5. 死锁

5.1. 死锁概念于特征

1️⃣概念：两个及以上进程互相等待对方资源，无外力作用就无法推进执行

2️⃣实例：进程P1占用打印机且申请IO设备，进程2占用IO但是申请打印机

3️⃣死锁的特点：

1. 互斥：对于资源，一次性只能有一个进程访问
2. 占有并等待：进程至少占有一个进程，等待另一个被其他进程占有的资源
3. 不可抢占：进程要在执行完后才释放资源，不会中途被抢走
4. 循环等待：等待资源的进程间存在环

总结：至少两个占有资源但又等待资源的进程才会产生死锁

5.2. 死锁的必要条件

5.2.1. OS的资源分类

1️⃣根据资源性质(事实上进程可抢占与否完全取决于资源类型)

1. 可剥夺/抢占资源：别的进程可以从本进程处把这个资源抢走(打印机)
2. 不可剥夺/抢占资源：除非本进程释放，别的进程根本抢不走(主存/CPU)

2️⃣根据使用方式：共享/独占资源

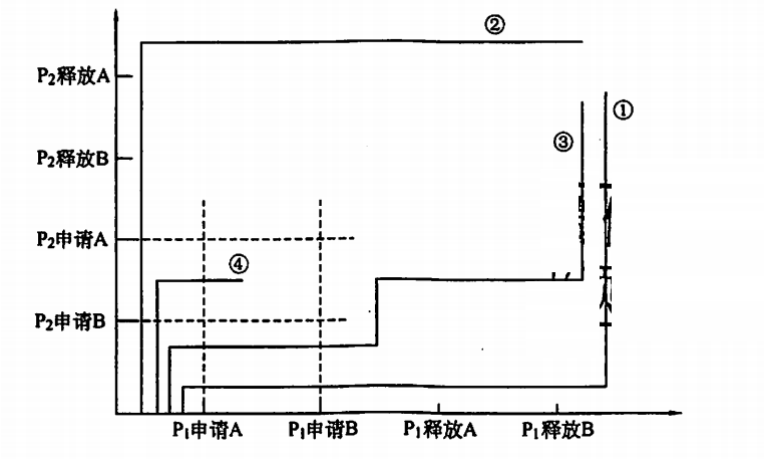
PS—共享资源的获取与释放：请求然后得到资源(不被立即允许时一直等待)→使用→释放

3️⃣根据使用期限：永久资源(无法被消耗，如打印机)+临时资源(可以被消耗殆尽，如内存)

5.2.2. 死锁产生的原因

并发执行的资源竞争+系统资源不足(根本原因)+进程推进顺序不当(直接原因)

如下图只有4(落入虚线方框内)会锁死



5.2.3. 进程锁死的必要条件

1️⃣互斥条件：一段时间内某种资源仅为一个进程占有

2️⃣不剥离条件：资源在未使用完之前，不能被其他进程夺走

3️⃣请求与保持条件：进程申请新资源的同时，继续占有以获得的资源

4️⃣环路等待条件：如下

1. 进程P1已经拥有资源A，但它还需要资源B来继续运行
2. 进程P2已经拥有资源B，但它还需要资源C来继续运行
3. 进程P3已经拥有资源C，但它还需要资源A来继续运行

5.3. 死锁的处理

1️⃣ 鸵鸟算法：不对死锁进行任何处理，如UNIX，降低了系统复杂性/但死锁时会导致资源浪费和响应延长

2️⃣ 死锁预防：设置严格限制来破坏死锁的必要条件，如可剥夺式的进程调度(优先级)，系统是永远不会死锁了/但会对影响并发性的性能

3️⃣ 死锁避免：分配资源时OS会预测接下来会不会死锁(再决定要不要这么分配资源)，如银行家算法，比死锁预防更有利于并发执/但预测操作会增加计算成本

4️⃣ 死锁检测及解除：(被动策略)OS定期检索是否死锁然后在死锁后采取措施，如剥终止or剥夺某进程资源来打破死锁，OS可以在大多数时间内高效地运行，但死锁后代价大

