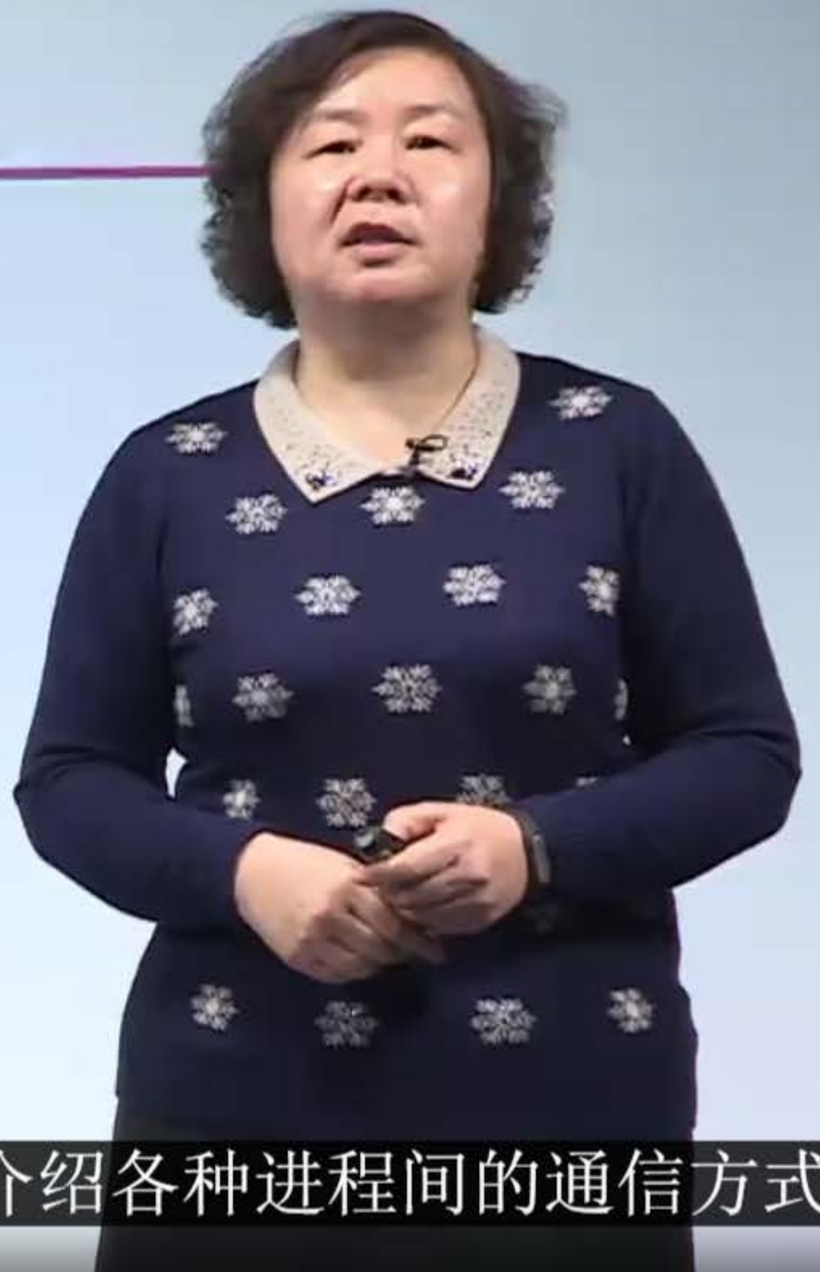
同步互斥机制与通信机制

- 管程monitor
- 进程间通信

