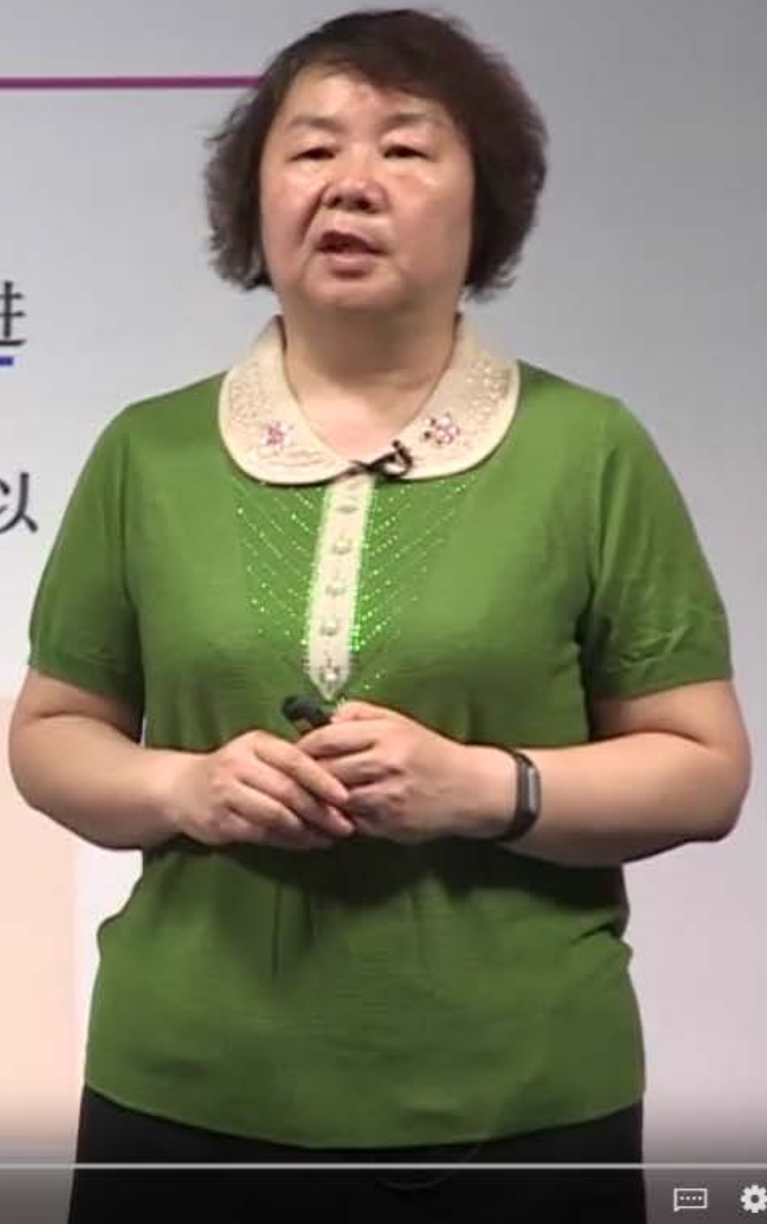
死锁的检测与解除

死锁检测：

- 允许死锁发生，但是操作系统会不断监视系统进展情况，判断死锁是否真的发生
- 一旦死锁发生则采取专门的措施，解除死锁并以最小的代价恢复操作系统运行

