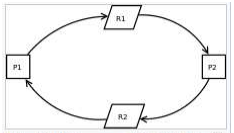
## 死锁与饥饿的定义：



P1、P2两个process都需要资源才能继续执行。P1拥有资源R2、还需要额外资源R1才能执行；P2拥有资源R1、还需要额外资源R2才能执行，两边都在互相等待而没有任何一个可执行。

**死锁**（英语：Deadlock），又译为死结，计算机科学名词。当两个以上的运算单元，双方都在等待对方停止执行，以取得系统资源，但是没有一方提前退出时，就称为死结。在多工作业系统中，作业系统为了协调不同行程，能否取得系统资源时，为了让系统运作，必须要解决这个问题。

这里指的是进程死锁，是个计算机技术名词。它是操作系统或软件运行的一种状态：在多工系统下，当一个或多个进程等待系统资源，而资源又被进程本身或其他进程占用时，就形成了死锁。(来自维基百科)

In concurrent computing, a **deadlock** is a state in which each member of a group is waiting for some other member to take action, such as sending a message or more commonly releasing a lock.[1] Deadlock is a common problem in multiprocessing systems, parallel computing, and distributed systems, where software and hardware locks are used to handle shared resources and implement process synchronization.[2]

In an operating system, a deadlock occurs when a process or thread enters a waiting state because a requested system resource is held by another waiting process, which in turn is waiting for another resource held by another waiting process. If a process is unable to change its state indefinitely because the resources requested by it are being used by another waiting process, then the system is said to be in a deadlock.[3] (From Wikipedia)

**进程饥饿**（英语：**Starvation**），指当等待时间给进程推进和响应带来明显影响称为进程饥饿。当饥饿到一定程度的进程在等待到即使完成也无实际意义的时候称为饥饿死亡。亦有定义为：由于别的并发的激活的过程持久占有所需资源，使某个异步过程载客预测的时间内不能被激活。(来自百度百科)

In computer science, **starvation** is a problem encountered in concurrent computing where a process is perpetually denied necessary resources to process its work.[1] Starvation may be caused by errors in a scheduling or mutual exclusion algorithm, but can also be caused by resource leaks, and can be intentionally caused via a denial-of-service attack such as a fork bomb. (From Wikipedia)

## 解释死锁与进程：

**死锁：每个进程都守着自己的资源不放，都想获取别的进程所获得自己需要的资源，谁都不肯投降，从而一直僵持下去，没有结果。**

**形象比喻：**迎面开来的汽车 A 和汽车 B 过马路，汽车 A 得到了半条路的资源（满足死锁发生条件 1：资源访问是排他性的，我占了路你就不能上来，除非你爬我头上去），汽车 B 占了汽车 A 的另外半条路的资源，A 想过去必须请求另一半被 B 占用的道路（死锁发生条件 2：必须整条车身的空间才能开过去，我已经占了一半，尼玛另一半的路被 B 占用了），B 若想过去也必须等待 A 让路，A 是辆兰博基尼，B 是开奇瑞 QQ 的屌丝，A 素质比较低开窗对 B 狂骂：快给老子让开，B 很生气，你妈逼的，老子就不让（死锁发生条件 3：在未使用完资源前，不能被其他线程剥夺），于是两者相互僵持一个都走不了（死锁发生条件 4：环路等待条件），而且导致整条道上的后续车辆也走不了。

**饿死：顾名思义，就是想要获取的资源一直得不到，无限等待，到了自己所承受的最大能力，从而饿死，此时就算得到了资源，也毫无意义。**

**形象比喻：**在“首堵”北京的某一天，天气阴沉，空气中充斥着雾霾和地沟油的味道，某个苦逼的临时工交警正在处理塞车，有两条道 A 和 B 上都堵满了车辆，其中 A 道堵的时间最长，B 相对相对堵的时间较短，这时，前面道路已疏通，交警按照最佳分配原则，示意 B 道上车辆先过，B 道路上过了一辆又一辆，A 道上排队时间最长的确没法通过，只能等 B 道上没有车辆通过的时候再等交警发指令让 A 道依次通过，这也就是 ReentrantLock 显示锁里提供的不公平锁机制，不公平锁能够提高吞吐量但不可避免的会造成某些线程的饥饿。