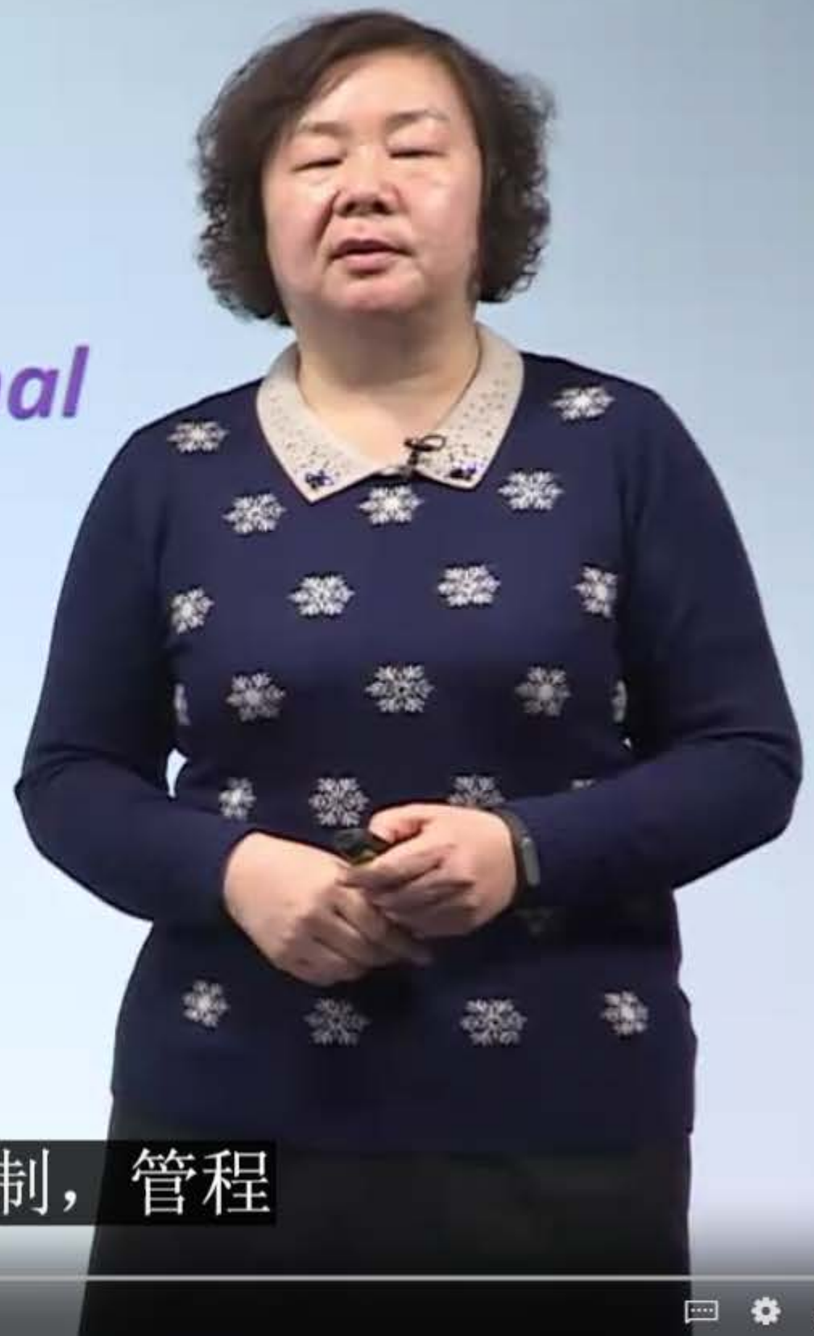
  Inter-Process Communication
- 典型操作系统的IPC机制



