死锁

一些概念

可抢占与不可抢占资源

大部分的死锁都和资源有关，在进程对设备、文件具有独占性（排他性）时会产生死锁。我们把这类需要排他性使用的对象称为**资源(resource)**。资源主要分为 **「可抢占资源和不可抢占资源」**

可抢占资源和不可抢占资源

资源主要有可抢占资源和不可抢占资源。**可抢占资源(preemptable resource)** 可以从拥有它的进程中抢占而不会造成其他影响，内存就是一种可抢占性资源，任何进程都能够抢先获得内存的使用权。

不可抢占资源(nonpreemtable resource) 指的是除非引起错误或者异常，否则进程无法抢占指定资源，这种不可抢占的资源比如有光盘，在进程执行调度的过程中，其他进程是不能得到该资源的。

死锁概念和产生原因

死锁是指多个进程循环等待彼此占有的资源而无限期的僵持等待下去的局面。原因是：

- 系统提供的资源太少了，远不能满足并发进程对资源的需求
- 进程推进顺序不合适，互相占有彼此需要的资源，同时请求对方占有的资源，往往是程序设计不合理

死锁模型

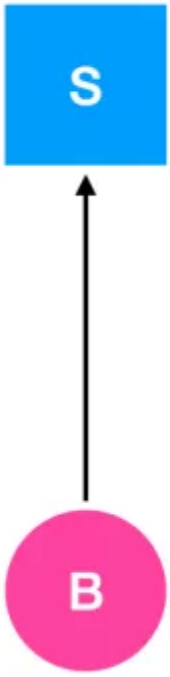
Holt 在 1972 年提出对死锁进行建模，建模的标准如下：

- 圆形表示进程
- 方形表示资源

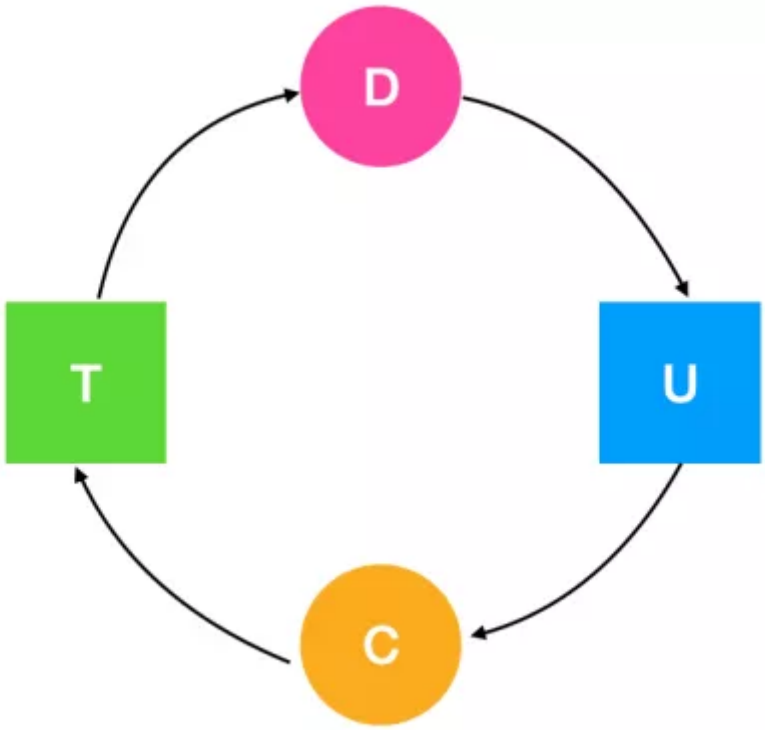
从资源节点到进程节点表示资源已经被进程占用，如下图所示

在上图中表示当前资源 R 正在被 A 进程所占用

由进程节点到资源节点的有向图表示当前进程正在请求资源，并且该进程已经被阻塞，处于等待这个资源的状态



在上图中，表示的含义是进程 B 正在请求资源 S 。Holt 认为，死锁的描述应该如下



这是一个死锁的过程，进程 C 等待资源 T 的释放，资源 T 却已经被进程 D 占用，进程 D 等待请求占用资源 U ，资源 U 却已经被线程 C 占用，从而形成环。

