破坏产生死锁的四个必要条件

死锁预防

0:00 / 14:06

解决死锁的方法

- 不考虑此问题（鸵鸟算法）
- 不让死锁发生
  - 死锁预防

    静态策略：设计合适的资源分配算法，不让死锁发生
  - 死锁避免

    动态策略：以不让死锁发生为目标，跟踪并评估资源分配过程，根据评估结果决策是否分配
- 让死锁发生
  - 死锁检测与解除

死锁预防(DEADLOCK PREVENTION)

定义：

- 在设计系统时，通过确定资源分配算法，排除发生死锁的可能性

- 具体的做法是：防止产生死锁的四个必要条件中任何一个条件发生

破坏产生死锁的四个必要条件之一

我们的设计思路就是破坏产生 死锁的四个必要条件之一。

死锁预防(1/4)

破坏“互斥使用/资源独占”条件

- 资源转换技术：把独占资源变为共享资源

- SPOOLing技术的引入

  解决不允许任何进程直接占有打印机的问题

  设计一个“守护进程/线程”负责管理打印机，进程需要打印时，将请求发给该daemon，由它完成打印任务

"资源独占/互斥使用"的条件 第二个条件呢，是"占有且等待"条件

死锁预防(2/4)

破坏“占有且等待”条件

- 实现方案1：要求每个进程在运行前必须一次性申请它所要求的所有资源，且仅当该进程所要资源均可满足时才给予一次性分配
  - 问题：资源利用率低；“饥饿”现象

- 实现方案2：在允许进程动态申请资源前提下规定，一个进程在申请新的资源不能立即得到满足而变为等待状态之前，必须释放已占有的全部资源，若需要再重新申请

另外呢，它申请新的资源又得不到了，那么它也可能会导致"饥饿"现象

死锁预防(3/4)

- 破坏“不可抢占”条件

- 实现方案：

  当一个进程申请的资源被其他进程占用时，可以通过操作系统抢占这一资源(两个进程优先级不同)

- 局限性：适用于状态易于保存和恢复的资源

  CPU、内存

磁盘上去，然后把其他的内容呢读入到内存的页框里头 那么这都属于一种资源的抢占

死锁预防(4/4)

破坏“循环等待”条件

- 通过定义资源类型的线性顺序实现

- 实施方案：资源有序分配法

把系统中所有资源编号，进程在申请资源时必须严格按资源编号的递增次序进行，否则操作系统不予分配

- 实现时要考虑什么问题呢？
- 例子：解决哲学家就餐问题

哲学家就餐问题的时候呢，可以采用资源的有序分配法

为什么资源有序分配法不会产生死锁？

有资源1，2，3，…，10

P_1：	P_2：	P_3：	…… P_n：
申请1	申请1		
申请3	申请2		
申请9	申请5		
…	…		

描述了一下这个证明的一个思路

0:00 下面我们来介绍，解决死锁问题的各种方案 首先我们介绍第一种类型，叫做死锁预防 我们首先先总体介绍一下 解决死锁问题的各种方法。 不同的设计者 对于死锁问题的看法是不一样的 有的设计者很乐观，有的设计者很悲观 因此，他用于解决死锁问题的方法，是不一样的 我们总体上，有四种 方法，来解决死锁问题。 第一种方法呢 叫做鸵鸟算法，其实它的思路是 不考虑此问题，不理睬死锁问题 第二种方法呢，是不让死锁发生 那么也就是说，死锁这个问题很严重 我们一定要防范，不让死锁发生 不让死锁发生呢，这一类方法当中呢，实际上又分成了两种方法 一种呢，是叫做死锁预防 死锁预防是不让死锁发生的一个静态策略 它的主要思想是，通过设计合适的资源分配算法， 由资源分配算法来保证 不会出现死锁。 不让死锁发生 的第二类方法呢，叫做死锁避免 死锁避免是指的一个解决死锁问题的动态策略 它指的是以不让死锁发生为目标 不断地动态地来跟踪 评估资源分配过程。 根据这个评估的结果来决定 分配是否进行。 这是 两个方法，但是呢都属于不让死锁发生的这样一个角度 第三类方法呢，我们是让死锁发生 但是呢，我们不是置之不理，而是 通过死锁的检测 判断是不是死锁真的发生，然后再采用 一些方法，来解除死锁问题 归纳起来呢，总体解决死锁的方法呢，可以算为四类 第一类呢我们称之为鸵鸟算法，就是不理睬这个死锁问题，非常的乐观 第二类呢，叫做死锁预防 它是一个静态的策略。 第三类叫做死锁 避免，它是一个动态的策略 第四类呢是叫做死锁的检测与解除。 我们后面呢，就要介绍死锁预防，死锁避免。 死锁检测与解除。 首先我们来介绍 死锁预防。 所谓死锁预防呢， 指的是在设计系统的过程中 就确定一个合适的资源分配算法 通过这样一个资源分配算法，我们排除 产生死锁的可能性。 怎么样来 设计相应的资源分配算法呢？通常呢有这样一个典型的解决思路，也就是 防止产生死锁的四个必要条件当中的任何一个条件发生 那么，就可以进行死锁预防了。 所以下面呢，我们就从不让 这四个必要条件当中的某一个条件发生来提出相应的解决方案 我们的设计思路就是破坏产生 死锁的四个必要条件之一。 我们先来看一下 第一个条件，第一个条件呢是互斥使用，或者叫做资源独占 那么我们怎么样来破坏这个条件呢 资源本身它的特性是独占的 是排他性使用的，所以我们必须采用一种资源转换 技术，把这个独占的资源变成一个可共享的资源 以打印机为例 由于我们引入了 SPOOLing 技术，我们就解决了 不允许任何进程直接占用打印机的这样一个问题 那么在这里呢，实际上是我们，方法其实我们前面已经介绍过了 它主要

