



操作系统原理

南京师范大学 计算机与电子信息学院

夏年

Tel : 15052875526

Email : nian.xia@nnu.edu.cn

明理楼—108

第五章 死锁

课程结构

背景

- Chapter 1 计算机系统
- Chapter 2 操作系统

进程

- Chapter 3 进程控制和管理
- Chapter 5 并发：互斥与同步
- Chapter 6 并发：死锁

操作系统原理

内存

- Chapter 7 存储管理
- Chapter 8 虚拟内存

调度

- Chapter 4 处理器调度

输入输出 (I/O)

- Chapter 9 设备管理

文件

- Chapter 10 文件系统

本章教学目标

- 掌握死锁的定义和产生条件
- 掌握死锁避免的银行家算法
- 掌握死锁检测和解除方法

目录

- 5.1 死锁产生
- 5.2 死锁防止
- 5.3 死锁避免
- 5.4 死锁检测和解除

5.1 死锁产生

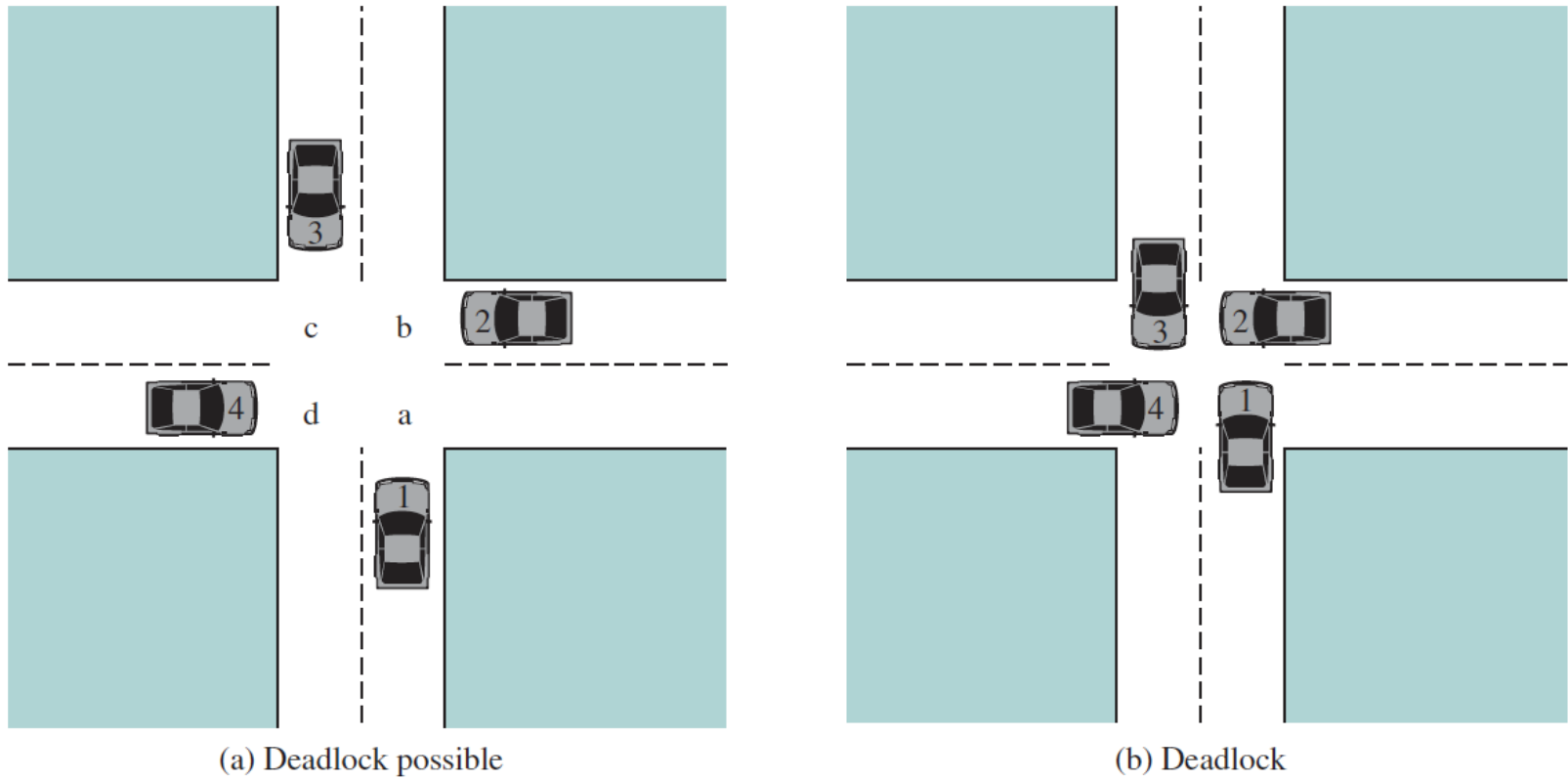


Figure 6.1 Illustration of Deadlock

5.1 死锁产生

```
Semaphore fork[5];  
  
for( int i = 0;i<5;i++)  
    fork[i] = 1;  
  
Process philosopher_i(){ //i=0,1,2,3,4;  
    while(true)  
    {  
        think();  
        P(fork[i]); //尝试获得左边叉子  
        P(fork[(i+1)%5]); //尝试获得右边叉子  
        eat();  
        V(fork[i]); //释放左边叉子  
        V(fork[(i+1)%5]); //释放右手叉子  
    }  
}
```

