- 死锁的四个必要条件
  - 互斥（Mutual exclusion）、持有并等待（Hold and wait）、不可抢占（No preemption）、环路等待（Circular wait condition）
- 资源分配图（Resource-Allocation Graph）
  - 每种类型一个资源
  - 每种类型一个资源每种类型多个资源
- 死锁检测（Deadlock Detection）
  - 每种类型一个资源→ 可以利用环路检测算法进行死锁检测
  - 每种类型一个资源每种类型多个资源→ 环路检测算法失效，需要利用基于资源分配矩阵、请求矩阵、可用向量的算法进行检测

- 死锁避免（Deadlock Avoidance）
  - 安全状态（safe）、非安全状态（Unsafe）
  - 银行家算法（Banker’ s algorithm）
- 死锁预防（Deadlock Prevention）
  - 破快互斥条件
  - 破坏占有并等待
  - 破坏不可抢占
  - 破坏环路等待

一、必要条件&&死锁预防

死锁预防：破坏互斥条件，使死锁无法发生

1、互斥条件

资源不能被多个进程同时使用

- RCU

2、不剥夺条件

资源用完之前不能被抢走，只能是主送释放

- 破坏不剥夺条件，暂时剥夺资源
- 缺点：可能前一阶段工作失效，且不能针对打印机，只能针对CPU寄存器和内存等易于恢复的

3、请求并保持条件

进程占有资源还请求其他资源

- 运行前一次性把资源全申请到（哲学家问题）
- 缺点：严重浪费资源，造成饥饿

4、循环等待条件

存在一种进程资源的循环等待链

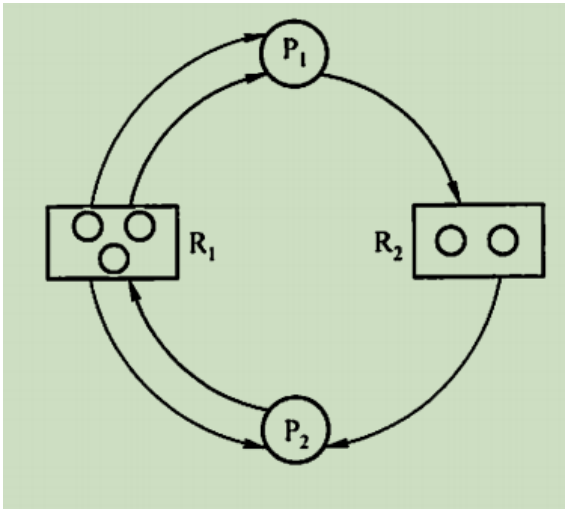
- 采用顺序资源分配，请求资源有顺序
- 缺点：编号必须稳定，新设备增加难

二、死锁检测&资源分配图

1、每种类型一个资源

有环就有死锁，检测环路即可

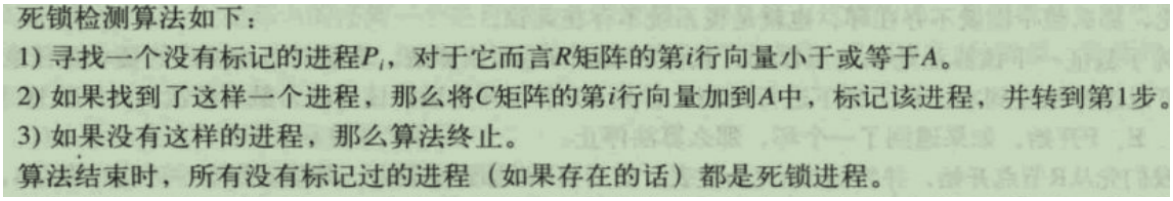
2、每种类型多个资源



- 请求边+分配边
- 资源分配图含有圈但未必死锁：同类资源数>1

检测方法：

- 找不阻塞不孤立的点（有边，且申请数量不超过空闲资源），运行完释放，删除相关边，使其孤立
- 死锁定理：S为死锁：资源分配图对当前S的状态不可完全简化


死锁检测算法如下：
1) 寻找一个没有标记的进程P_i，对于它而言R矩阵的第i行向量小于或等于A。
2) 如果找到了这样一个进程，那么将C矩阵的第i行向量加到A中，标记该进程，并转到第1步。
3) 如果没有这样的进程，那么算法终止。
算法结束时，所有没有标记过的进程（如果存在的话）都是死锁进程。

三、死锁避免

安全状态

可以找到一个安全序列

- 并非所有不安全状态都是死锁

> 可能会提前终止，可能会提前释放部分资源，实际需要的最大资源小于声明的最大需求资源，申请的资源为可消耗性资源

银行家算法

1、矩阵

Available

Max（或Claim）

Allocation：已分配

Need小于等于Max-Allocation

2、流程

- 检查请求资源小于最大声明资源
- 检查请求资源小于可用资源
- 尝试分配资源，更改Available、Allocation、Need（-+-）
- 执行安全性算法（找到安全序列才能完成本次分配，否则等待）

3、安全性算法

看能不能找到安全序列

- 初始化安全序列为空
- 计算Need后，看Available对哪个进程来说够用，作为安全序列的第一个进程
- 释放这个进程资源，更新Available，删减Need
- 重复直到找到安全序列

刷题

被锁住的可能是部分进程，判断不要半途而废
当前存在安全序列不代表万事大吉，序列后面的进程提出申请还是要拒绝，因为满足了就会有死锁
注意检查Need+Allocation不能超过Max
银行家算法难以实现：很难知道每个进程的Max需求，进程数量是动态变化的