是设计一个"守护进程"或者"守护线程"， 由这个进程或线程来负责管理打印机 一个进程需要打印的时候呢，是把请求 发送给这个守护进程或者线程--daemon 由它来完成打印的任务 这样的话呢，就把一个独占的资源，把它转换成一个共享的资源，就破坏了这样一个 "资源独占/互斥使用"的条件 第二个条件呢，是"占有且等待"条件 "占有且等待"呢，指的是一个进程申请到了一部分资源 在申请其他资源的时候由于得不到满足而进入等待状态 那么，破坏这个条件呢，我们可以通过这样一个实现方案 也就是说，要求每个进程，在运行前 一次性申请它所需要的所有资源 而系统呢，会 当这个进程所需要的资源都可满足的时候，一次性给 这个进程呢进行分配。 因此呢这个进程 拿到了它所需要的所有资源，在运行过程中不会再申请新的资源，也就不会出现等待状态 当然我们知道，这个做法，实际上 它的资源利用率非常的低，因为 原来我们采用的是动态申请资源的策略 这样它就又退化成了一个静态资源分配策略了，所以资源利用率就降低了 同时它可能还会导致一个"饥饿"现象的发生，因为一个进程它要的各种资源 不能同时满足的情况下，这个进程的 执行就会被推迟，延迟，所以呢就产生了饥饿 那么第二种实现方案呢，就是说，在允许 进程动态申请资源的前提下 那么当一个进程在申请新的资源得不到满足的时候 那么它要进入等待状态，在它进入等待状态之前，它把它以前 得到的这个资源，占有的资源呢，全部释放掉 那么如果它从等待变成就绪呢 那么它再重新申请这些资源 也就是，主动把它放弃，在等待之前 当然这样的话呢，也会导致一些问题，比如说它再申请新的资源 那么这些资源的状态，不是它刚才使用的状态了，怎么办 另外呢，它申请新的资源又得不到了，那么它也可能会导致"饥饿"现象 下面呢我们来介绍破坏第三个条件，"不可抢占" 那么实现的方案呢是这样的。 当一个进程 在申请资源的时候，那么这个资源被其他的进程所占有时 可以通过操作系统来抢占这个资源，那么 这种实现方案呢，主要要考虑，两个 进程他们的优先级是不一样的，我们要区别对待 那么这种方法呢，是具有局限性 它主要是适合于，状态易于保存和恢复的这样的资源 前面我们提到的内存、 CPU，都属于这种资源 CPU的话我们说进程可以通过抢占式的调度算法 然后从其他进程手里头，抢占CPU 内存也可以通过页面置换算法，把一个进程的一些页面呢置换到 磁盘上去，然后把其他的内容呢读入到内存的页框里头 那么这都属于一种资源的抢占 下面我们来介绍破坏"循环等待"条件所采用的方法 主要的思想呢，是通过定义资源类型的线性顺序 来实现破坏循环等待

条件 具体的方案呢，有一个称之为 资源的有序分配法，所谓资源的有序分配法 指的是首先把系统当中的所有资源进行编号 并且要求进程在申请资源的时候 要严格按照编号的递增的次序来进行 如果你违反了这样一个次序 操作系统就不予分配，在实现资源的有序分配法时 我们要考虑如何对资源进行编号 哪些资源的编号在前面？ 哪些资源的编号在后面？通常呢 我们 可以按照资源使用的频繁性 常使用的资源呢，我们通常把它放到前面 不常使用的资源呢，我们把它在后面，那这样的话呢，可以 很好地来实现资源的有序分配法 当然了，我们后面我们会 介绍哲学家就餐问题，那么它在解决 哲学家就餐问题的时候呢，可以采用资源的有序分配法 为什么资源有序分配法 采用了之后，不会产生死锁问题呢？ 那么这里可以证明这一点，那么我们在这里呢给出一个 简单的序数，假如说 有资源编号是从 1 到 10 现在有若干个进程需要使用这些资源 我们简单地举了一个小例子 P1 这个进程呢它需要三个资源，1 号、 3 号和 9 号 P2 呢这个进程呢，它需要 1 号、 2 号、 5 号 三个资源，当然了后面的就省略没有展示出来 那么我们看一下，在任何一个时刻 我们在这个进程的集合当中，总能够找到一个进程 这个进程它得到的 资源编号是最大的那一个，也就是说它得到了一些资源 那么在这些资源当中呢，因为它是按顺序申请的 所以它拿到了目前为止，得到资源的编号最大的那个资源是在这个进程数里头 所以我们在这个进程集合里头，可以找到这样的进程 它拿到了当前所分配资源当中，资源编号最大的一个 那我们从这个进程出发 后序，因为后序的资源编号 都会比它所占有的资源编号大 而后序的这些资源都没有分配出去，所以让这个进程 继续申请，它后面再申请的任何资源其实都是能满足的 按照我们前面的假设，如果资源都满足了，那么它就可以把 资源使用完还回给系统，这样的话呢进程，这个进程就执行完成 那么这个集合呢，从 N 个进程，就变成了 N-1 个进程 然后我们再从 N-1 个进程当中，继续寻找 当前占有这个资源编号最大的那个进程 然后把后序的资源分给这个进程，那么这个 N-1 这个集合的进程数呢 就变成了 N-2 那么不断地消解，那我们最后所有的进程都能够完成 那么这样系统就不会出现死锁，啊，这就是简单 描述了一下这个证明的一个思路