# 锁机制

在多线程编程中，操作系统引入了锁机制。通过锁机制，能够**保证在多核多线程环境中，在某一个时间点上，只能有一个线程进入临界区代码，从而保证临界区中操作数据的一致性**。

## 死锁

是指**两个或两个以上的进程在执行过程中,因争夺资源而造成的一种互相等待的现象**,**若无外力作用,它们都将无法推进下去**。此时称系统处于死锁状态或系统产生了死锁,这些永远在互相等待的进程称为死锁进程。

## 产生死锁的原因

1. 因为**系统资源不足**。
2. 进程运行**推进的顺序不当**。
3. **资源分配不当**等。

如果系统资源充足，进程的资源请求都能够得到满足，死锁出现的可能性就很低，否则

就会因**争夺有限的资源而陷入死锁**。其次，**进程运行推进顺序与速度不同**，也可能产生死锁。

## 死锁的四个必要条件

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，其它线程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。

这四个条件是死锁的必要条件，只要系统发生死锁，这些条件必然成立，而只要上述条件之

一不满足，就不会发生死锁。

## 死锁的解除与预防

<http://blog.csdn.net/abigale1011/article/details/6450845/>

理解了死锁的原因，尤其是产生死锁的四个必要条件，就可以最大可能地避免、预防和解除死锁。所以，在系统设计、进程调度等方面**注意如何不让这四个必要条件成立**，如何**确定资源的合理分配算法**，避免进程永久占据系统资源。此外，也要防止进程在处于等待状态

的情况下占用资源。因此，对资源的分配要给予合理的规划。

如何避免死锁：

<http://mp.weixin.qq.com/s?__biz=MjM5NzMyMjAwMA==&mid=2651479577&idx=2&sn=448e59a49ccc9dacff60e0dbd5ced113&chksm=bd2532668a52bb702e253175868aa479994aaa09363ed5cbf192cb00b4489c62b7c6d999a207&mpshare=1&scene=23&srcid=1207UAXOL8zDTAM9ln48x2Tr#rd>

### 加锁顺序

当多个线程需要相同的一些锁，但是按照不同的顺序加锁，死锁就很容易发生。

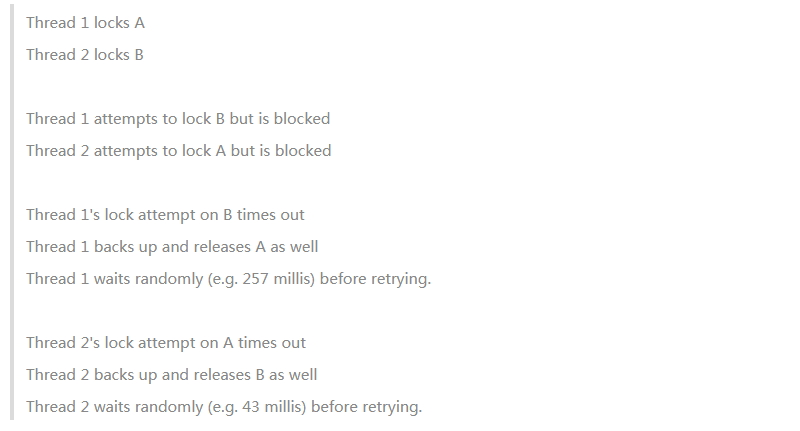
如果能确保所有的线程都是按照相同的顺序获得锁，那么死锁就不会发生。

按照顺序加锁是一种有效的死锁预防机制。但是，这种方式需要你事先知道所有可能会用到的锁(并对这些锁做适当的排序)，但总有些时候是无法预知的。



### 加锁时限

在尝试获取锁的时候加一个超时时间，这也就意味着在尝试获取锁的过程中若超过了这个时限该线程则放弃对该锁请求。若一个线程没有在给定的时限内成功获得所有需要的锁，则会进行回退并释放所有已经获得的锁，然后等待一段随机的时间再重试。这段随机的等待时间让其它线程有机会尝试获取相同的这些锁，并且让该应用在没有获得锁的时候可以继续运行(加锁超时后可以先继续运行干点其它事情，再回头来重复之前加锁的逻辑)。



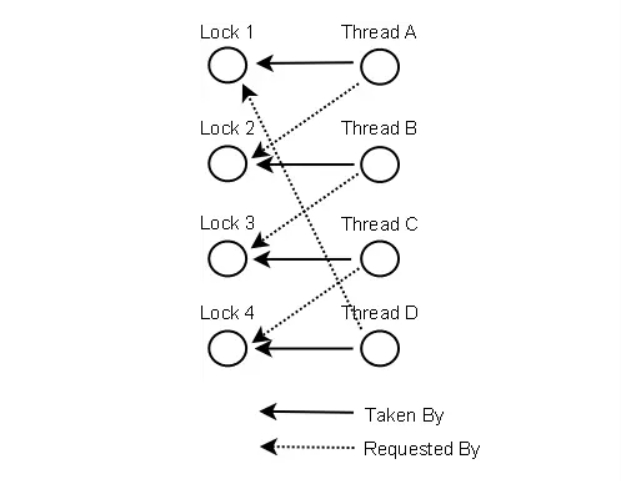
超时和重试机制是为了避免在同一时间出现的竞争，但是当线程很多时，其中两个或多个线程的超时时间一样或者接近的可能性就会很大，因此就算出现竞争而导致超时后，由于超时时间一样，它们又会同时开始重试，导致新一轮的竞争，带来了新的问题。

### 死锁检测

死锁检测是一个更好的死锁预防机制，它主要是针对那些不可能实现按序加锁并且锁超时也不可行的场景。

每当一个线程获得了锁，会在线程和锁相关的数据结构中（map、graph等等）将其记下。除此之外，每当有线程请求锁，也需要记录在这个数据结构中。

**当一个线程请求锁失败时，这个线程可以遍历锁的关系图看看是否有死锁发生**。



给这些线程设置优先级，让一个（或几个）线程回退，剩下的线程就像没发生死锁一样继续保持着它们需要的锁。如果赋予这些线程的优先级是固定不变的，同一批线程总是会拥有更高的优先级。为避免这个问题，可以在死锁发生的时候设置随机的优先级。

# 线程调度