关于同步互斥机制呢，我们着重介绍管程 然后我们介绍各种进程间的通信方式

语言机制、条件变量/wait/signal

管程 MONITOR



下面我们首先介绍 一种新的同步机制，管程

为什么会出现管程？

问题

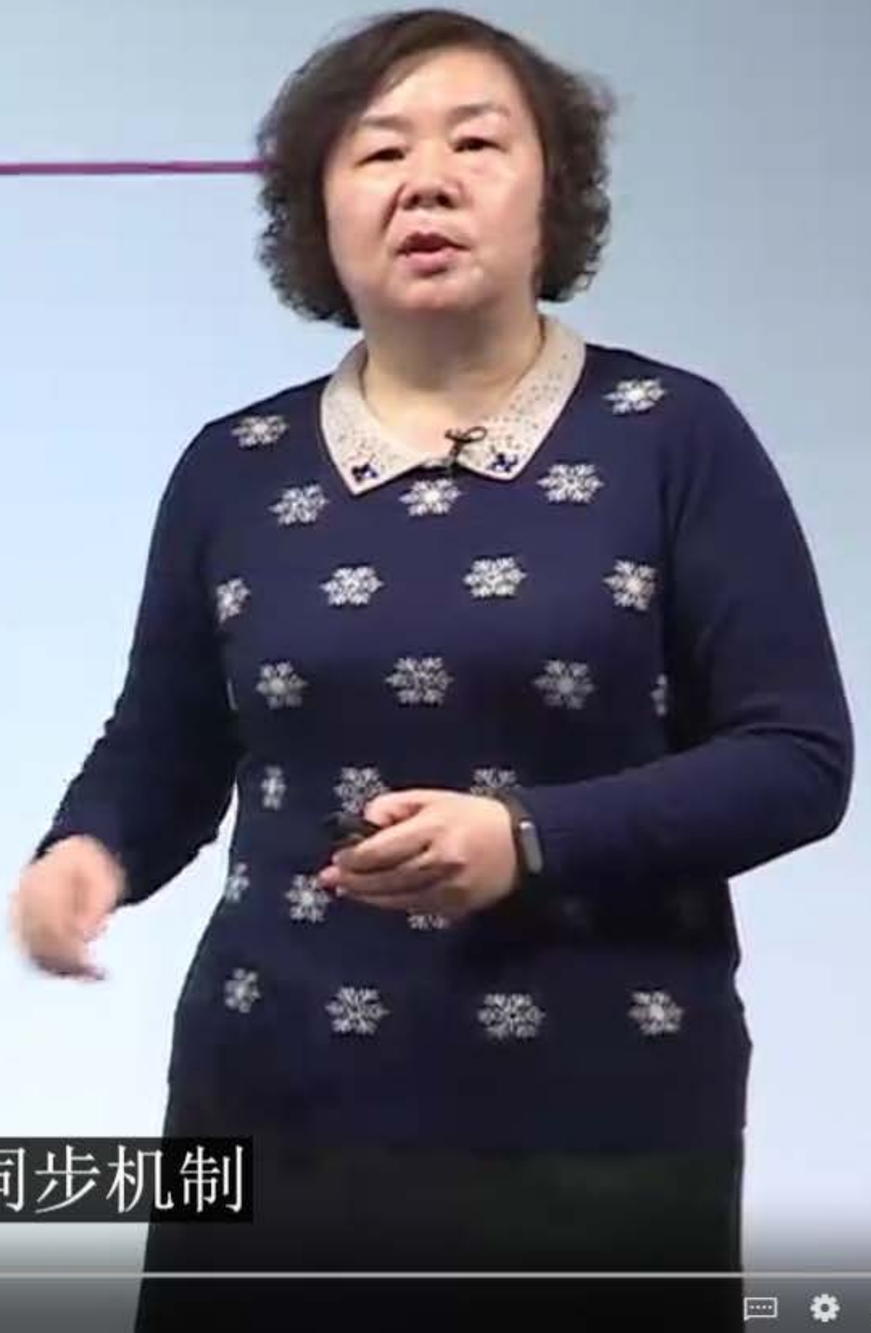
- 信号量机制的不足：程序编写困难、易出错

解决

- Brinch Hansen(1973)
- Hoare（1974）

方案

- 在程序设计语言中引入管程成分
- 一种高级同步机制



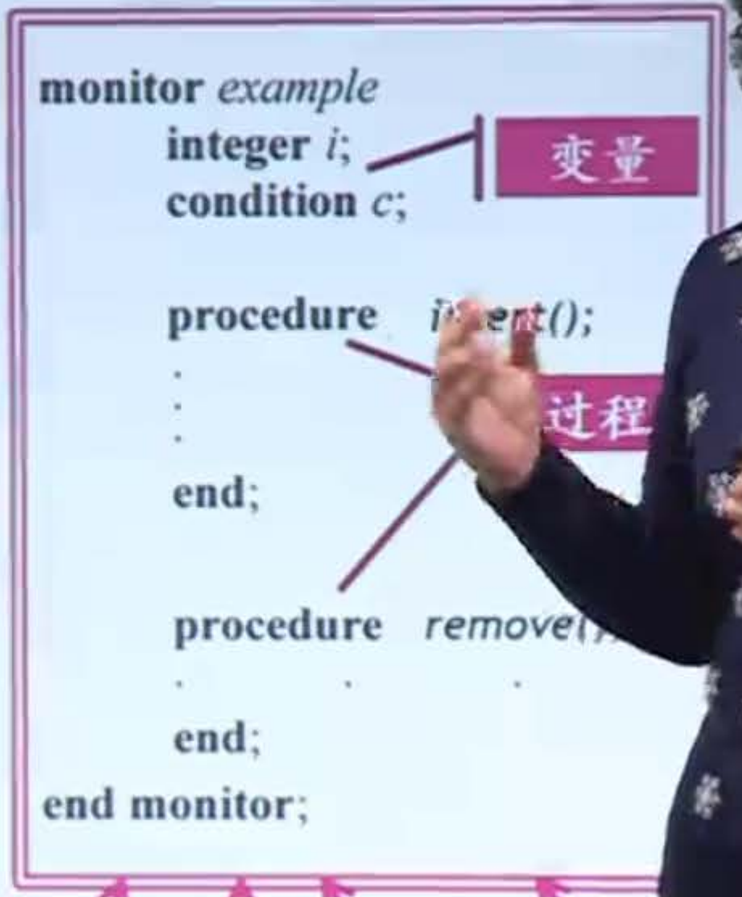
那么也是一种我们称之为高级同步机制

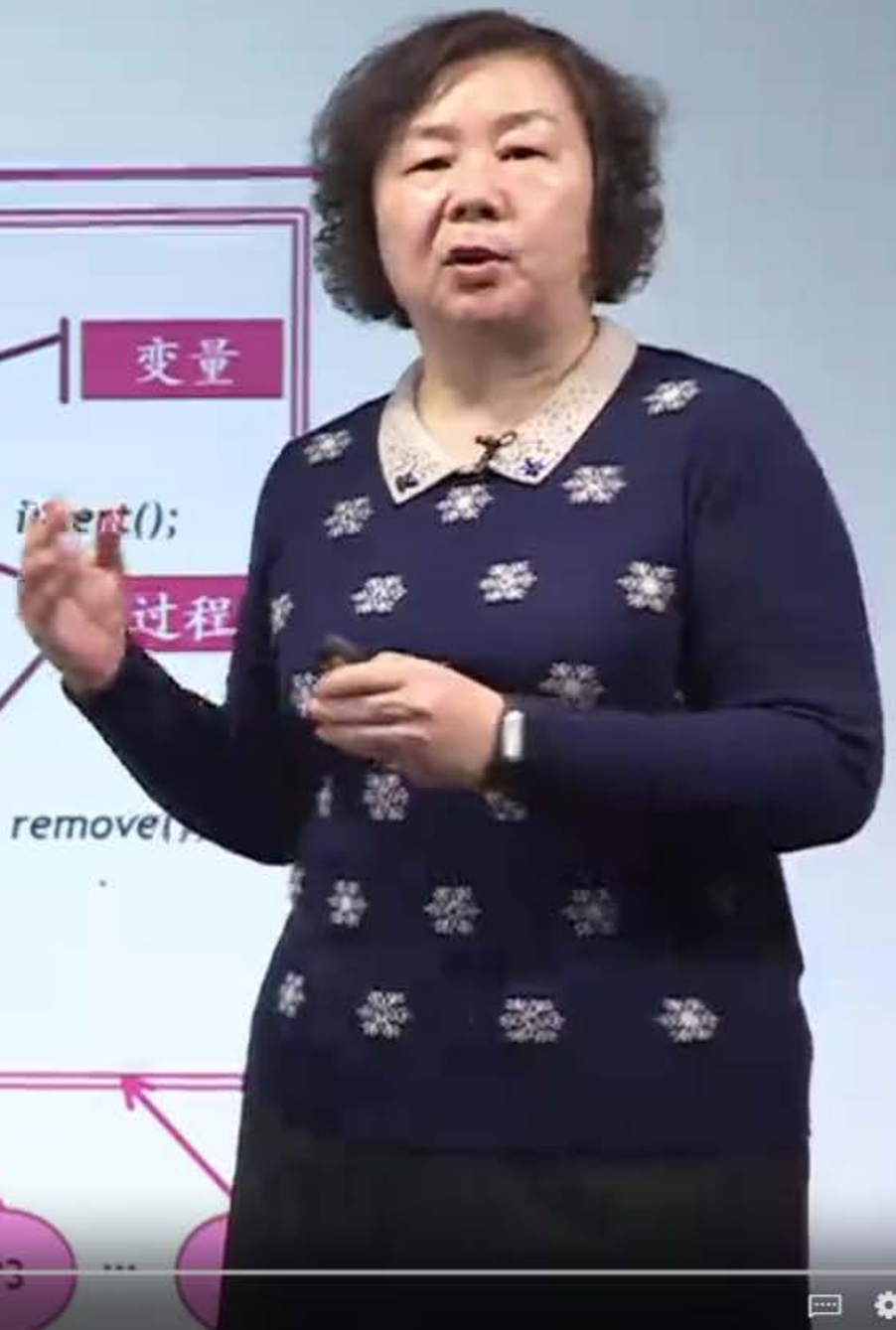
管程的定义

- 是一个特殊的模块
- 有一个名字
- 由关于共享资源的数据结构及在其上操作的一组过程组成

□ 进程与管程

