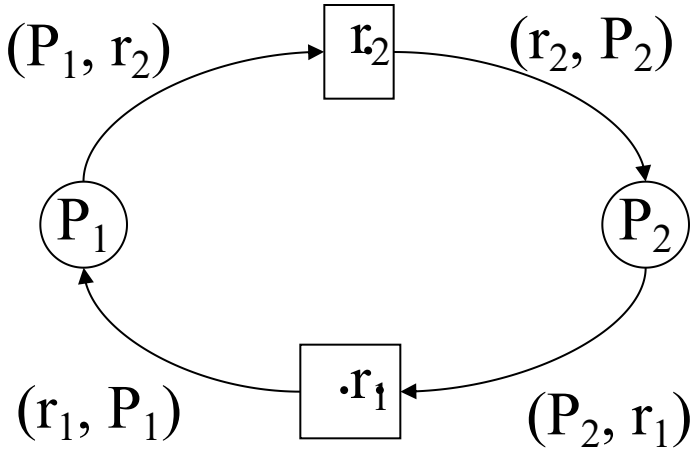


图2

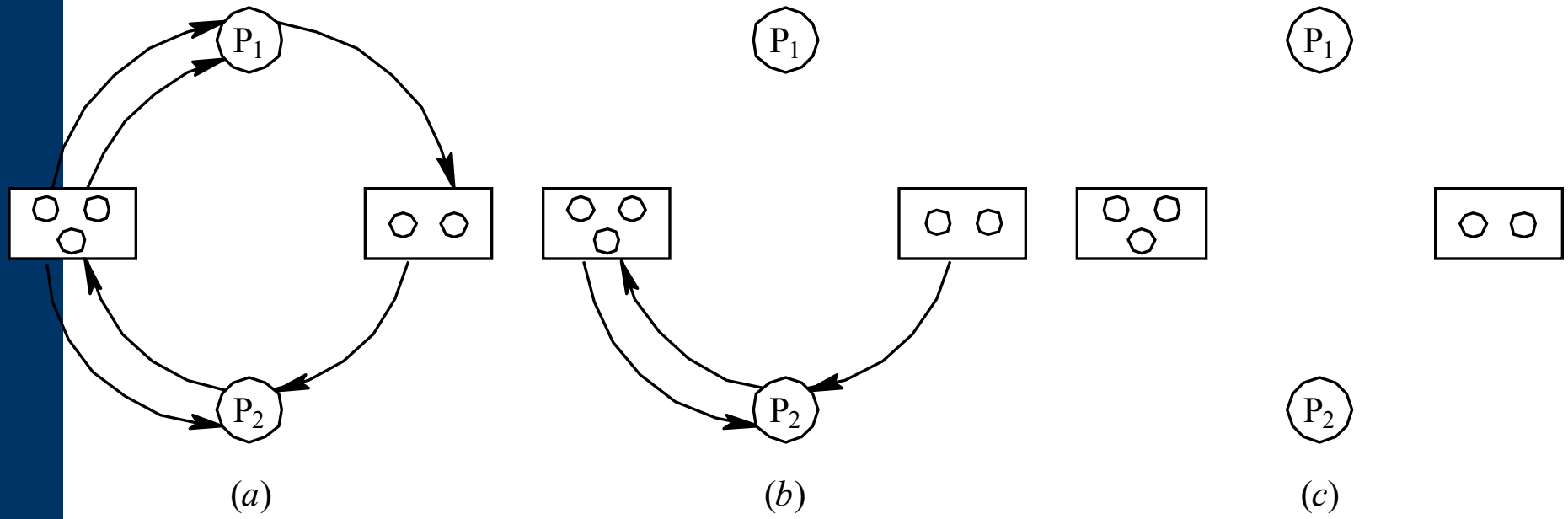
死锁定理

- 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。
- 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。