若5个哲学家同时拿起自己左手边 (或右手边)的筷子，则所有哲学家将处于等待状态，出现死锁。

哲学家就餐问题中的死锁

5.1 死锁产生

死锁产生的因素

- 系统拥有的资源数量
 - 进程对资源的使用要求
 - 资源分配策略
 - 并发进程的推进顺序
- 静态因素
- 动态因素

• 独占型资源

- 硬件资源：磁带机、绘图仪等；
- 软件资源：共享数据结构、链表等；

• 进程使用独占型资源的方式

- 申请资源，忙则等待
- 使用资源
- 归还资源

死锁的定义

如果一个进程集合中的每个进程都在等待只能由此集合中的其他进程才能引发的事件，而无限期陷入僵持的局面称为**死锁**

5.1 死锁产生

死锁产生的条件

- 互斥访问
 - 系统中存在临界资源，进程应互斥地使用这些资源。
- 占有和等待
 - 进程在请求资源得不到满足而等待时，不释放已有资源。
- 非剥夺
 - 已被占用的资源只能由属主在使用完时自愿释放，而不允许被其他进程剥夺。
- 循环等待
 - 存在循环等待链，每个进程在链中等待下一个进程所持有的资源，造成这组进程处于永远等待状态。

**必要
条件**

**充要
条件**

5.1 死锁产生

死锁的对策

- 死锁防止
 - 采用一种资源分配策略，消除造成死锁产生的4个条件之一。
- 死锁避免
 - 依据当前资源分配的状态作出动态资源分配选择，以避免进入死锁状态。
 - 如，银行家算法
- 死锁检测
 - 检测死锁的存在，并采用恢复机制将系统恢复到非死锁状态。

	方法	优点	缺点
死锁防止	预设资源分配策略，消除造成死锁产生的4个条件之一	依据预设的资源分配策略，保证不产生死锁	需要预先知道进程对于未来资源请求的知识；严重降低了程序的并发性和资源利用率
死锁避免	依据当前资源分配的状态对资源分配请求作出动态决策，避免进入死锁状态	对造成死锁的前三个必要条件不做限制；能支持更高的并发度	需要预先知道进程对于未来资源请求的知识；计算复杂度高
死锁检测和解除	对死锁的产生不采取任何预防措施； 定期检测死锁是否产生，若产生死锁，则予以解除	并发度最高； 当死锁是小概率事件时，较为经济	会出现死锁； 出现死锁时，解除死锁的代价较高

目录

- 5.1 死锁产生
- **5.2 死锁防止**
- 5.3 死锁避免
- 5.4 死锁检测和解除

5.2 死锁防止

- **破坏条件1（互斥条件）**

- 使资源可同时访问而非互斥使用，也就没有进程会阻塞在资源上，从而不发生死锁。
- 可重入程序、只读数据文件、时钟、磁盘等资源可以非互斥使用
- 但可写文件、磁带机等只能互斥占有
- **Q: 哲学家进餐问题中破坏互斥条件意味着什么？**

5.2 死锁防止

- **破坏条件2（占有和等待）**

- 要求进程在执行之前就申请所需要的全部资源，且直到所有资源都得到满足后才开始执行。进程在执行过程中不再申请资源，就不会出现占有某些资源再等待另一些资源的情况。
 - 易于实现，但缺点在于严重降低资源利用率。
 - 需要进程知道运行过程中将使用到的所有资源。
-
- **Q: 哲学家进餐问题中破坏占有和等待条件意味着什么？**

5.2 死锁防止

- **破坏条件3（非剥夺条件）**

- 方法一：如果拥有资源的进程在申请新资源时未得到满足，则进程放弃所有已占用资源（隐式剥夺），若仍需要占用上述资源，则应向系统重新提出申请，同时申请原有资源和新资源；
 - 方法二：当一个进程A申请当前被另一个进程B占用的资源，而B正等待其它资源时，操作系统可以剥夺B进程，要求其释放资源（显式剥夺）。
 - 该方法仅适用于资源的状态可以很容易被保存和恢复的资源，如处理器。
-
- **Q: 哲学家进餐问题中破坏不剥夺条件意味着什么？**

5.2 死锁防止

• 破坏条件4（循环等待条件）

- 可以为资源类型排序，并采用按序分配策略，即进程只能按照规定的顺序申请资源。
- 为什么上述方法能保证不产生死锁？
- 反证法：
 - 若产生死锁，则一定存在进程序列 P_1, P_2, \dots, P_m 形成循环等待。
 - 设 P_1 占有资源 R_1 ，请求资源 R_2 ，而 P_2 占有资源 R_2 ，申请资源 R_3, \dots, P_m 占有资源 R_m ，申请资源 R_1 。
 - 则根据按序申请原则，有：