进程只能通过调用管程中的过程来间接地访问管程中的数据结构





管程要保证什么？

- 作为一种同步机制，管程要解决两个问题？
- *互斥*

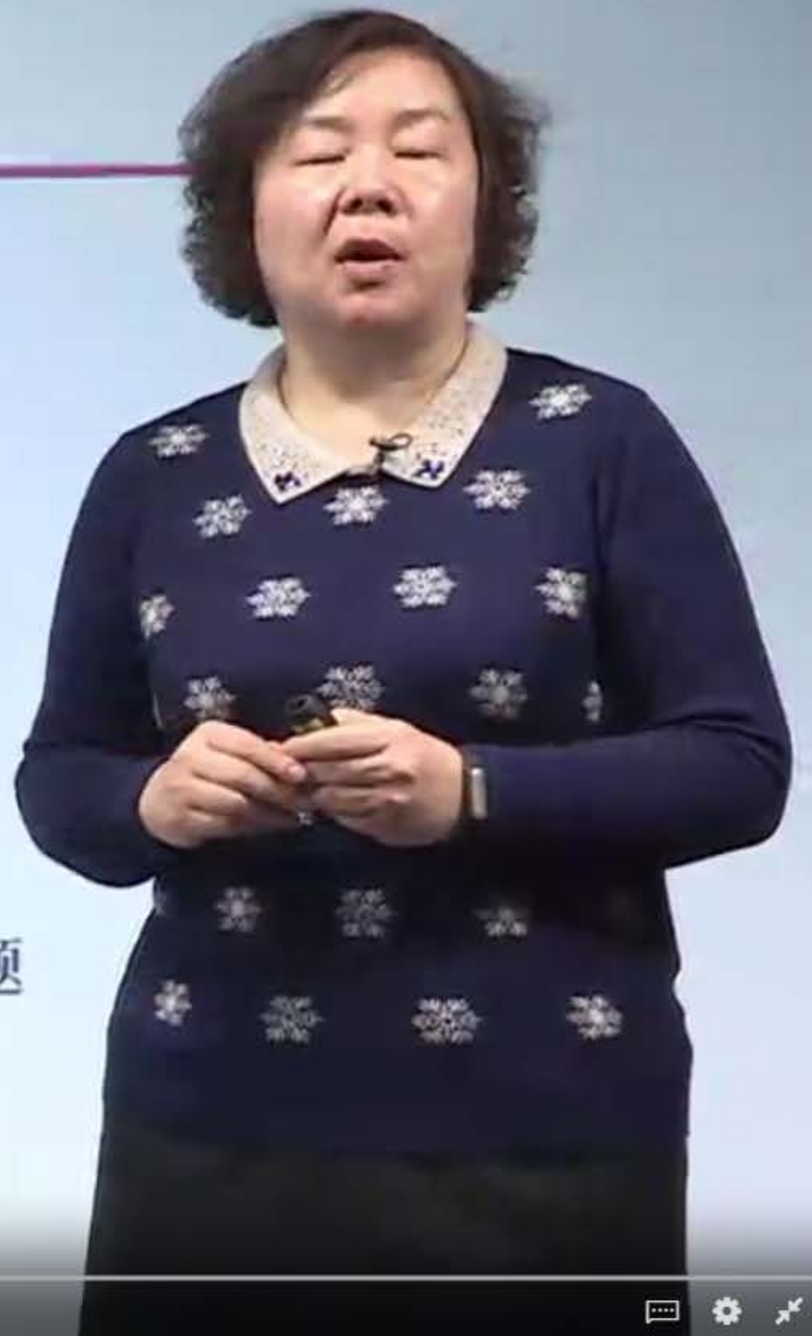
  管程是互斥进入的

  ——为了保证管程中数据结构的数据完整性

  管程的互斥性是由编译器负责保证的
- *同步*

  管程中设置条件变量及等待/唤醒操作以解决同步问题

  可以让一个进程或线程在条件变量上等待（此时，应先释放管程的使用权），也可以通过发送信号将等待在条件变量上的进程或线程唤醒



应用管程时遇到的问题

设问：是否会出现这样一种场景，有多个进程同时在管程中出现？

场景：

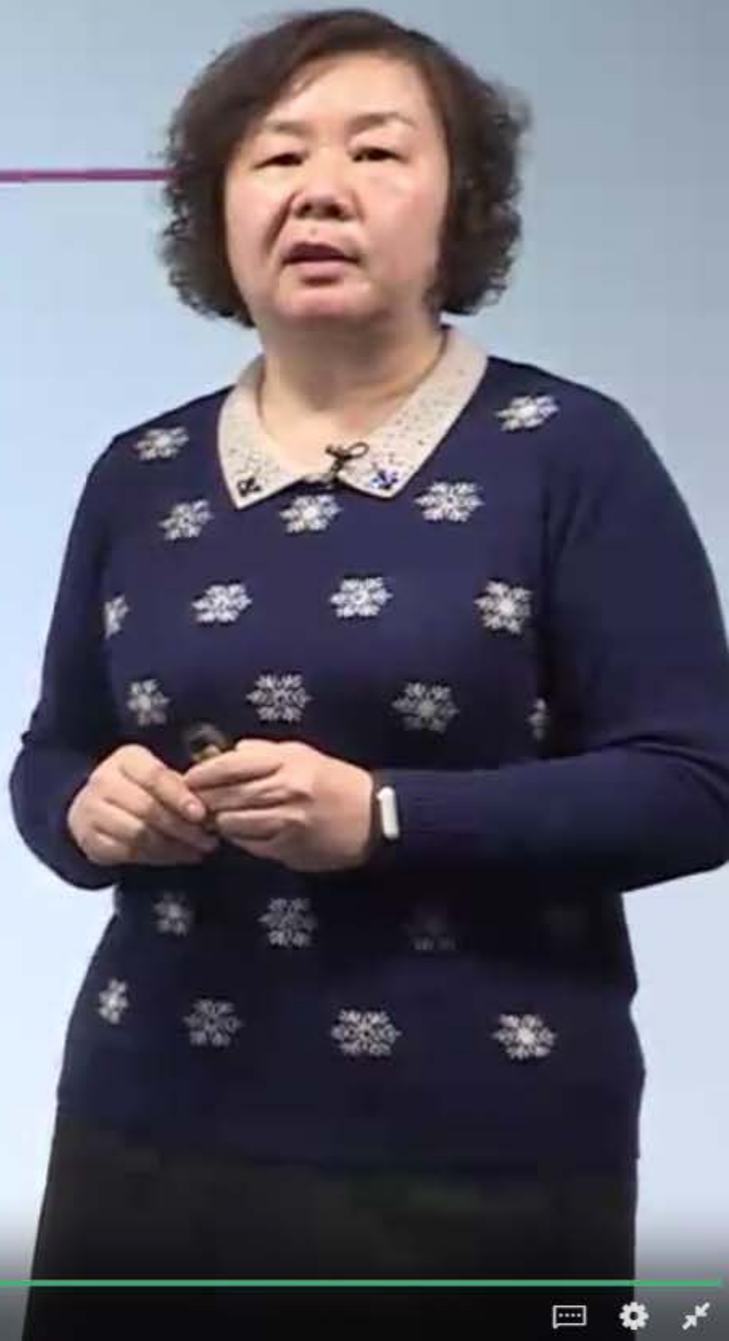
当一个进入管程的进程执行等待操作时，它应当释放管程的互斥权

当后面进入管程的进程执行唤醒操作时（例如P唤醒Q），管程中便存在两个同时处于活动状态的进程