资源分配图的化简

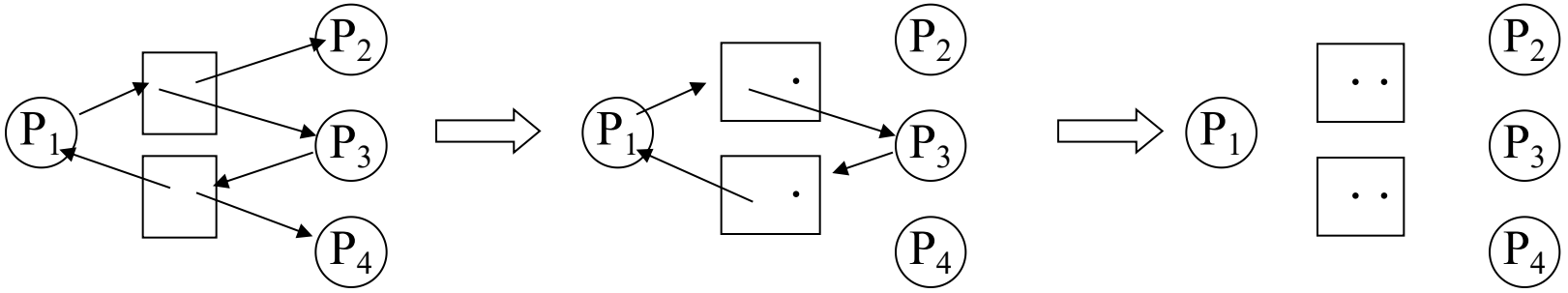
- 1）找一个非孤立点进程结点且只有分配边，去掉分配边，将其变为孤立结点
- 2）再把相应的资源分配给一个等待该资源的进程，即将某进程的申请边变为分配边
- 3）重复以上步骤，若所有进程成为孤立结点，称该图是可完全简化的，否则称该图是不可完全简化的。