$$H(R_1) < H(R_2) < \dots < H(R_m) < H(R_1)$$

矛盾！

- Q: 哲学家进餐问题中破坏循环等待条件怎么实现？

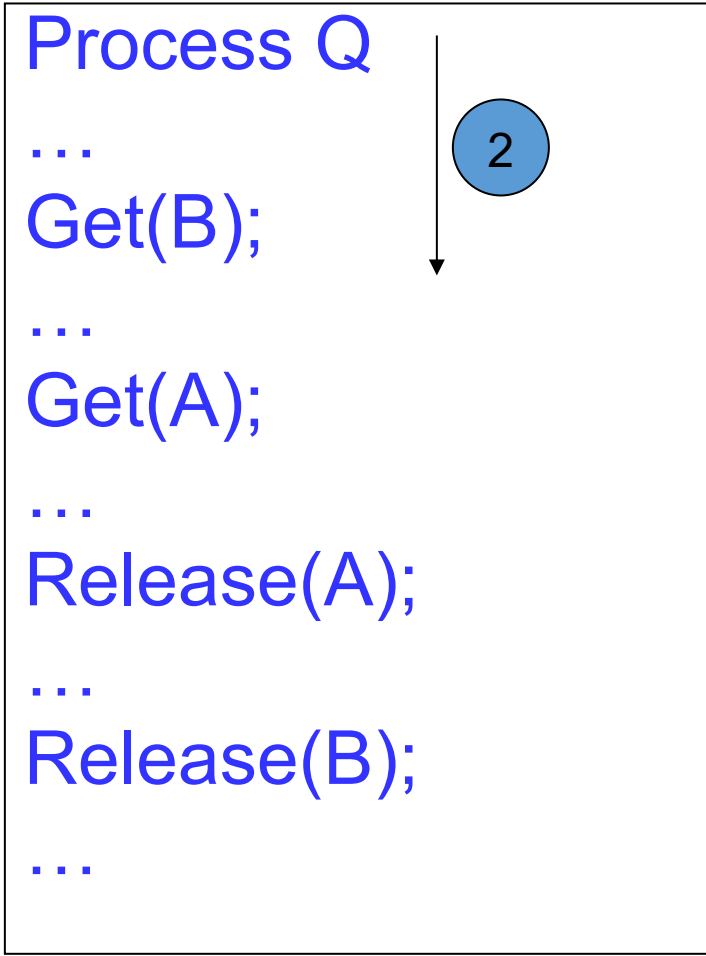
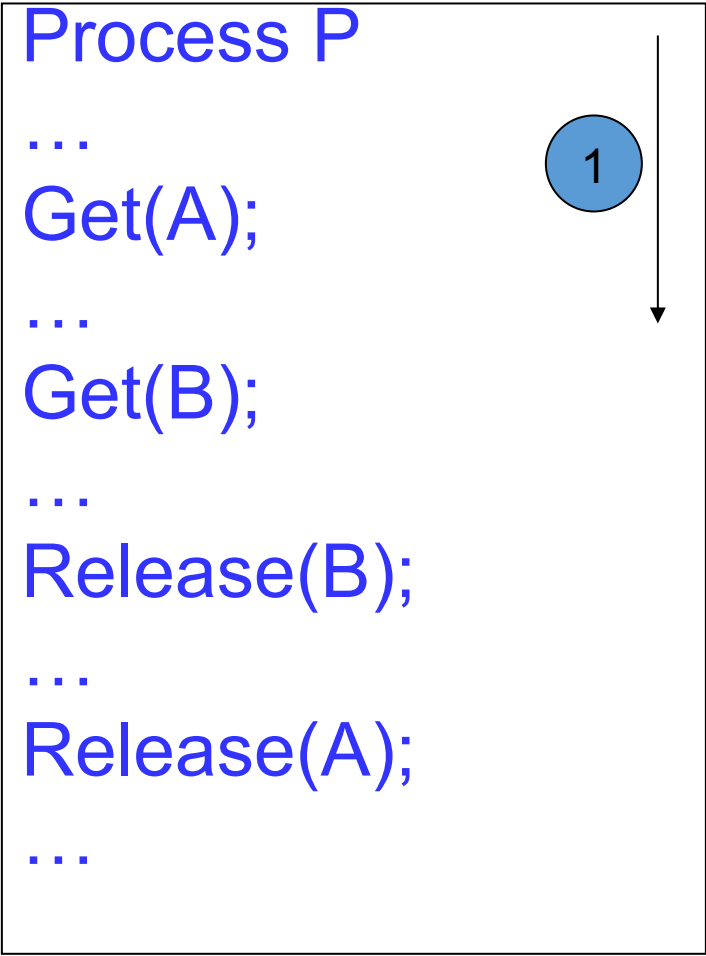
目录

- 5.1 死锁产生
- 5.2 死锁防止
- **5.3 死锁避免**
- 5.4 死锁检测和解除

5.3 死锁避免

- 允许系统中**同时存在前三个必要条件**，通过合适的资源分配算法确保不会出现进程循环等待条件。
- 死锁避免方法能**支持更多进程并发执行**，动态的确定是否分配资源给提出请求的进程。
- 如果一个进程当前请求资源会导致死锁，系统将拒绝启动此进程；如果一个资源分配会导致系统下一步死锁，便拒绝本次分配。

5.3 死锁避免



是否允许将 **B** 分配给 **Q** ?

设系统中只有两个资源A, B; 且资源数分别为1

5.3 死锁避免

	资源需求		已分配资源		尚需分配资源		可用资源数	
	A	B	A	B	A	B	A	B
P	1	1	1	0	0	1	0	1
Q	1	1	0	0	1	1		

进程Q执行Get(B)之前的系统状态

5.3 死锁避免

	资源需求		已分配资源		尚需分配资源		可用资源数	
	A	B	A	B	A	B	A	B
P	1	1	1	0	0	1	0	0
Q	1	1	0	1	1	0		

假设将B分配给Q

产生死锁！

5.3 死锁避免

Process P	Process Q
....
Get A	Get B
....
Get B	Get A
....
Release A	Release B
....
Release B	Release A
....