5.4. 死锁的预防：四个必要条件各个击破

1️⃣互斥条件：资源只能给一个进程→能给多个进程，但是这会打破进程固有属性(两个进程公用打印机？)所以不实际

2️⃣不剥夺(非抢占)条件：进程持有资源后就霸占→若该进程新的资源请求不被满足就放弃之前已经有的资源，但是这太复杂还可能造成以前工作作废(打印到一半丢掉)

3️⃣请求与保持条件：进程持有资源后请求其他资源→强制进程一次性申请得到所有资源后再运行，但是这样资源利用率低且进程容易饥饿

4️⃣环路等待条件：将所有的资源类型放入资源列表中，并且要求进程按照资源表申请资源；编号递增申请，但限制了新设备的增加(重写编号)，吞吐量低

5.5. 死锁的避免：相比死锁的预防限制更弱

5.5.1. 安全/不安全状态

1️⃣基本概念

1. 安全状态：系统按某顺序(安全序列，不唯一)为每进程分配其所需资源，保证每个进程都可顺利完成
2. 不安全状态：不存在上述的安全顺序，但是不一定所有不安全状态都有死锁，只是可能(有些资源执行到一半就放弃了)

2️⃣安全状态实例：P1-3共享一个资源，资源总数为10

进程	资源总需求	已分配资源	可用资源
P_1	8	3	4
P_2	4	2	
P_3	9	1	

可用资源按照P2→P1→P3顺序配是安全的(安全序列)，如果可用资源先分给P1就会直接锁死

3️⃣死锁避免的核心：使OS一直处于安全状态之一的状态

分配资源前先计算分配的安全性→确认能安全分配再分配

5.5.2. 资源分配图算法：每种资源只有一个实例

1️⃣请求边与需求边

1. 请求边：实线$< P_i, r_i >$表示P_i请求一个r_i资源且尚未分配
2. 需求边：虚线$< P_i, r_i >$表示P_i可能会请求一个r_i资源
3. 转化：申请资源后虚线转实线，释放资源后实现转虚线

2️⃣死锁预防

把申请边实线$< P_i, r_i >$转化为分配边$< r_i, P_i >$，如果出现环路则不安全，如果不出现环路那么安全可分配

5.5.3. 银行家算法：每种资源可有多个实例

5.5.3.1. 算法中用到的数据结构

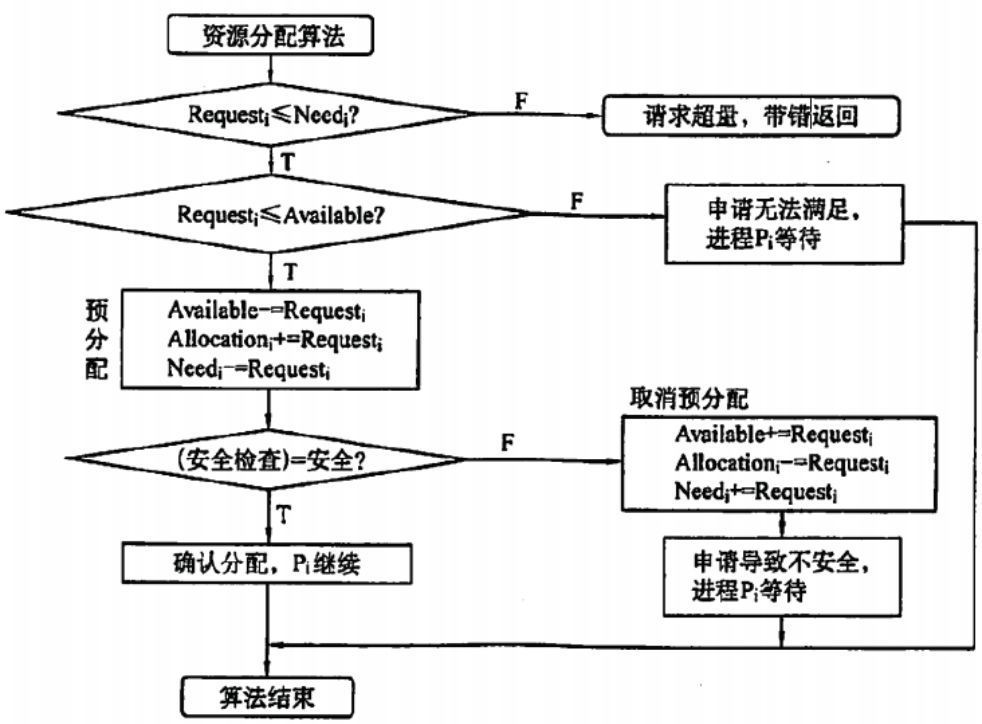
假设有进程$(P_1, P_2 \dots P_n)$and资源$(R_1, R_2 \dots R_m)$，则用到的数据结构

```
Available[i]      //可用资源向量,表示第Ri类资源的现有空闲数量
Request[i][j]     //请求矩阵，表示进程Pi请求的Rj类资源的数量
max[i][j]         //最大需求矩阵，表示进程Pi对Rj类资源最大需求数
Allocation[i][j] //分配矩阵，进程Pi对Rj类资源的持有数
Need[i][j]        //需求矩阵，进程Pi对Rj类资源仍然需要的数目
Need[i]           //Pi的资源需求向量，即所需全部资源类型及数目
```

```
Need[i] [j] = Max[i] [j] - Allocation[i] [j]
```