- 一个给定的进程-资源图的全部化简序列导致同一不可化简图。
- 死锁状态的充分条件：当且仅当资源分配图是不可完全简化的。

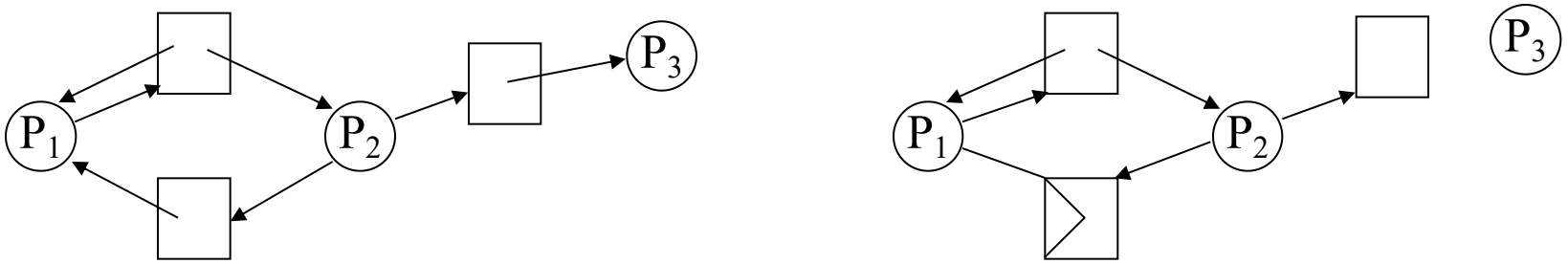


资源分配图的简化

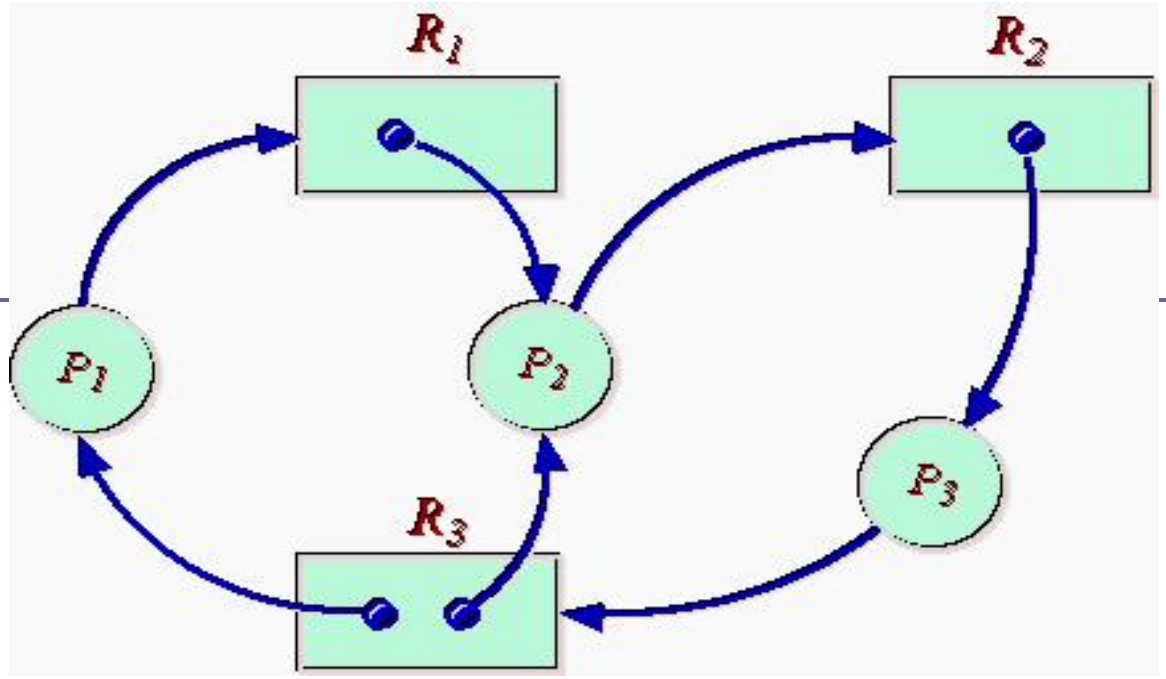
* 如果化简后所有进程顶点都为孤立点，则称该图为可完全化简图 (如图A)，否则称之为不可完全化简的(如图B)。