检测时机：

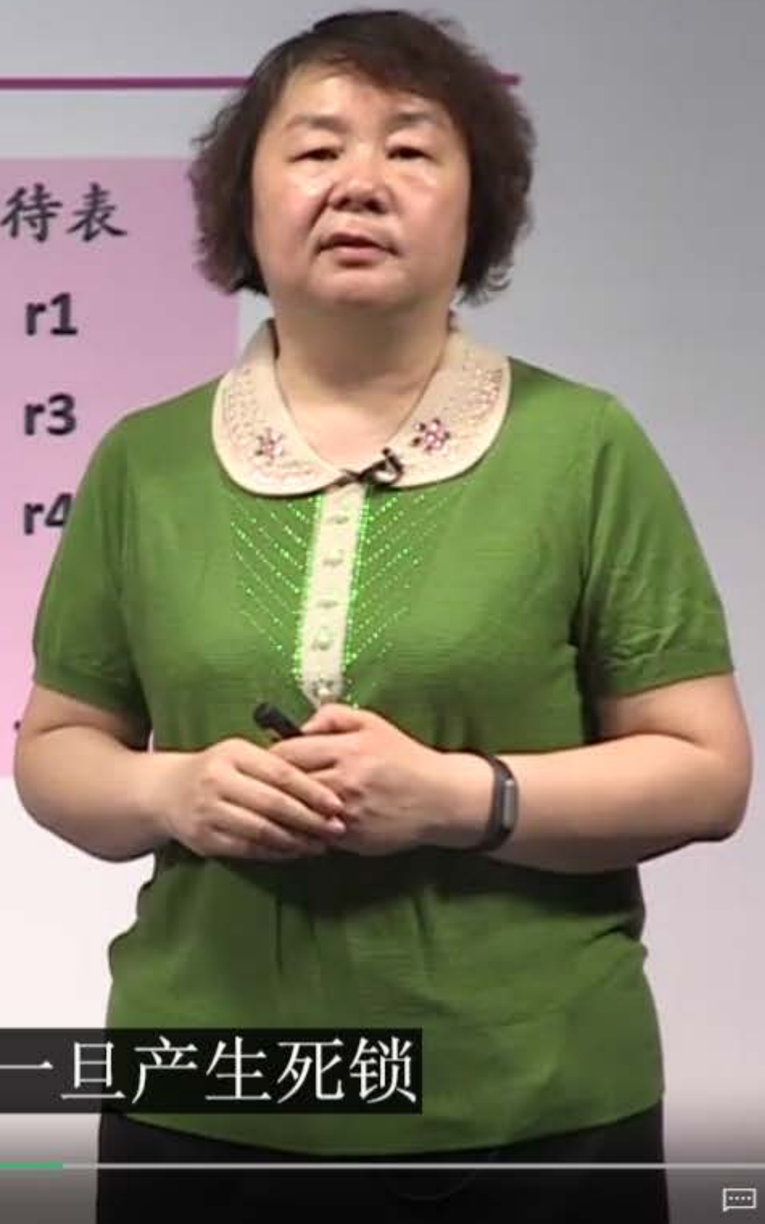
- 当进程由于资源请求不满足而等待时检测死锁
  缺点：系统开销大
- 定时检测
- 系统资源利用率下降时检测死锁