死锁的条件

针对我们上面的描述，资源死锁可能出现的情况主要有

- **互斥条件**：每个资源都被分配给了一个进程或者资源是可用的
- **保持和等待条件**：已经获取资源的进程被认为能够获取新的资源
- **不可抢占条件**：分配给一个进程的资源不能强制的从其他进程抢占资源，它只能由占有它的进程显示释放

- **循环等待**：死锁发生时，系统中一定有两个或者两个以上的进程组成一个循环，循环中的每个进程都在等待下一个进程释放的资源。

发生死锁时，上面的情况必须同时会发生。如果其中任意一个条件不会成立，死锁就不会发生。可以通过破坏其中任意一个条件来破坏死锁，下面这些破坏条件就是我们探讨的重点

有四种处理死锁的策略：

- 忽略死锁带来的影响

> ## 鸵鸟算法
>
> 最简单的解决办法就是使用**鸵鸟算法(ostrich algorithm)**，把头埋在沙子里，假装问题根本没有发生。每个人看待这个问题的反应都不同。数学家认为死锁是不可接受的，必须通过有效的策略来防止死锁的产生。工程师想要知道问题发生的频次，系统因为其他原因崩溃的次数和死锁带来的严重后果。如果死锁发生的频次很低，而经常会由于硬件故障、编译器错误等其他操作系统问题导致系统崩溃，那么大多数工程师不会修复死锁。

- 通过破坏死锁产生的四个条件之一来避免死锁
- 通过仔细分配资源来避免死锁
- 检测死锁并回复死锁，死锁发生时对其进行检测，一旦发生死锁后，采取行动解决问题

死锁预防

死锁本质上是无法避免的，因为它需要获得未知的资源和请求，但是死锁是满足四个条件后才出现的，它们分别是

- 互斥
- 保持和等待
- 不可抢占
- 循环等待

我们分别对这四个条件进行讨论，按理说破坏其中的任意一个条件就能够破坏死锁

破坏互斥条件

我们首先考虑的就是**「破坏互斥使用条件」**。如果资源不被一个进程独占，那么死锁肯定不会产生。如果两个打印机同时使用一个资源会造成混乱，打印机的解决方式是使用 **假脱机打印机(spooling printer)**，这项技术可以允许多个进程同时产生输出，在这种模型中，实际请求打印机的唯一进程是打印机守护进程，也称为后台进程。后台进程不会请求其他资源。我们可以消除打印机的死锁。

后台进程通常被编写为能够输出完整的文件后才能打印，假如两个进程都占用了假脱机空间的一半，而这两个进程都没有完成全部的输出，就会导致死锁。

因此，尽量做到尽可能少的进程可以请求资源。

破坏保持等待的条件

第二种方式是如果我们能阻止持有资源的进程请求其他资源，我们就能够消除死锁。一种实现方式是让所有的进程开始执行前请求全部的资源。如果所需的资源可用，进程会完成资源的分配并运行到结束。如果有任何一个资源处于频繁分配的情况，那么没有分配到资源的进程就会等待。