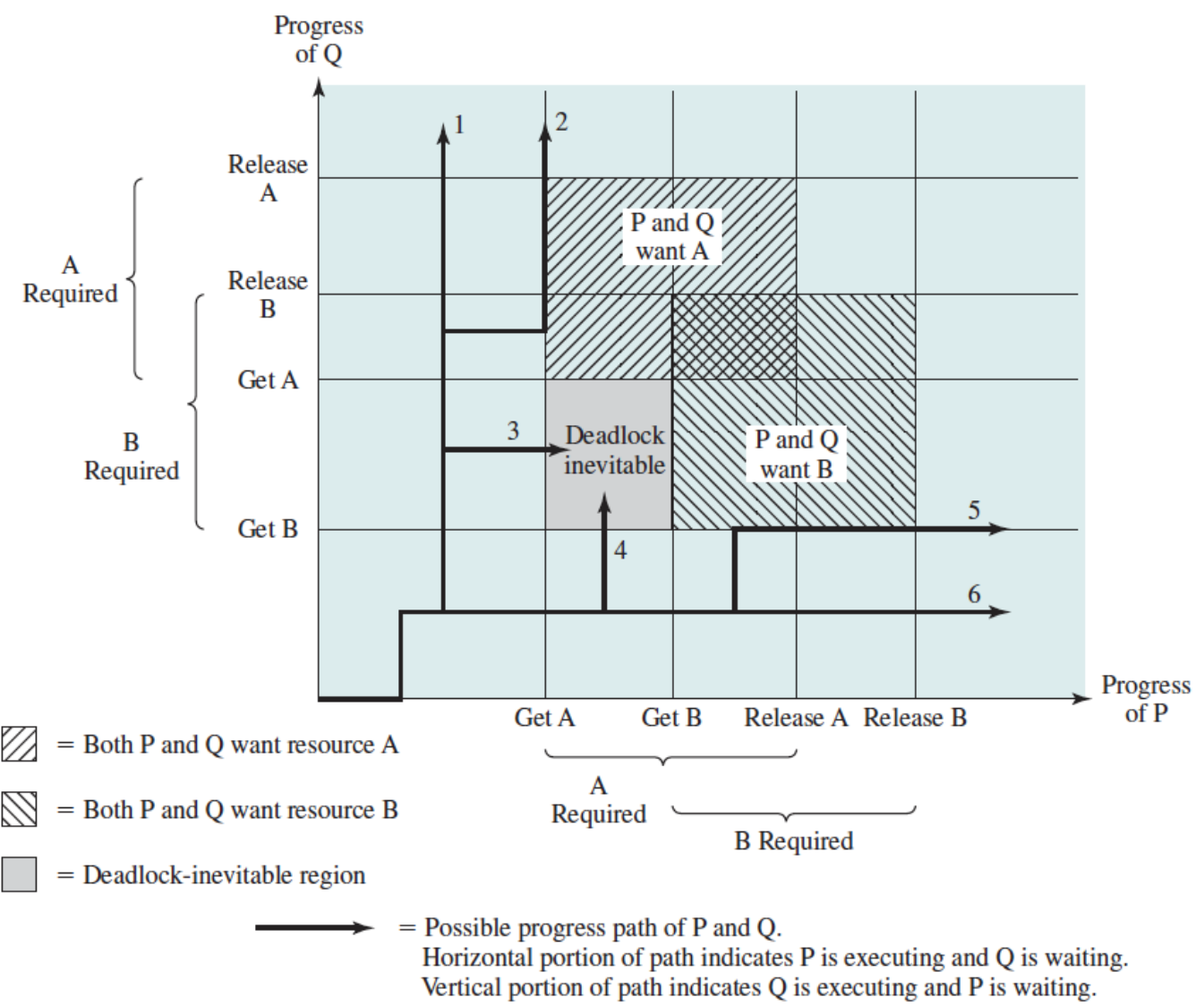


Figure 6.2 Example of Deadlock

5.3 死锁避免

- **银行家算法**：1965年由荷兰计算机科学家Dijkstra提出
- 假定小城镇银行家拥有资金数量为 Σ ，被 N 个客户共享，银行家对客户提出下列**约束条件**：
 - 每个客户必须预先说明所要的最大资金量；
 - 每个客户每次提出部分资金量申请并获得分配；
 - 如果银行满足客户对资金的最大需求量，那么客户在资金运作后，在有限的时间内全部归还银行。
- **银行家保证做到**：
 - 若一个客户所要的最大资金量不超过 Σ ，银行一定会接纳此客户，并满足其资金要求；
 - 银行在收到一个客户的资金申请中，可能会因资金不足而让客户等待，但保证在有限的时间内让客户获得资金。

5.3 死锁避免

- 银行家算法里的数据结构
 - 资源总数向量
 - 可用资源数向量
 - 最大需求矩阵
 - 已分配矩阵
- 银行家算法里的两个重要概念
 - 安全状态
 - 安全序列

5.3 死锁避免

- 假设系统中有 n 个进程和 m 类资源

- 系统每类资源总数向量

$$Resource = (R_1, R_2, \dots, R_m)$$

- 系统中当前每类资源可用数向量

$$Available = (V_1, V_2, \dots, V_m)$$

- 进程对各类资源的最大需求矩阵

$$Claim = \begin{bmatrix} C_{11} & C_{12} & \cdots & C_{1m} \\ C_{21} & C_{22} & \cdots & C_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ C_{n1} & C_{n2} & \cdots & C_{nm} \end{bmatrix}$$

- 系统中当前资源的已分配情况矩阵

$$Allocation = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1m} \\ A_{21} & A_{22} & \cdots & A_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nm} \end{bmatrix}$$

5.3 死锁避免

资源总数恒等关系

$$R_i = V_i + \sum_{k=1}^n A_{ki} \quad i = 1, 2, \dots, m$$

申请资源上限限制

$$C_{ki} \leq R_i \quad k = 1, 2, \dots, n; i = 1, 2, \dots, m$$

分配资源上限限制

$$A_{ki} \leq C_{ki} \quad k = 1, 2, \dots, n; i = 1, 2, \dots, m$$

5.3 死锁避免

安全状态和安全序列

- 系统的状态是安全的是指系统**存在**某种方法为每个进程分配资源，使每个进程都能完成执行而不导致死锁。
- 在 T_0 时刻系统处于**安全状态**，是指**当且仅当**存在一个进程序列 P_1, P_2, \dots, P_n ，对**任何进程** P_k 满足：

$$C_{ki} - A_{ki} \leq V_i + \sum_{j=1}^{k-1} A_{ji}, \quad k = 1, 2, \dots, n; i = 1, 2, \dots, m$$