5.5.3.2. 银行家算法的描述

1️⃣银行家算法



2️⃣ 安全性检测算法：

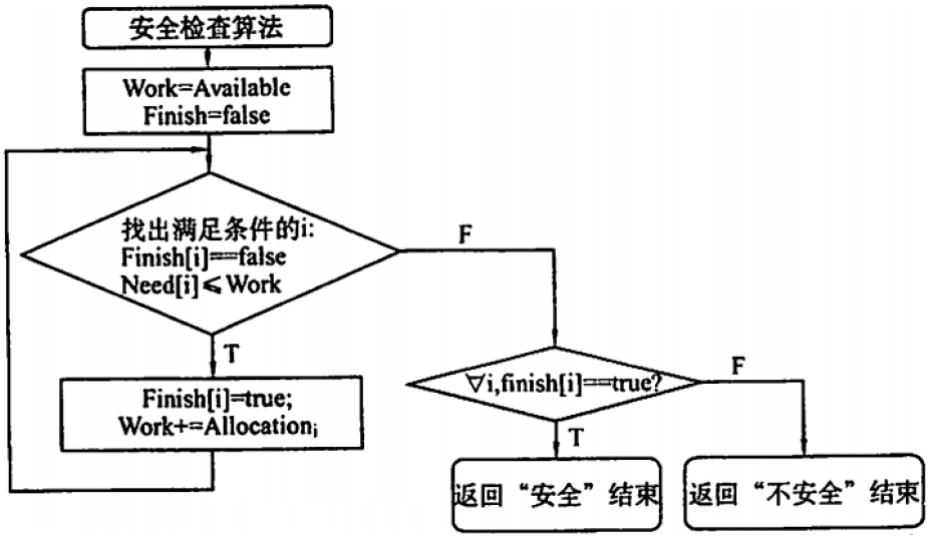
1. 先要建立两个向量Work(可用资源)和Finish(进程结束)

```
Work=Available;
Finish[i]=false; //表示Pi进程还未执行完
```

2. 找到符合条件的进程：未结束+所需资源小于系统可用

2.1. 如果有这样的进程则执行完进程后释放其所有资源Allocation

2.2. 找不到这样的进程的话有两种可能，那就是全部执行完了(安全)，否则不安全



5.5.3.3. 银行家算法实例

1️⃣ 题目

某一时刻系统状态：

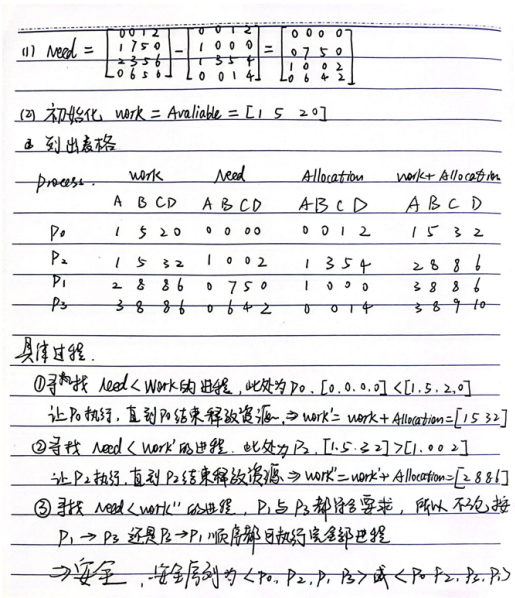
Process	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P_0	0	0	1	2	0	0	1	2	1	5	2	0
P_1	1	0	0	0	1	7	5	0				
P_2	1	3	5	4	2	3	5	6				
P_3	0	0	1	4	0	6	5	6				