很多进程**「无法在执行完成前就知道到底需要多少资源」**，如果知道的话，就可以使用银行家算法；还有一个问题是这样**「无法合理有效利用资源」**。

还有一种方式是进程在请求其他资源时，先释放所占用的资源，然后再尝试一次获取全部的资源。

破坏不可抢占条件

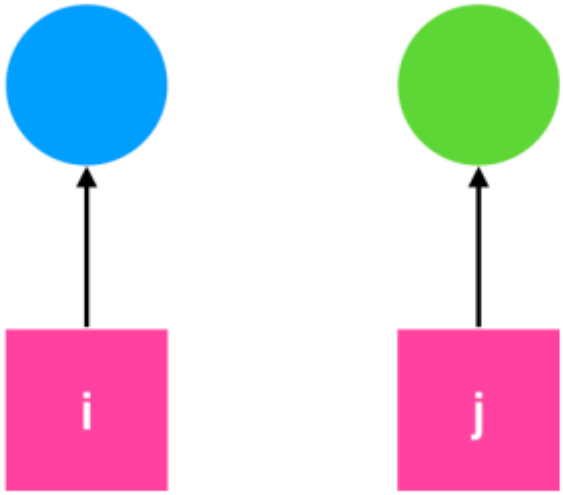
破坏不可抢占条件也是可以的。可以通过虚拟化的方式来避免这种情况。

破坏循环等待条件

现在就剩最后一个条件了，循环等待条件可以通过多种方法来破坏。一种方式是制定一个标准，一个进程在任何时候只能使用一种资源。如果需要另外一种资源，必须释放当前资源。对于需要将大文件从磁带复制到打印机的过程，此限制是不可接受的。

另一种方式是将所有的资源统一编号，如下图所示

进程可以在任何时间提出请求，但是所有的请求都必须按照资源的顺序提出。如果按照此分配规则的话，那么资源分配之间不会出现环。



尽管通过这种方式来消除死锁，但是编号的顺序不可能让每个进程都会接受。

死锁避免

安全状态与不安全状态

安全状态指系统能按某种进程顺序来为每个进程分配其所需资源，直至最大需求，使每个进程都可顺利完成。若系统不存在这样一个序列，则称系统处于不安全状态。

单个资源的银行家算法

多个资源的银行家算法

死锁检测

第二种技术是死锁的检测和恢复。这种解决方式不会尝试去阻止死锁的出现。相反，这种解决方案会希望死锁尽可能的出现，在监测到死锁出现后，对其进行恢复。下面我们就来探讨一下死锁的检测和恢复的几种方式

每种类型一个资源的死锁检测方式

每种资源类型都有一个资源是什么意思？我们经常提到的打印机就是这样的，资源只有打印机，但是设备都不会超过一个。

可以通过构造一张资源分配表来检测这种错误，比如我们上面提到的

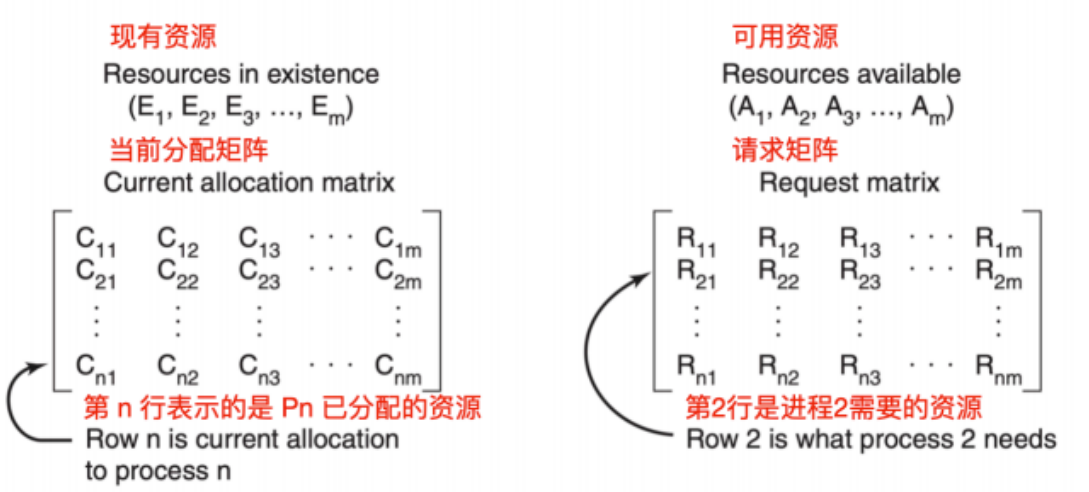
如果这张图包含了一个或一个以上的环，那么死锁就存在，处于这个环中任意一个进程都是死锁的进程。