一个简单的死锁检测算法

* 每个进程、每个资源指定唯一编号
* 设置一张资源分配表
  记录各进程与其占用资源之间的关系
* 设置一张进程等待表
  记录各进程与要申请资源之间的关系

资源分配表		进程等待表	
r1	p2	p1	r1
r2	p5	p2	r3
r3	p4	p4	r4
r4	p1		
…	…	…	…



p1 → r1 → p2 → r3 → p4 → r4 → p1

在这样一个状态下，死锁产生了 那么一旦产生死锁

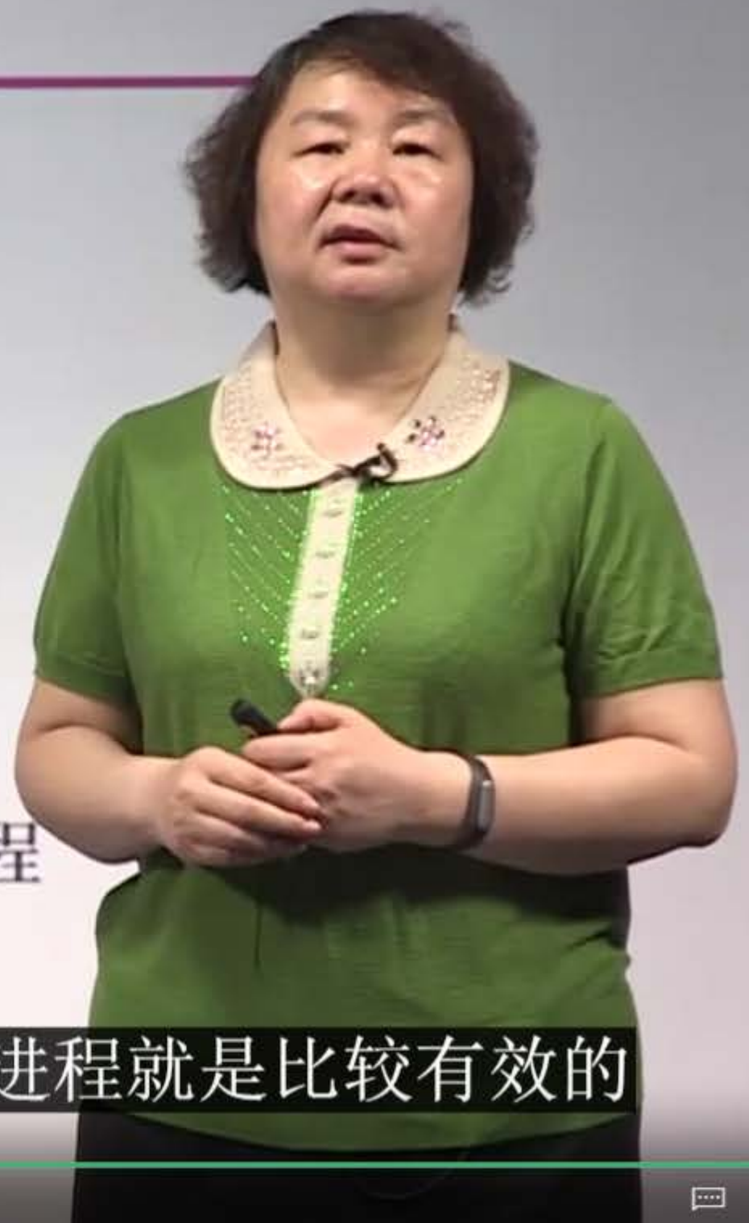
死锁的解除

重要的是以最小的代价恢复系统的运行

方法如下：

- 撤消所有死锁进程
- 进程回退（Roll back）再启动
- 按照某种原则逐一撤消死锁进程，直到...
- 按照某种原则逐一抢占资源（资源被抢占的进程必须回退到之前的对应状态），直到...