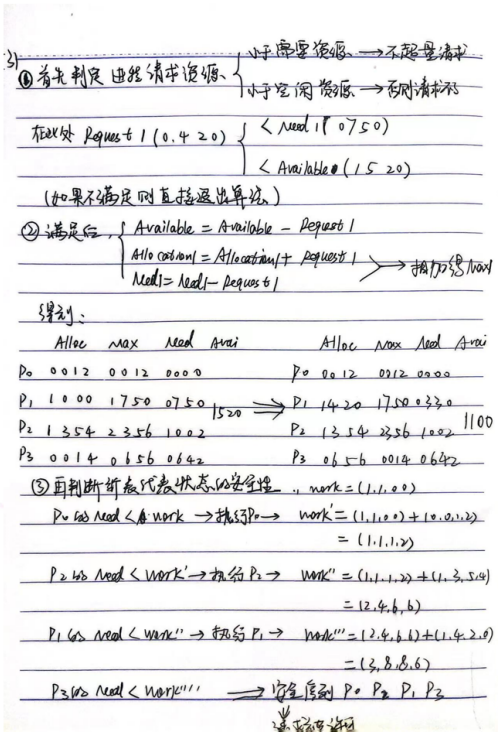
(1) Need = Max矩阵 - Allocation矩阵

(2) 分析系统当前安全性

(3) P_1 发来 Request (0, 4, 2, 0) 是否能被立刻满足？

2 解答





5.6. 死锁的检测

5.6.1. 资源分配图：SRAG=(V,E)的有向图

1 点集：分为两类，每个进程P_i一个点(圆圈)，一类资源r_i一个点(方框)，而一类资源中还可能有多个资源则用方框中的点来表示(每个资源表示一个实例)

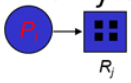
2 有向边集：$< P_i, r_i >$表示P_i请求一个r_i资源且尚未分配(申请边)，$< r_i, P_i >$表示r_i资源中一个资源已经分配给了P_i(分配边)