- **安全序列**：满足该条件的进程序列称为安全序列

注：安全序列可能有多

5.3 死锁避免

银行家算法描述

输入：当前状态、进程 P_i 的资源申请请求 $Request$ ：

- ① 若 $Request > Claim_i - Allocation$ ，报错，否则，转步骤2
- ② 若 $Request > Available$ ，表示资源不够，进程 P_i 等待，否则转步骤3
- ③ 系统对 P_i 进程的资源请求进行**试探性分配**，执行：
$$Available = Available - Request;$$
$$Allocation_i = Allocation_i + Request$$
- ④ 执行**安全性测试算法**，如果状态安全则接受试分配，否则放弃试分配，进程 P_i 等待。

5.3 死锁避免

安全性测试算法

- ① 设 $CurrentAvail$ 和 $Finish$ 分别为 m 和 n 维向量，初始化
 $CurrentAvail = Available$, $Finish[k] = false$ ($k = 1, 2, \dots, n$);
- ② 找到同时满足下列条件的 k
 - a. $Finish[k] == false$;
 - b. $Claim_k - Allocation_k \leq CurrentAvail$若不存在这样的 k , 则转步骤4
- ③ $CurrentAvail = CurrentAvail + Allocation_k$;
 $Finish[k] = true$;
转步骤2
- ④ 若对所有 k , 均有 $Finish[k] = true$, 则系统是安全的, 否则系统是不安全的。

5.3 死锁避免

进程	Claim			Allocation			Resource			Claim-Allocation			available		
	A	B	C	A	B	C	A	B	C	A	B	C	A	B	C
P ₀	7	5	3	0	1	0	10	5	7	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			

5.3 死锁避免

进程	Claim			Allocation			Resource			Claim-Allocation			available		
	A	B	C	A	B	C	A	B	C	A	B	C	A	B	C
P ₀	7	5	3	0	1	0	10	5	7	7	4	3	5	3	2
P ₁	0	0	0	0	0	0				0	0	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			

5.3 死锁避免

进程	Claim	Allocation	Resource	Claim- Allocation	available
	A B C	A B C	A B C	A B C	A B C
P ₀	7 5 3	0 1 0	10 5 7	7 4 3	7 4 3
P ₁	0 0 0	0 0 0		0 0 0	
P ₂	9 0 2	3 0 2		6 0 0	
P ₃	0 0 0	0 0 0		0 0 0	
P ₄	4 3 3	0 0 2		4 3 1	

5.3 死锁避免

进程	Claim	Allocation	Resource	Claim- Allocation	available
	A B C	A B C	A B C	A B C	A B C
P ₀	0 0 0	0 0 0	10 5 7	0 0 0	7 5 3
P ₁	0 0 0	0 0 0		0 0 0	
P ₂	9 0 2	3 0 2		6 0 0	
P ₃	0 0 0	0 0 0		0 0 0	
P ₄	4 3 3	0 0 2		4 3 1	

5.3 死锁避免

进程	Claim	Allocation	Resource	Claim- Allocation	available
	A B C	A B C	A B C	A B C	A B C
P ₀	0 0 0	0 0 0	10 5 7	0 0 0	10 5 5
P ₁	0 0 0	0 0 0		0 0 0	
P ₂	0 0 0	0 0 0		0 0 0	
P ₃	0 0 0	0 0 0		0 0 0	
P ₄	4 3 3	0 0 2		4 3 1	

5.3 死锁避免

进程	Claim	Allocation	Resource	Claim- Allocation	available
	A B C	A B C	A B C	A B C	A B C
P ₀	0 0 0	0 0 0	10 5 7	0 0 0	10 5 7
P ₁	0 0 0	0 0 0		0 0 0	
P ₂	0 0 0	0 0 0		0 0 0	
P ₃	0 0 0	0 0 0		0 0 0	
P ₄	0 0 0	0 0 0		0 0 0	

存在安全序列：P₁, P₃, P₀, P₂, P₄, 初始状态为安全状态

5.3 死锁避免

初始状态下，是否允许 P_1 申请1个A资源，2个C资源？

$$Request(P_1) = (1, 0, 2)$$

初始检查：

$$Request(P_1) = (1, 0, 2) \leq Claim(P_1) - Allocation(P_1) = (1, 2, 2)$$

$$Request(P_1) = (1, 0, 2) \leq available = (3, 3, 2)$$

5.3 死锁避免

进程	Claim	Allocation	Resource	Claim-Allocation	available
	A B C	A B C	A B C	A B C	A B C
P ₀	7 5 3	0 1 0	10 5 7	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	

$$Request(P_1) = (1, 0, 2)$$

5.3 死锁避免

进程	Claim			Allocation			Resource			Claim-Allocation			available		
	A	B	C	A	B	C	A	B	C	A	B	C	A	B	C
P ₀	7	5	3	0	1	0	10	5	7	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			

试探性分配 $Request(P_1) = (1,0,2)$

5.3 死锁避免

进程	Claim			Allocation			Resource			Claim-Allocation			available		
	A	B	C	A	B	C	A	B	C	A	B	C	A	B	C
P ₀	7	5	3	0	1	0	10	5	7	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			

