# 死锁与银行家算法

## 死锁产生的四个条件

（1） **互斥条件**
一个资源每次只能被一个进程使用。
（2） **请求与保持条件**
一个进程因请求资源而阻塞时，对已获得的资源保持不放。
（3） **不剥夺条件**
进程已获得的资源，在末使用完之前，不能强行剥夺。
（4） **循环等待条件**
若干进程之间形成一种头尾相接的循环等待资源关系。

## 死锁与饥饿的区别

资源的分配策略不公，导致某些进程很长时间无法获得资源，以至于出现为了获取资源而出现等待时间无预期上届的情况。

例子一：当有多个进程需要打印文件时，如果系统分配打印机的策略是最短文件优先，那么长文件的打印任务将由于短文件的源源不断到来而被无限期推迟，导致最终的饥饿甚至饿死。

例子二：某些 CPU 优先等级低的进程永远无法获得运行的机会。

## 银行家算法 – 安全判定算法与资源请求算法

**安全序列：**对当前申请资源的进程排出一个序列，保证按照这个序列分配资源完成进程

**数据结构：**

```
int n,m;                    //系统中进程总数 n 和资源种类总数 m
int Available[1..m];        //资源当前可用总量
int Allocation[1..n,1..m]; //当前给分配给每个进程的各种资源数量
int Need[1..n,1..m];        //当前每个进程还需分配的各种资源数量
int Work[1..m];             //当前可分配的资源
bool Finish[1..n];          //进程是否结束
```

**安全判定算法执行过程：**

**1．初始化**
Work = Available（动态记录当前剩余资源）
Finish[i] = false（设定所有进程均未完成）

**2．查找可执行线程 Pi**
Finish[i] = false
Need[i] <= Work
如果没有这样的进程 Pi，则跳转到第 4 步

**3．（若有则）Pi 一定能完成，并归还其占用的资源**
Finish[i] = true
Work = Work +Allocation[i]
GOTO 第 2 步，继续查找

**4．如果所有进程 Pi 都是能完成的，即 Finish[i]=ture 则系统处于安全状态，否则系统处于不安全状态**

**安全判定算法伪代码：**

```
Boolean Found;
Work = Available; Finish[1..n] = false;
while(true){
    //不断的找可执行进程
    Found = false;
    for(i=1; i<=n; i++){
        if(Finish[i]==false && Need[i]<=Work){
            Work = Work + Allocation[i];//把放出去的贷款也当做自己的资产
            Finish[i] = true;
            Found = true;
        }
    }
    if(Found==false)
        break;
}
for(i=1;i<=n;i++)
    if(Finish[i]==false)
        return “deadlock”; //如果有进程是完不成的，那么就是有死锁
```

**资源请求算法：**

第 i 个进程请求的资源数记为 Requests[i]

**1.检查累积资源获取量是否超过最大值**

如果 Requests[i]<=Need[i]，则转到第二步。否则，返回异常。

这一步是控制进程申请的资源不得大于需要的资源

**2.检查资源是否足够分配**

如果 Requests[i]<=Available，则转到第三步，否则 Pi 等待资源

**3.如果满足前两步，那么做如下操作：**

```
Available = Available -Requests[i]
Allocation = Allocation[i]+Requests[i]
Need[i]=Need[i]-Requests[i]
调用安全判定算法，检查是否安全
if(安全){
    申请成功，资源分配
}else{
    申请失败，资源撤回。第三步前几个操作进行逆操作
}
```