3 图例：

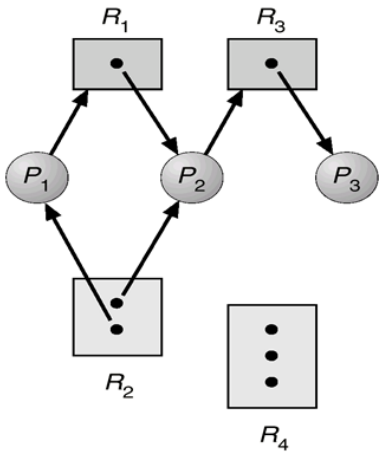
- **Process进程**

- **Resource Type with 4 instances有四个实例的资源类型**

- ***P_i* requests instance of *R_j*（*P_i* *请求一个R_j的实例*）**

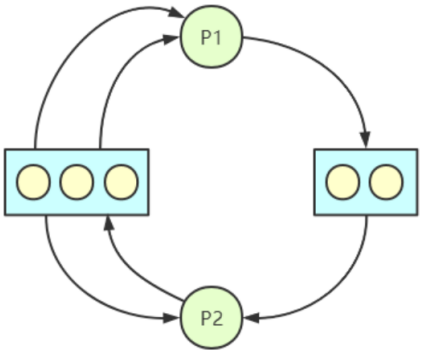


- ***P_i* is holding an instance of *R_j*（*P_i* *持有一个R_j的实例*）**



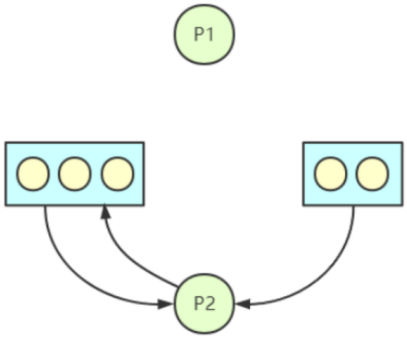
5.6.2. 死锁定理：用SRAG检验系统是否死锁

1 图中的非阻塞进程：首先要有边与进程结点连接(允许仍未结束)，其次该节点申请的资源数要小于等于该类资源的空弦数。如下图的P1就是非阻塞的

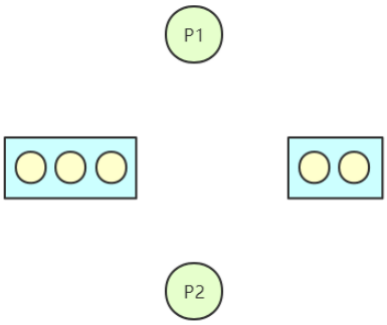


1 可完全简化

1. 找到非阻塞结点，该节点会执行到底然后释放资源，然后孤立。如下图



2. 其他进程因为得到了被释放的资源，也开始执行。在本例中是P2
3. 按照这个规则周而复始，如果最后图中无边则图是可完全简化的，否则得到唯一的不可简化图



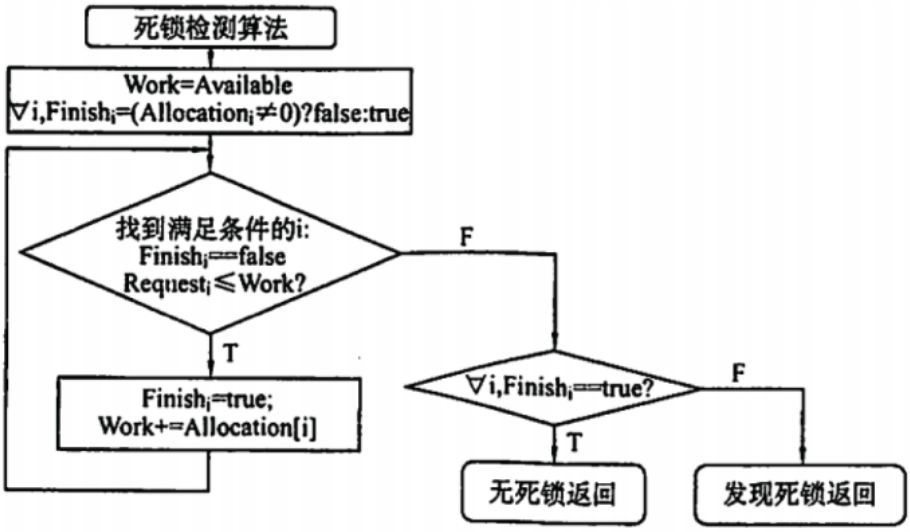
2 死锁定理：状态S是死锁 $\Longleftrightarrow$ S状态的资源分配图不可完全简化

5.6.3. 死锁检测方法

1️⃣死锁的必要性检测：

1. 如果SRAG无环，那么比不可能死锁
2. 如果SEGA有环，每类资源都只有一个实例，则一定死锁。否则见下：

2️⃣死锁的检测(他妈就是死锁定理吧！)：确定是否存在一种方式使所有进程都可以获得所需资源并运行完



1. Work表示当前可用的资源，如果Allocation不为零(分配有资源)则Finish为假(进程不该结束)，反之
2. 然后寻找这样的进程：进程仍未结束，请求的资源小于可用资源
   2.1. 如果有这样的进程就让他执行完然后释放所占资源
   2.2. 没有的话，要么所有进程都执行完了，要么就死锁了

5.7. 死锁的解除

1️⃣剥夺资源：从其他进程处抢来足够资源解除死锁

2️⃣撤销进程：灭掉一些进程为其他进程提供更多资源

3️⃣进程回退：根据记录信息，进程回到死锁前(自愿放弃资源)

PS. 活锁/饥饿/饿死

1️⃣饥饿：进程长时间等待

2️⃣饿死：进程等待时间过长，即使得到资源执行了也无意义了

3️⃣活锁：特殊的一种饥饿，进程在执行但是无法被调度前进，像被死锁了一样