每种类型多个资源的死锁检测方式

如果有多种相同的资源存在，就需要采用另一种方法来检测死锁。可以通过构造一个矩阵来检测从 P1 -> Pn 这 n 个进程中的死锁。

现在我们提供一种基于矩阵的算法来检测从 P1 到 Pn 这 n 个进程中的死锁。假设资源类型为 m，E1 代表资源类型1，E2 表示资源类型 2 ，Ei 代表资源类型 i (1 <= i <= m)。E 表示的是 **现有资源向量 (existing resource vector)**，代表每种已存在的资源总数。

现在我们就需要构造两个数组：C 表示的是**当前分配矩阵(current allocation matrix)** ，R 表示的是 **请求矩阵(request matrix)**。Ci 表示的是 Pi 持有每一种类型资源的资源数。所以，Cij 表示 Pi 持有资源 j 的数量。Rij 表示 Pi 所需要获得的资源 j 的数量



一般来说，已分配资源 j 的数量加起来再和所有可供使用的资源数相加 = 该类资源的总数。

$$\sum_{i=1}^{n} C_{ij} + A_j = E_j$$

死锁检测算法如下：

- 1）寻找一个没有标记的进程风，对于它而言R矩阵的第 i行向量小于或等于A 。
- 2）如果找到了这样一个进程，那么将C矩阵的第i行向量加到A 中，标记该进程，并转到第 l 步。

- 3）如果没有这样的进程，那么算法终止。

算法结束时，所有没有标记过的进程（如果存在的话）都是死锁进程。
算法的第1步是寻找可以运行完毕的进程，该进程的特点是它有资源请求并且该请求可被当前的可用资源满足。这一选中的进程随后就被运行完毕，在这段时间内它释放自己持有的所有资源并将它们返回到可用资源库中。然后，这一进程被标记为完成。如果所有的进程最终都能运行完毕的话，就不存在死锁的情况。如果其中某些进程一直不能运行，那么它们就是死锁进程。虽然算法的运行过程是不确定的（因为进程可按任何行得通的次序执行），但结果总是相同的。

上面我们探讨了两种检测死锁的方式，那么现在你知道怎么检测后，你何时去做死锁检测呢？一般来说，有两个考量标准：

- 每当有资源请求时就去检测，这种方式会占用昂贵的 CPU 时间。
- 每隔 k 分钟检测一次，或者当 CPU 使用率降低到某个标准下去检测。考虑到 CPU 效率的原因，如果死锁进程达到一定数量，就没有多少进程可以运行，所以 CPU 会经常空闲。

从死锁中恢复

上面我们探讨了如何检测进程死锁，我们最终的目的肯定是想让程序能够正常的运行下去，所以针对检测出来的死锁，我们要对其进行恢复，下面我们会探讨几种死锁的恢复方式

通过抢占进行恢复

在某些情况下，可能会临时将某个资源从它的持有者转移到另一个进程。比如在不通知原进程的情况下，将某个资源从进程中强制取走给其他进程使用，使用完后又送回。这种恢复方式一般比较困难而且有些简单粗暴，并不可取。

通过回滚进行恢复

如果系统设计者和机器操作员知道有可能发生死锁，那么就可以定期检查流程。进程的检测点意味着进程的状态可以被写入到文件以便后面进行恢复。检测点不仅包含**存储映像(memory image)**，还包含**资源状态(resource state)**。一种更有效的解决方式是不要覆盖原有的检测点，而是每出现一个检测点都要把它写入到文件中，这样当进程执行时，就会有一系列的检查点文件被累积起来。

为了进行恢复，要从上一个较早的检查点上开始，这样所需要资源的进程会回滚到上一个时间点，在这个时间点上，死锁进程还没有获取所需要的资源，可以在此时对其进行资源分配。

杀死进程恢复

最简单有效的解决方案是直接杀死一个死锁进程。但是杀死一个进程可能照样行不通，这时候就需要杀死别的资源进行恢复。

另外一种方式是选择一个环外的进程作为牺牲品来释放进程资源。