如何解决？

三种处理方法：

- P等待Q执行 Hoare √
- Q等待P继续执行 MESA
- 规定唤醒操作为管程中最后一个可执行的操作 Hansen、并发pascal



0:00 大家好，今天我给大家带来的是操作系统原理的第六讲 同步互斥机制 2，还有一个是进程间通信的机制 这一讲里头呢 关于同步互斥机制呢，我们着重介绍管程 然后我们介绍各种进程间的通信方式 下面我们首先介绍 一种新的同步机制，管程 在引入了信号量及 PV 操作之后 为什么又出现了一种新的同步机制管程呢？ 主要的原因是，信号量 这种机制具有一些缺点，比如说 用信号量及 PV 操作解决问题的时候 程序编写需要很高的技巧。 另外 如果没有合理地安排 PV 操作的位置，就会导致一些出错的结果 比如说出现死锁等问题。 好，那么 有了这样一些问题之后，在上个世纪的 1973 年、 1974 年 分别有两位科学家 提出了一种新的同步机制 几乎是在同一时间，那么这个新的同步机制呢 实际上呢，就是管程。 管程 我们首先看一下，它实际上是在程序设计语言当中引入的一个成分 那么也是一种我们称之为高级同步机制 我们来看一下管程的定义 管程呢，其实是一种特殊的模块 那么每个管程都有一个名字 那么管程里头主要是管理了 共享资源所对应的数据结构 所以管程里头是管理这个共享资源 同时，提供了在这个共享资源之上需要的各种各样的操作，也就是由一组 操作的过程来组成。 所以管程是这样一个成分 那我们看一个示意，在这个管程的示意当中呢 我们看到管程有一个名字，比如说这个 管程，我们把它可以看成是管理生产者、 消费者的一个 缓冲区的这样一个管程。 那么这里头呢有一些 变量，当然每个变量有它的自己的作用 然后还有一些过程，比如说生产者、 消费者要 往缓冲区里头放数据或者是取数据，那我们可以说设计成 插入和取出这样一些过程 那这就是一个管程的一个大致的示意，所以它是由 共享的数据结构以及在这些数据结构之上 一组操作组成。 那么进程与管程的关系是什么呢？ 作为进程，它只能通过调用管程给提供的各种过程 来间接地来使用管程当中的数据结构 那么我们以这个管程为例 有若干个进程，这些是生产者或者消费者 如果他们要往 缓冲区里送数据或者去从缓冲区里读数据的话 那么他就要调用管程当中的 insert 或者是 remove 这样一些过程 这就是这些生产者、 消费者进程它通过调用管程的 提供的过程来对管程里的数据结构进行操作这样一句话的含义 作为管程，它是一种同步机制 它需要解决两个问题，哪两个问题呢？ 第一个问题是互斥，因为管程 管理了这样一些共享的资源，因此 它要求凡是使用这些资源的进程 只能有一个进程来在对这个数据结构进行相应的操作，也就换句话说 管程是互斥进入的，有一个 进程在调用管程里的过程，就不允许其他进程调用 那么这主要是为了保证管程中的数据结构的完整性 管程如何来解决互斥问题呢？ 通常，管程的互斥 是由编译器来 负责的，是由编译器来保证的 这是一个非常