安全性测试

5.3 死锁避免

进程	Claim			Allocation			Resource			Claim-Allocation			available		
	A	B	C	A	B	C	A	B	C	A	B	C	A	B	C
P ₀	7	5	3	0	1	0	10	5	7	7	4	3	5	3	2
P ₁	0	0	0	0	0	0				0	0	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	安全性测试			4	3	1			

回到之前的状态，存在安全序列P₁, P₃, P₀, P₂, P₄, 因此可以将资源分配给P₁

5.3 死锁避免

进程	Claim			Allocation			Resource			Claim-Allocation			available		
	A	B	C	A	B	C	A	B	C	A	B	C	A	B	C
P ₀	7	5	3	0	1	0	10	5	7	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			

接受试探性分配

5.3 死锁避免

在为 P_1 分配资源后，是否允许 P_4 申请3个A资源，3个B资源？

$$Request(P_4) = (3, 3, 0)$$

初始检查：

$$\begin{aligned} Request(P_4) = (3, 3, 0) &\leq Claim(P_4) - Allocation(P_4) \\ &= (4, 3, 1) \end{aligned}$$

$$Request(P_4) = (3, 3, 0) > available = (2, 3, 0), \quad \text{拒绝分配！}$$

5.3 死锁避免

在为 P_1 分配资源后，是否允许 P_0 申请2个B资源？

$$Request(P_0) = (0, 2, 0)$$

初始检查：

$$\begin{aligned} Request(P_0) = (0, 2, 0) &\leq Claim(P_0) - Allocation(P_0) \\ &= (7, 4, 3) \end{aligned}$$

$$Request(P_0) = (0, 2, 0) \leq available = (2, 3, 0)$$

5.3 死锁避免

进程	Claim	Allocation	Resource	Claim- Allocation	available
	A B C	A B C	A B C	A B C	A B C
P ₀	7 5 3	0 1 0	10 5 7	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	

$$Request(P_0) = (0,2,0)$$

5.3 死锁避免

进程	Claim			Allocation			Resource			Claim-Allocation			available		
	A	B	C	A	B	C	A	B	C	A	B	C	A	B	C
P ₀	7	5	3	0	3	0	10	5	7	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			

试探性分配 $Request(P_0) = (0,2,0)$
 剩余资源无法满足任何进程，因此，拒绝本次分配请求！

5.3 死锁避免

银行家算法步骤总结

- 三步骤
 - 初始检查
 - 试分配
 - 安全性检测

目录

- 5.1 死锁产生
- 5.2 死锁防止
- 5.3 死锁避免
- **5.4 死锁检测和解除**

5.4 死锁检测和解除

- 死锁防止和死锁避免都对资源分配施加了一定的限制，不利于进程对系统资源的充分共享
- 死锁检测和解除
 - 不对资源的分配施加任何限制，允许死锁的发生
 - 定时地**检测**系统内是否出现死锁
 - 如何检测？
 - 若出现死锁，则采取措施**解除**死锁
 - 如何解除？