有三类资源 A(17)、B(5)、C(20)。有 5 个进程 $P_1$—$P_5$。$T_0$ 时刻系统状态如下：

| | 最大需求 | 已分配 |
|---|---|---|
| $P_1$ | 5 5 9 | 2 1 2 |
| $P_2$ | 5 3 6 | 4 0 2 |
| $P_3$ | 4 0 11 | 4 0 5 |
| $P_4$ | 4 2 5 | 2 0 4 |
| $P_5$ | 4 2 4 | 3 1 4 |

**(1)$T_0$ 时刻是否为安全状态，给出安全系列。**

**(2)$T_0$ 时刻，$P_2$: Request(0,3,4)，能否分配，为什么？**

**(3)在(2)的基础上 $P_4$：Request(2,0,1)，能否分配，为什么？**

**(4)在(3)的基础上 $P_1$：Request(0,2,0)，能否分配，为什么？**

| | Max | | | Allocation | | | Need | | | Available | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | A | B | C | A | B | C | A | B | C | A | B | C |
| $P_1$ | 5 | 5 | 9 | 2 | 1 | 2 | 3 | 4 | 7 | 2 | 3 | 3 |
| $P_2$ | 5 | 3 | 6 | 4 | 0 | 2 | 1 | 3 | 4 | | | |
| $P_3$ | 4 | 0 | 11 | 4 | 0 | 5 | 0 | 0 | 6 | | | |
| $P_4$ | 4 | 2 | 5 | 2 | 0 | 4 | 2 | 2 | 1 | | | |
| $P_5$ | 4 | 2 | 4 | 3 | 1 | 4 | 1 | 1 | 0 | | | |

**$T_0$ 时刻的资源分配表**

解答：

**(1) $T_0$ 时刻是否为安全状态，给出安全系列。**

$T_0$ 时刻的安全性：利用安全性算法对 $T_0$ 时刻的资源分配情况进行分析可知，在 $T_0$ 时刻存在着一个安全序列{$P_4$，$P_2$，$P_3$，$P_5$，$P_1$}，故系统是安全的，如下表所示。

| | Max | | | Need | | | Allocation | | | Work+Allocation | | | Finish |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | A | B | C | A | B | C | A | B | C | A | B | C | |
| $P_4$ | 2 | 3 | 3 | 2 | 2 | 1 | 2 | 0 | 4 | 4 | 3 | 7 | True |
| $P_2$ | 4 | 3 | 7 | 1 | 3 | 4 | 4 | 0 | 2 | 8 | 3 | 9 | True |
| $P_3$ | 8 | 3 | 9 | 0 | 0 | 6 | 4 | 0 | 5 | 12 | 3 | 14 | True |
| $P_5$ | 12 | 3 | 14 | 1 | 1 | 0 | 3 | 1 | 4 | 15 | 4 | 18 | True |
| $P_1$ | 15 | 4 | 18 | 3 | 4 | 7 | 2 | 1 | 2 | 17 | 5 | 20 | True |

**(2) $T_0$ 时刻，$P_2$: Request(0, 3, 4)，能否分配，为什么？**

$P_2$ 请求资源：$P_2$ 发出请求向量 $Request_2(0,3,4)$，系统按银行家算法进行检查：

①$Request_2(0, 3, 4) \leqslant Need_2(1, 3, 4)$;