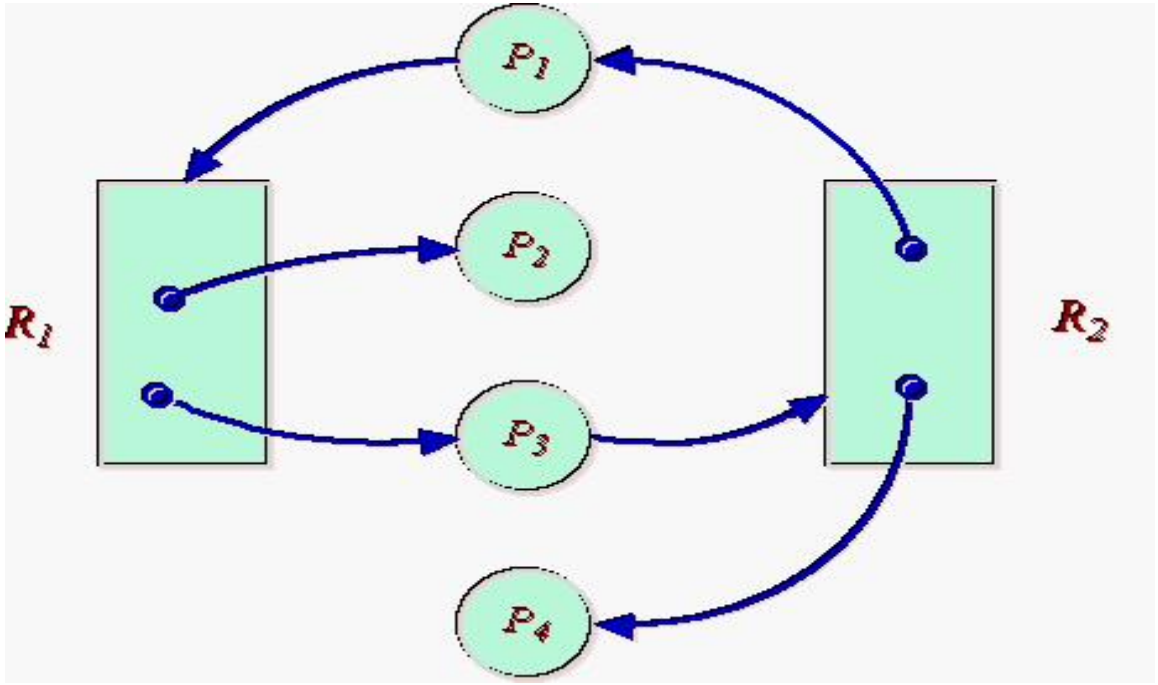
(图A)可完全化简



(图B)不可完全化简



有环有死锁



有环无死锁

死锁检测中的数据结构

(1) 可利用资源向量Available，它表示了m类资源中每一类资源的可用数目。

(2) 把不占用资源的进程(向量Allocation：=0)记入L表中，即$L_i \cup L$。

(3) 从进程集合中找到一个$Request_i \leq Work$的进程，做如下处理：

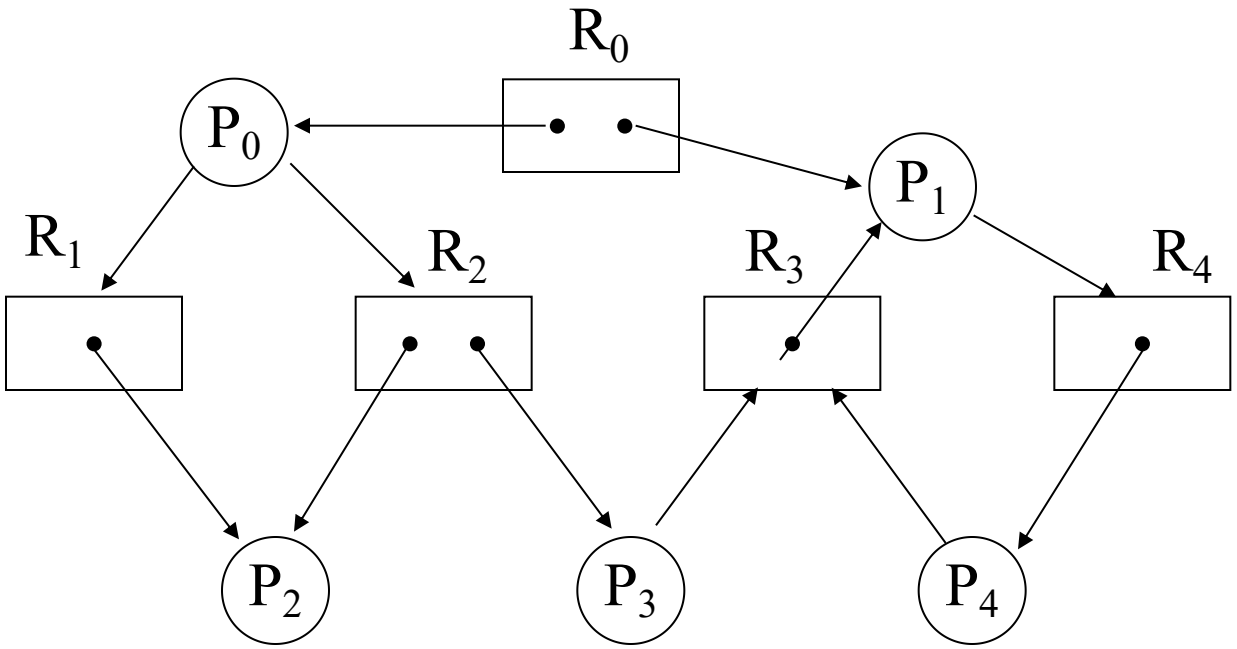
① 将其资源分配图简化，释放出资源，增加工作向量Work：$=Work+Allocation_i$。