需要强调的一点，因为管程是一个语言机制 那么编译器可以做到 让管程是互斥进入的 第二个要解决的问题，就是同步问题 那么在管程当中如何解决同步问题呢？ 主要是设置了条件变量 以及在条件变量上实施的 wait 和 signal 操作，也就是等待和唤醒操作 所以条件变量再加上条件变量上实施的这两个操作 就可以解决管程的这个同步问题 那么这个两个操作 一个是等待，一个是唤醒，它的作用是，可以使一个进程或者线程，当条件不满足的时候呢 在条件变量上呢等待。 或者 通过唤醒操作 使得在条件变量上等待的进程或者线程呢 把它们唤醒。 这就是同步 要解决的，一个是等待，如果条件不成立 需要的信息没到，那么需要等待 如果需要的信息到来，那么就需要把这个 等待的进程唤醒。 所以需要这样两个操作，但这里头呢 我们先强调一下，当一个进入管程的进程 去执行了等待的操作之后，由于管程是互斥进入的 所以它进入等待了，它要把管程的互斥权或者管程的使用权 释放，允许其他的进程进入管程 好，这是关于管程要保证 互斥和同步。 那么我们在使用管程的时候呢 会遇到什么问题？我们提一个问题 请问是不是会出现这样一个场景？ 有多个进程同时在管程中出现 因为我们刚才已经说了，管程是互斥进入的 那么我们想问一下，会不会出现这样一个场景？ 那我们来举一个例子 假设有一个 进程先进入了管程，但是由于进了管程之后 它要做的事情由于条件不成熟，还不能做，所以它 调用了等待操作，执行了等待操作 那么我们刚才说了，在它执行等待操作的时候呢，应该把管程的 互斥权释放，允许其它的进程进入管程做其它的事情 假设后面有一个进程进入了管程 在在管程内的时候，这个进程呢 做了一个唤醒操作，把前面那个等待的进程唤醒了 我们譬如说，我们就说 P 进程唤醒了 Q 进程 P 进程是后进来的，前面的是 Q 进程，所以 P 进程唤醒了 Q 进程 那么这种情况下，在管程当中 就同时出现了两个活跃的 进程，所以我们说这个问题，这种场景是可能会出现的 但发现了这个问题之后，我们如何来解决呢？ 于是研究人员就在 想出不同的解决方案，一共提出了三种处理方法 第一种处理方法呢，我们先看，从底下看一下 那么这种处理方法呢，是规定唤醒操作 是管程当中的最后一个 执行的操作，也就是做完这个操作之后 这个进程就出管程了，所以呢 这样就可能使得在 原来在管程里头被唤醒的那个进程，其实它要执行也没有关系，因为 前面一个进程已经出管程了。 所以这是 Hansen 提出的一种解决方案，他是在 并发 pascal 这个语言当中做了一个实现 那么第二种，我们从上面看 那么 P 进程唤醒了 Q ，那么如果我们允许 P 等待，而 Q 执

行，那么 也就是说后面进来的这个进程呢 等待，让前面一个进程执行，那么这种方法呢 实现的管程就是 Hoare 管程，因为是 Hoare 提出这种方案 还有一种方案呢就是 P 进程唤醒了 Q 进程 那么 Q 进程等待，继续等待 然后 P 进程执行，当然了，我们说等待 比如说 Q 进程等待，它刚才等待的是在条件变量上 那现在等待呢，就应该在另外一个地方，另外一个队列等待 不在一个地方等待。 那么这个方法呢 我们呢叫 MESA，MESA 管程，也是 一个比较新的一个方案 好，那么这就是在应用管程的时候 遇到的问题以及三种解决方案