②$Request_2(0, 3, 4) > Available(2, 3, 3)$，让 $P_2$ 等待。

**(3) 在(2)的基础上 $P_4$：Request(2,0,1)，能否分配，为什么？**

$P_4$ 请求资源：$P_4$ 发出请求向量 $Request_4(2,0,1)$，系统按银行家算法进行检查：

①$Request_4(2, 0, 1) \leq Need_4(2, 2, 1)$

②$Request_4(2, 0, 1) \leq Available_4(2, 3, 3)$

③系统先假定可为 $P_4$ 分配资源，并修改 Available, $Allocation_4$ 和 $Need_4$ 向量，由此形成的资源变化情况如下图 中的圆括号所示。

| | Max | | | Allocation | | | Need | | | Available | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | A | B | C | A | B | C | A | B | C | A | B | C |
| $P_1$ | 5 | 5 | 9 | 2 | 1 | 2 | 3 | 4 | 7 | 2 | 3 | 3 |
| $P_2$ | 5 | 3 | 6 | 4 | 0 | 2 | 1 | 3 | 4 | (0 | 3 | 2) |
| $P_3$ | 4 | 0 | 11 | 4 | 0 | 5 | 0 | 0 | 6 | | | |
| $P_4$ | 4 | 2 | 5 | 2 | 0 | 4 | 2 | 2 | 1 | | | |
| | | | | (4 | 0 | 5) | (0 | 2 | 0) | | | |
| $P_5$ | 4 | 2 | 4 | 3 | 1 | 4 | 1 | 1 | 0 | | | |

④再利用安全性算法检查此时系统是否安全。

| | Work | | | Need | | | Allocation | | | Work+Allocation | | | Finish |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | A | B | C | A | B | C | A | B | C | A | B | C | |
| $P_4$ | 0 | 3 | 2 | 0 | 2 | 0 | 4 | 0 | 5 | 4 | 3 | 7 | True |
| $P_2$ | 4 | 3 | 7 | 1 | 3 | 4 | 4 | 0 | 2 | 8 | 3 | 9 | True |
| $P_3$ | 8 | 3 | 9 | 0 | 0 | 6 | 4 | 0 | 5 | 12 | 3 | 14 | True |
| $P_5$ | 12 | 3 | 14 | 1 | 1 | 0 | 3 | 1 | 4 | 15 | 4 | 18 | True |
| $P_1$ | 15 | 4 | 18 | 3 | 4 | 7 | 2 | 1 | 2 | 17 | 5 | 20 | True |

由所进行的安全性检查得知，可以找到一个安全序列 {$P_4$，$P_2$，$P_3$，$P_5$，$P_1$}。因此，系统是安全的，可以立即将 $P_4$ 所申请的资源分配给它。

**(4) 在(3)的基础上 $P_1$：Request(0,2,0)，能否分配，为什么？**

$P_1$ 请求资源：$P_1$ 发出请求向量 $Requst_1(0，2，0)$，系统按银行家算法进行检查：

①$Request_1(0, 2, 0) \leq Need_1(3, 4, 7)$;

②$Request_1(0, 2, 0) \leq Available(0, 3, 2)$;

③系统暂时先假定可为 $P_1$ 分配资源，并修改有关数据，如下图所示。

| | Max | | | Allocation | | | Need | | | Available | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | A | B | C | A | B | C | A | B | C | A | B | C |
| $P_1$ | 5 | 5 | 9 | 2 | 3 | 2 | 3 | 2 | 7 | 0 | 1 | 2 |
| $P_2$ | 5 | 3 | 6 | 4 | 0 | 2 | 1 | 3 | 4 | | | |
| $P_3$ | 4 | 0 | 11 | 4 | 0 | 5 | 0 | 0 | 6 | | | |
| $P_4$ | 4 | 2 | 5 | 4 | 0 | 5 | 0 | 2 | 0 | | | |
| $P_5$ | 4 | 2 | 4 | 3 | 1 | 4 | 1 | 1 | 0 | | | |

④进行安全性检查：可用资源 Available(0,1,2)已经不能满足任何进程的需要，故系统进入不安全状态，此时系统不分配资源。