2.4死锁

死锁产生的原因

- 系统资源的竞争
- 进程推进顺序非法

死锁产生的必要条件

- 互斥条件：仅为一个进程所占有（打印机）
- 不可剥夺条件：自己释放
- 请求并保持条件：保持一个资源，保持不放
- 循环等待条件

死锁的处理策略

死锁的预防

设发破坏产生死锁的条件之一

- 破坏互斥条件：将互斥使用的资源改造为允许共享使用（打印机只能互斥使用）（不太可行）
- 破坏不可剥夺条件：释放已经保持的所有资源，已占用的资源会被暂时释放（这种方法常用于易于保存和和恢复的资源，如CPU的寄存器和内存资源，不能用于打印机之类的资源）（复杂）
- 破坏请求并保持条件
  - 1.采用预先静态分配方法：在运行前一次申请完所需要的全部资源 （缺点：会产生浪费 会导致饥饿现象）
  - 2.进程只获取初期所需的资源后运行
- 破坏循环等待条件：采用顺序资源分配法，编号递增的方法进行请求资源（缺点：不便于增加新类型设备 且必须按次序进行申请资源）

避免死锁

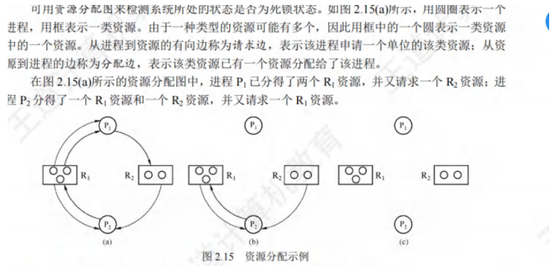
用某种方法防止系统进入不安全状态

- 1.系统安全状态：系统安全状态一定是无死锁过程，死锁状态一定是不安全状态
- 2.银行家算法
  - 需要知道总量信息
  - 判断是否为不安全状态

死锁的检查与解除

允许进程发生死锁，通过某种方法检查并解除

死锁定理



可用资源分配图来程测系统所处的状态是否为死锁状态。如图 2.15(a)所示，用圆圈表示一个进程，用框表示一类资源。由于一种类型的资源可能有多个，因此用框中的一个圆表示一类资源中的一个资源。从进程到资源的有向边称为请求边，表示该进程申请一个单位的该类资源；从资源到进程的边称为分配边，表示该类资源已有一个资源分配给了该进程。

在图 2.15(a)所示的资源分配图中，进程 P_1 已分得了两个 R_1 资源，并又请求一个 R_2 资源；进程 P_2 分得了一个 R_1 资源和一个 R_2 资源，并又请求一个 R_1 资源。

图 2.15 资源分配示例

简化资源分配图可检测系统状态 S 是否为死锁状态。简化方法如下：

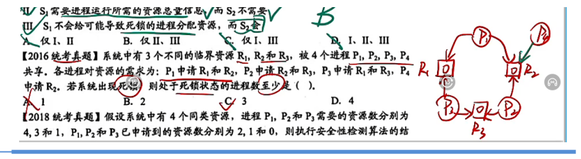
1）在资源分配图中，找出既不阻塞又不孤立的进程 P_i（找出一条有向边与它相连，且该有向边对应资源的申请数量小于或等于系统中已有的空闲资源数量，如在图 2.15(a)中，R_1 没有空闲资源，R_2 有一个空闲资源。若所有连接该进程的边均满足上述条件，则这个进程能继续运行直至完成，然后释放它所占有的所有资源）。消去它所有的请求边和分配边，使之成为孤立的节点。在图 2.15(a)中，P_1 是满足这一条件的进程节点，将 P_1 的所有边消去，便得到图 2.15(b)所示的情况。

这里要注意一个问题，判断某种资源是否有空闲，应该用它的资源数量减去它在资源分配图中的出度，例如在图 2.15(a)中，R_1 的资源数为 3，而出度也为 3，所以 R_1 没有空闲资源，R_2 的资源数为 2，出度为 1，所以 R_2 有一个空闲资源。

2）进程 P_i 所释放的资源，可以唤醒某些因等待这些资源而阻塞的进程，原来的阻塞进程可能变为非阻塞进程。在图 2.15(a)中，P_2 就满足这样的条件。根据 1）中的方法进行一系列简化后，若能消去图中所有的边，则称该图是可完全简化的，如图 2.15(c)所示。

S 为死锁的条件是当且仅当 S 状态的资源分配图是不可完全简化的，该条件为死锁定理。

- 1.死锁检测：可用资源分配图
- 2.死锁解除
  - 1.资源剥夺法
  - 2.撤销进程法
  - 3.进程回退法



Ⅱ. S_1 需要进程运行所需的资源总量很大，而 S_2 不需要
Ⅲ. S_1 不会给可能导致死锁的进程分配资源，而 S_2 会

A. 仅Ⅰ、Ⅱ　B. 仅Ⅱ、Ⅲ　C. 仅Ⅰ、Ⅲ　D. Ⅰ、Ⅱ、Ⅲ

【2016 统考真题】系统中有 3 个不同的临界资源 R_1、R_2 和 R_3，被 4 个进程 P_1、P_2、P_3、P_4 共享，各进程对资源的需求为：P_1 申请 R_1 和 R_2，P_2 申请 R_2 和 R_3，P_3 申请 R_1 和 R_3，P_4 申请 R_2。若系统出现死锁，则处于死锁状态的进程数至少是（　）。

A. 1　B. 2　C. 3　D. 4

【2018 统考真题】假设系统中有 4 个同类资源，进程 P_1、P_2 和 P_3 需要的资源数分别为 4, 3 和 1，P_1、P_2 和 P_3 已申请到的资源数分别为 2, 1 和 0，则执行安全性检测算法的结

死锁处理策略的比较

	主要优点	主要缺点	各种可能模式	资源分配策略
死锁预防	突发处理，不必进行剥夺	效率低，初始化时间长，剥夺次数多，不灵活申请资源	一次请求所有资源，资源剥夺，资源按序分配	保守，宁可资源闲置
死锁避免	不必进行剥夺	必须知道将来的资源需求，进程不能被长时间阻塞	寻找可能安全允许顺序	是预防与检测的折中（判断是否发生死锁）
死锁检测	不延长进程初始化的时间，允许对死锁进行现场处理	通过剥夺解除死锁，造成损失	定期检查死锁是否已经发生	宽松，只要允许就分配资源