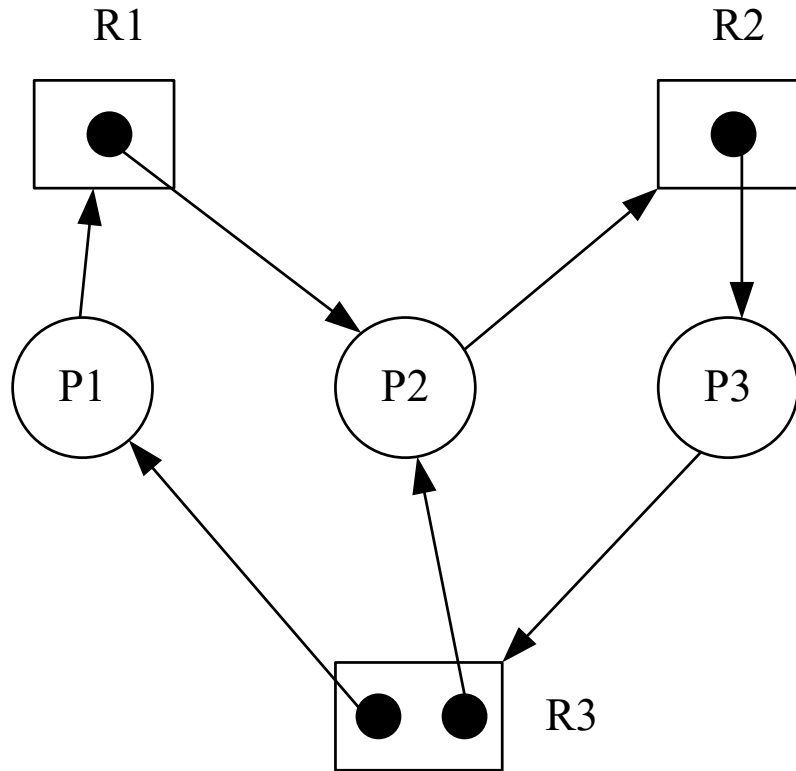
5.4 死锁检测和解除

资源分配图

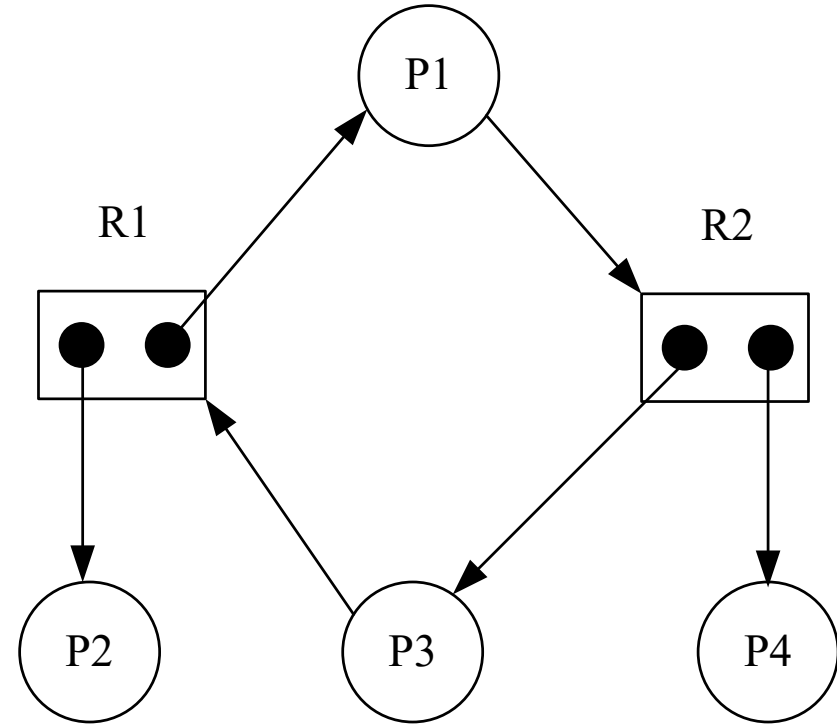
- 节点集
 - 由所有进程和资源类构成
- 边集
 - 请求边 $P_i \rightarrow R_q$, 进程 P_i 请求资源类 R_q 中的一个资源
 - 分配边 $R_q \rightarrow P_i$, 资源类 R_q 中的一个资源被分配给了进程 P_i .

5.4 死锁检测和解除

资源分配图



(a)



(b)

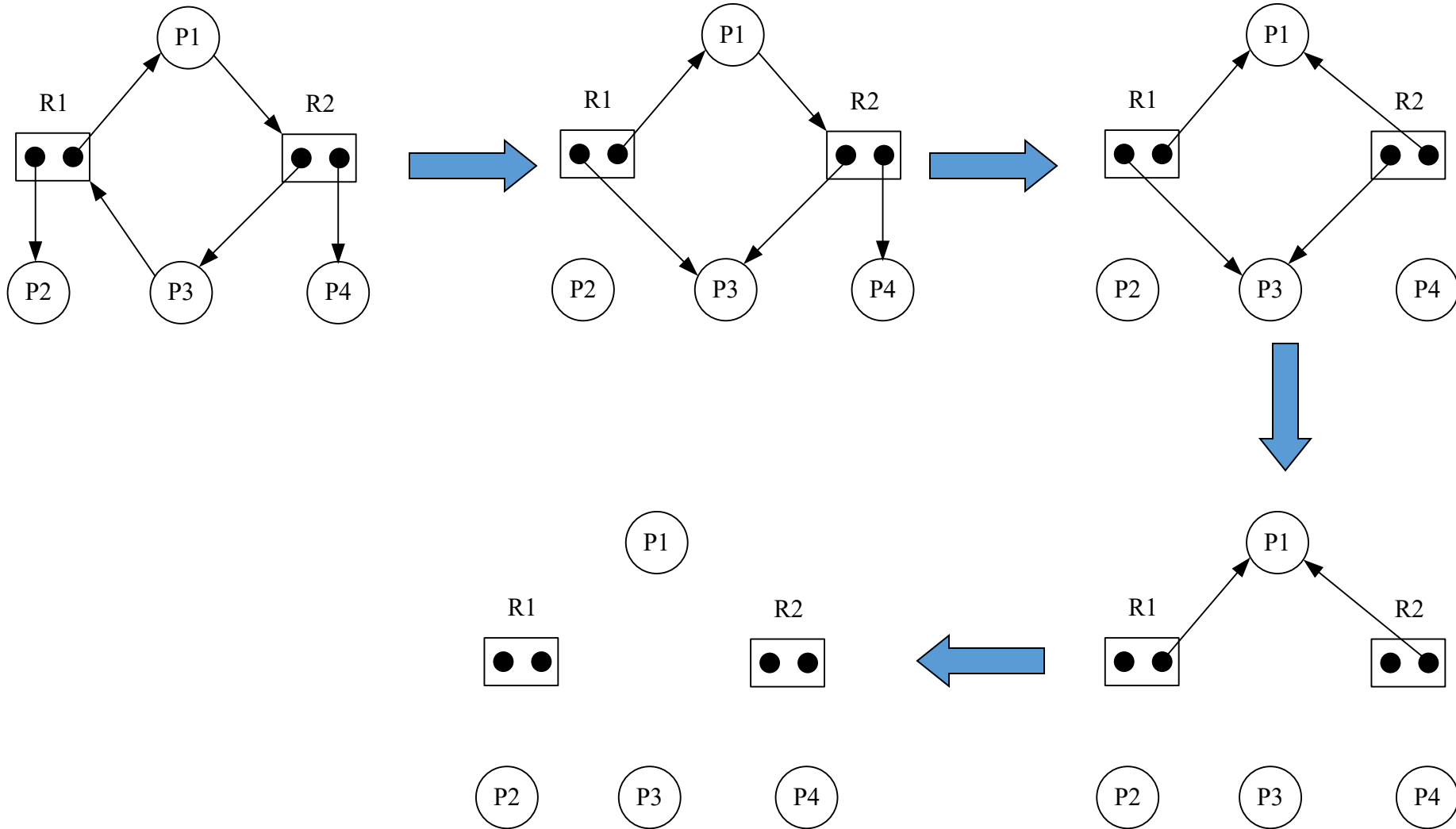
资源分配图示例

5.4 死锁检测 and 解除

资源分配图与死锁的关系

- 分两种情况考虑
 - 每类资源仅有一个实例
 - 每类资源有多个实例
 - 若每类资源仅有一个实例，则存在死锁的**充要条件**是资源分配图中存在环路。
 - 若每类资源存在多个实例，则资源分配图中存在环路只是死锁发生的**必要条件**，而非充分条件
- 可完全简化
不可完全简化 \iff 系统处于死锁状态

