死锁

死锁 饥饿 死循环

死锁是等待对方手里资源导致的 如果发生死锁，那么至少有两个或者两个以上的进程发生死锁，并且一定处于阻塞态

可能只有一个进程发生饥饿 可能是长时间阻塞态即得不到io设备，也可能是就绪态，即得不到cpu

死循环可以上处理机运行的运行态 可能不是代码逻辑错误产生的，可能是故意的

死锁发生四条件 只要有一个不成立 死锁不发生

1 互斥条件 只有对必须互斥使用的资源争抢的时候才会导致死锁，像内存，扬声器这样可以同时让多个进程使用的资源是不会死锁的

2 不可剥夺条件 进程所获得的资源在未使用完之前，不能由其他进程强行夺走，只能主动释放。

3 请求和保持条件 进程已经至少持有一个资源，但又提出了新的资源请求，该资源又被其他资源占有，此时请求进程被阻塞 但又对自己持有的资源保持不放

4 循环等待条件 存在一种资源的循环等待链 链中的每一个进程已获得的资源同时被下一个进程所请求

循环等待不一定出现死锁，但是死锁一定出现循环等待

也就是循环等待是死锁的必要补充饭条件

但是如果同类资源只有一种 那么就是充分必要条件

当对不可剥夺资源分配不合理可能导致死锁

# 预防死锁

想办法破坏死锁产生的一个或者几个必要条件

1. 破坏互斥资源：将有限的互斥资源改进为共享的资源SPOOLing技术
2. 破坏不可剥夺条件:
   1. 方案1：当进程得不到需要资源的时候立刻放弃所有资源
   2. 方案2: 当更高优先级进程需要被占用的资源的时候，操作系统会帮助抢夺资源，比如cpu优先级调度、

缺点：实现复杂/可能让进程前一段时间的工作失效 并且增加系统开销，降低吞吐量，可能导致饥饿

1. 破坏请求和保持条件：采用静态分配方法，也就是进程运行之前就申请所需要的所有资源，如果有的资源申请不到，那么就不运行 缺点 对于运行时间短的进程导致浪费，并且可以造成饥饿
2. 破坏循环等待条件

将所有的资源编号，只有申请了小编号资源的进程才能申请大编号的资源，并且大编号资源不能反过来申请小编号的资源，这样就防止了循环等待条件

1.不方便

2.进程的使用顺序与编号不一致，导致资源的浪费

3.用户编程麻烦

# 避免死锁

什么是安全序列

如果系统能找到安全序列，那么系统就是出于安全状态，这种情况下就一定不会发生死锁

如果系统进入不安全状态，那么就有可能发生死锁，但是并不一定

进入不安全状态不一定发生死锁，但是发生死锁一定出于不安全状态

也就是不安全状态是发生死锁的必要条件不充分

思路：分配资源之前预先判断一下这次分配是否会让系统进入不安全状态 如果不会，就分配

如何扩展多种资源，即将单维的资源拓展为多维的向量 如：资源总数(10,5,7)

|  |  |  |  |
| --- | --- | --- | --- |
| 进程 | 最大需求 | 已分配 | 最多还需要 |
| P0 | (7,5,3) | (0,1,0) | (7,4,3) |
| 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) |

共分配(7,2,5),剩余(10,5,7)-(7,2,5)=(3,3,2)

这时候只有P1,P3可以满足，如果满足P1，那么P1执行之后会返回(3,2,2)的资源 这时候共有（5，3，2）的资源

以此类推，p1，p3可以先加入安全序列，P1,P3都用完之后 系统资源变为(7,4,3)，并且743可以满足所有的需求，那么就强所有的进程都加入安全序列中，这样的安全序列不出现死锁。

# 银行家算法

假设现在有n个进程m种资源，设置一个最大需求的n\*m矩阵，记录当前最大的资源数量，设置一个n\*m的allocation矩阵，表示已经分配的资源数量，设置一个n\*m的need矩阵，记录还需要多少资源，并且设置available数组用来记录还有多少资源可用，设置request数组，记录请求的资源量，具体算法如下:  
（1）当有request请求的时候，将request i[j]与need[i][j]比较，如果request i[j]>need[i][j]说明资源请求超出了预定需求，错误，如果小于，则进行第二步

(2)将request i[j]与available[i][j]进行比较，如果request大于说明系统暂时没有这个资源可用，要等待，如果小于则下一步

(3)虚拟分配空间available[i][j]-request i[j],allocation[i][j]+request i[j],need[i][j]-request[i][j]

(4)进行安全序列判定，即如果need矩阵里有小于available的行，那么就将其资源数组加到available，继续对比，直到所以资源回归位置，如果不能回归不给分配。

# 死锁的检测和解除

资源分配图

进程结点指向资源结点，表示请求的资源，资源节点指向进程节点，表示资源已经分配的个数。

如果资源分配图可以完全简化，那么就不会发生死锁

（检测：先找到一个不阻塞也不孤立的结点pi，去掉所有边后继续找，直到不能再消去，就是死锁进程 ）

解决方法

1. 资源剥夺
2. 撤销进程
3. 进程回退 设置还原点