选择原则



解放出来，那么这样的话呢，选择这个牺牲的进程就是比较有效的

0:00 下面我们来介绍解决死锁问题的第三类方法 死锁检测与解除 所谓死锁检测呢，是指允许死锁发生 但是操作系统呢 会不断地来监视系统的进展情况 通过监视，判断 死锁是否真的发生了 那么一旦发现有死锁发生 就会采用专门的措施 以最小的代价来解除死锁，恢复操作系统的运行 那么系统什么时候检测 判断是否有死锁发生呢？ 我们给出来三个典型的检测时机，第一个 检测时机呢，是当进程在申请资源的时候 这个资源系统不能满足它，那么这个进程进入等待状态，这个时候来 检测由于这个进程进入等待状态，是不是会导致死锁的发生 因为我们知道，死锁实际上是一组等待的，互相等待的进程 但是每个进程进入等待的时候，就去检测死锁是不是发生 这里头会带来巨大的系统开销，而且 像读盘啊，你去等待读盘的结果，很正常的一个等待 要去判断死锁是不是真的发生，那么未免有些小题大做 第二个时机呢 就是定时检测 我们可以设定一个检测的周期，比如说是一天 或者是一周，或者是划定多长时间 我们来做一次这样的检测，那么第三个时机 就是当系统的资源利用率下降的时候，我们才来进行相应的检测 这样的话呢，我们就可以设定一个值，我们这个阈值 这是系统资源利用率的值，当系统资源 利用率低于这个值的时候，我们就启动检测的算法 来检测由于资源利用率太低了，是不是出现什么问题了，是不是出现死锁了 下面我们来介绍一个比较简单的死锁检测算法 这个算法呢，它是给每个进程和每个资源呢都 给定一个唯一的编号，然后呢设置了两个数据结构 第一个数据结构呢 叫做资源分配表 它记录了进程与它占有资源之间的一个关系 第二个数据结构呢，叫做进程等待表 它记录了各个进程与它要等待的资源之间的关系 我们来看一个例子 这个资源分配表当中 给出了这个资源分配给了哪个进程？ 而进程等待表当中，记录了 某个进程它在等哪个资源，那么 死锁检测算法呢，实际上就是来检查这两张表 通过检查来判断一下死锁是否真的发生 我们来看一个例子，我们先从进程等待表出发 我们呢 从进程等待表当中，找到了进程 p1 我们来看 p1 在等哪个资源，是等 r1 这个资源 那么 r1 这个资源一定是分配给了某个进程，我们就去查 资源分配表，看看 r1 这个资源究竟分配给了哪个进程呢？ 我们得到它是分配给了 p2 ，那么 p2 这个进程 它是不是处于等待状态呢？如果 p2 进程它没有处于等待状态，那么 这个结束，这个判断结束，因为没有环 那么 p2 如果出现在了进程等待表里头，我们就要继续判断 p2 等待的是 r3， 而 r3 呢 通过查资源分配表发现，分配给了 p4 这个进程 而 p4 又在进程等待表中出现，它呢 是等 r4 这个资源 而 r4 这个资源呢，我们看到它是分配给了 p1 因此，我们就通过这样一个分析，判断出 p1 等自己，那么肯定 无限期的等待

下去了， 我们也可以说这里出现了一个进程等待的环路 那么根据死锁定理 有环路，就有死锁发生，所以我们得出 在这样一个状态下，死锁产生了 那么一旦产生死锁 最重要的就是以最小的代价，来尽快恢复系统的运行 具体的死锁解除的方法呢 有几个，第一个 就是撤销所有陷入死锁的进程 这个代价是比较大的，第二个呢 是采用的叫进程回退再启动的方法 所谓进程回退呢，就是在进程执行过程中 系统会为每个进程记录一些中间结点 这些中间结点的话， 当 进程出现了死锁之后，那么让这些死锁的进程都往回退一步 退到上一个中间结点上 然后再重现从，大家所有的进程都从 它们上一个点重新继续地往前执行 因为我们知道，死锁发生实际上是偶然现象 它可能是在一个进程运行、 调度、 并发执行的环境当中 由于非常恰巧的一些资源分配的顺序会产生 死锁，那么如果每个进程都回退了一步 再接着去执行，那么调度算法，并发环境可能 变化了之后，那么就可能不会出现死锁问题了 所以进程的回退，就指的是利用这样一个特性 但是呢进程的回退呢代价也非常大，因为你要记录进程的很多的中间点 那么都是需要花费开销的，后面两种方法呢 其中一种是按照某种原则，逐一地撤销进程 我不是把所有进程全部撤销，而是一个一个撤销 然后撤销一个，再判断一下，没有死锁了，那么撤销就结束了，如果还有死锁，再撤销一个 或者是剥夺资源，也是逐一地来进行 这就是两种，就是逐步迭代的方法来解除死锁 当然了选择哪一个进程还是很关键的，那么最好 选择的进程它可能占用的资源是最关键的，一旦这个 资源被还回去，被抢占了，那么就可以把其他的进程都 解放出来，那么这样的话呢，选择这个牺牲的进程就是比较有效的