5.4 死锁检测和解除



可完全简化的资源分配图

5.4 死锁检测和解除

死锁检测算法

与银行家算法中的安全性测试类似：

- ① 设 $CurrentAvail$ 和 $Finish$ 分别为 m 和 n 维向量。初始化 $CurrentAvail = Available$;对任意 k , 若 $Allocation_k \neq 0$, 则 $Finish[k] = false$, 否则 $Finish[k] = true$;
- ② 查找同时满足下列条件的 k
 - a. $Finish[k] = false$;
 - b. $Claim_k - Allocation_k \leq CurrentAvail$若不存在这样的 k , 转向步骤4
- ③ $CurrentAvail = CurrentAvail + Allocation_k$
 $Finish[k] = true$;
转步骤2.
- ④ 若对某些 i , 有 $Finish[i] = false$, 则系统处于死锁状态, 并且, 进程 P_i 处于死锁链中。

5.4 死锁检测和解除

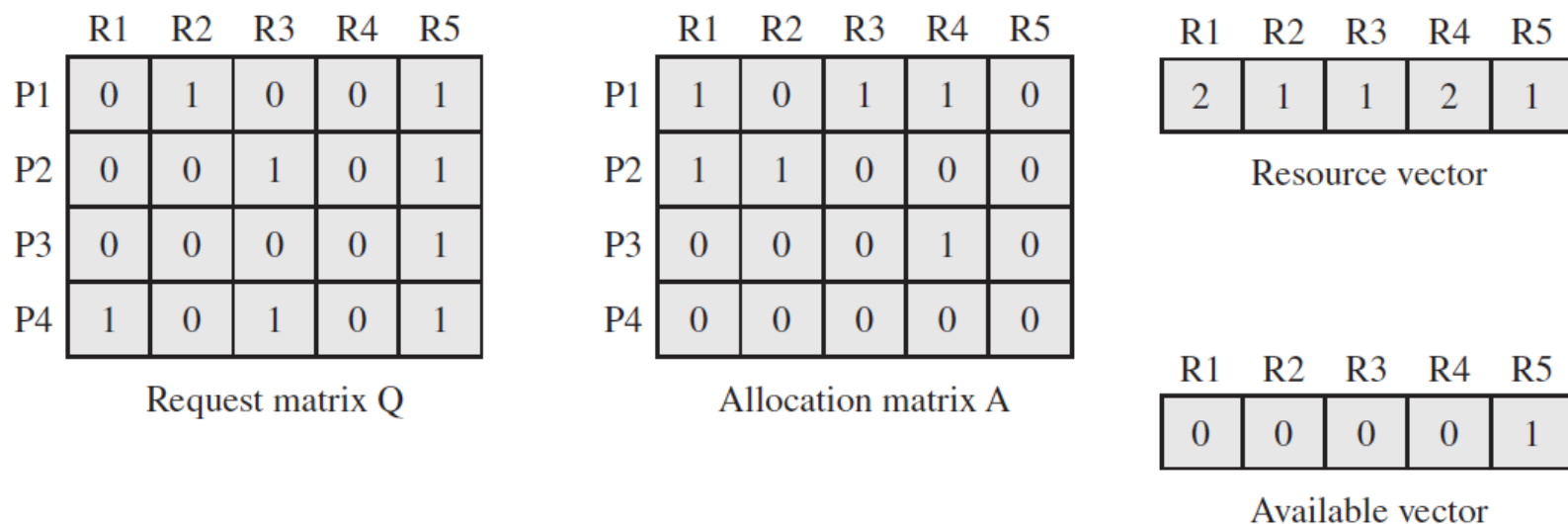


Figure 6.10 Example for Deadlock Detection

=Claim-Allocation

$$1 \text{ CurrentAvail} = (0,0,0,0,1)$$

$$2 \text{ Claim}_3 - \text{Allocation}_3 = (0,0,0,0,1) \leq \text{CurrentAvail}, \text{ 因此, } k=3$$

$$3 \text{ CurrentAvail} = (0,0,0,0,1) + \text{Allocation}_3 = (0,0,0,1,1)$$

至此，不存在 k ，使得 $\text{Claim}_k - \text{Allocation}_k \leq \text{CurrentAvail}$ ，因此该状态下，系统存在死锁， $P1, P2, P4$ 处于死锁链中

5.4 死锁检测和解除

死锁解除

- 结束所有死锁的进程
 - 代价大
- 逐个结束死锁链中的进程，直到死锁解除为止
 - 每结束一个进程，都需要运行一次死锁检测算法
 - 选择哪个进程终止？类似CPU调度的策略，如CPU消耗时间最少，优先级最低，分得资源数最少等
- 剥夺陷于死锁的进程所占用的资源，但并不撤销此进程，直至死锁解除
 - 如何选择进程？如占有资源最少的死锁进程
 - 回退。被剥夺的进程如何处理？需要回退到安全点

本章小节

- 死锁是指并发进程由于互相等待独占性资源而无限期陷入僵局的一种局面
- 死锁产生的条件包括互斥条件、占有和等待条件、不剥夺条件、循环等待条件
- 解决死锁问题的方法有三类：死锁防止、死锁避免、死锁检测和解除；前两者对资源分配加以限制，后者不做限制，允许死锁产生
- 银行家算法是一种死锁避免算法，主要包括试探性分配和安全性检测两个步骤