② 将它记入L表中。

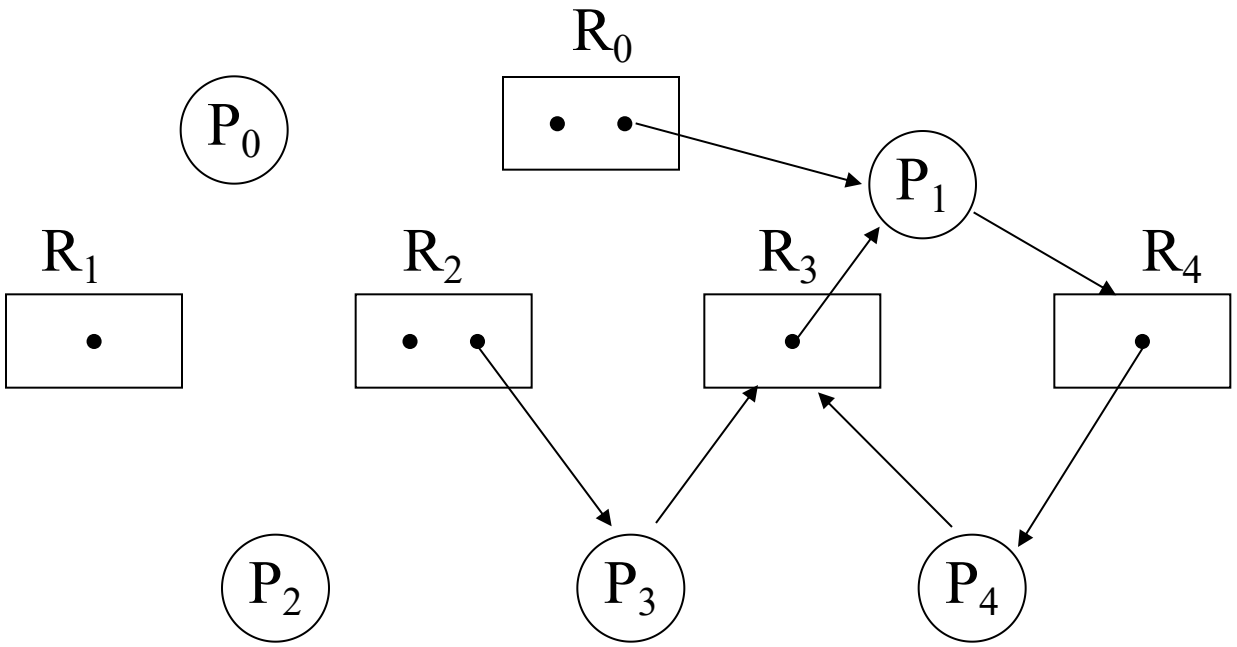
(4) 若不能把所有进程都记入L表中， 便表明系统状态S的资源分配图是不可完全简化的。 因此，该系统状态将发生死锁。

```
Work : =Available;
   L  : =  {Li|Allocationi=0∩Requesti=0}
for all Li ∉ L do
   begin
      for all Requesti≤Work do
         begin
            Work  : =Work+Allocationi;
            Li∪L;
         end
      end
deadlock:  =  (L={p1, p2, …, pn});
```

分配图化简：



解：化简得：



有一个循环等待的圈，死锁。

3.7.7 死锁的解除
Deadlock Recovery

- 以最小的代价恢复系统的运行。方法如下：
  - 1）重新启动
  - 2）撤消进程
  - 3）剥夺资源
  - 4）进程回退

- 方法2：进程终止
  - 终止所有的死锁进程 – OS中常用方法
  - 一次只终止一个进程直到取消死锁循环为止 – 基于某种最小代价原则。
- 选择原则
  - 已消耗CPU时间最少
  - 到目前为止产生的输出量最少
  - 预计剩余的时间最长
  - 目前为止分配的资源总量最少
  - 优先级最低
- 终止进程需要做很多工作

- 方法3：资源抢占：逐步从进程中强占资源给其它进程使用，直到死锁环被打破为止。考虑如下问题：
  - 选择一个牺牲品：抢占哪些资源和哪个进程，确定抢占顺序以使代价最小。
  - 饥饿：确保资源不会总是从同一个进程中被抢占
- 方法4：回滚：把每个死锁进程备份到前面定义的某些检查点，并且重新启动所有进程－需要系统构造重新运行和重新启动机制

What you need to do?

- 复习课本3. 7节的内容
- 课后作业： 习题19、22、29、31、34

See you next time!