一种经典的进程同步机制

信号量及P、V操作

下面我们介绍一种非常经典的进程的同步机制

信号量及PV操作

- 一个特殊变量
- 用于进程间传递信息的一个整数值
- 定义如下：

```
struc semaphore
{
        int count;
        queueType queue;
}
```

- 信号量说明：semaphore s;
- 对信号量可以实施的操作：初始化、P和V（P、V分别是荷兰语的test(proberen)和increment(verhogen)）

是一种卓有成效的进程同步机制

1965年，由荷兰学者Dijkstra提出

P、V操作定义

```
P(s)
{
   s.count --;
   if (s.count < 0)
   {
      该进程状态置为阻塞状态;
      将该进程插入相应的等待队列s.queue末尾;
      重新调度;
   }
}
```

down, semWait

```
V(s)
{
   s.count ++;
   if (s.count < = 0)
   {
      唤醒相应等待队列s.queue中等待的一个进程;
      改变其状态为就绪态，插入就绪队列;
   }
}
```

up, semSignal

大家也要看书的时候，也就 down 就是 P 操作，up

有关说明

- P、V操作为原语操作(primitive or atomic action)

- 在信号量上定义了三个操作

  初始化(非负数)、P操作、V操作

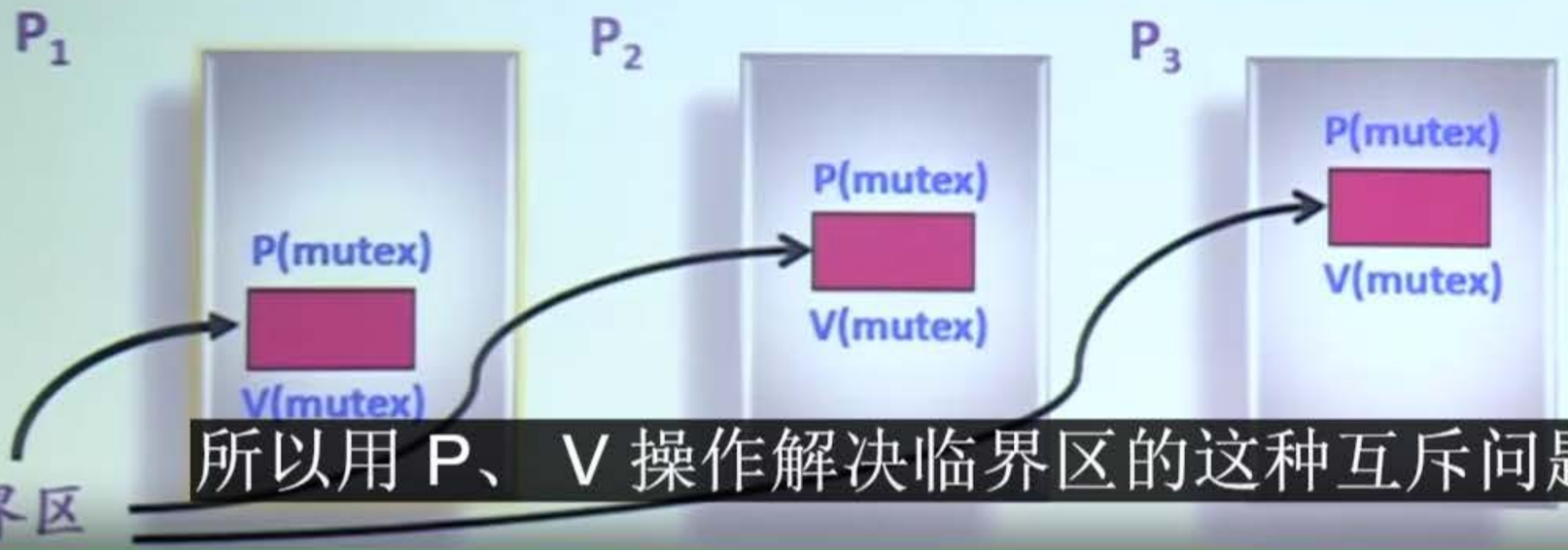
- 最初提出的是二元信号量（解决互斥）

  之后，推广到一般信号量（多值）或计数信号量（解决同步）

因为 根据它的值，我们可以用它来解决同步问题，也可以来解决互斥问题

用PV操作解决进程间互斥问题

- 分析并发进程的关键活动，划定临界区
- 设置信号量 mutex，初值为1
- 在临界区前实施 P(mutex)
- 在临界区之后实施 V(mutex)



所以用 P、V 操作解决临界区的这种互斥问题是非常简单的

0:00 下面我们介绍一种非常经典的进程的同步机制 之所以叫同步机制呢，是因为 我们通常把进程的互斥看成是一种特殊的同步 所以，我们就称之为同步机制，它既解决同步的问题，也能解决互斥的问题 那么这种典型的 进程的同步机制呢就称之为信号量 及 P、 V 操作。 那么这种机制 是 1965 年非常著名的荷兰的这个学者 Dijkstra 他提出来的。 那么 什么是信号量呢？信号量呢其实是一种特殊的变量 它实际上呢是用于进程之间传递信息的这么一个整数值 信号量呢我们给它这么一个定义：它由一个值和一个队列组成 那么也就是这个值是你要传递的一个整数值，传递信息的整数值 而队列呢是允许进程 挂到这个队列上的。 那么如果我们要声明一个信号量呢，我们可以这么来描述 信号量既然是一个变量，那么它在上面就可以实施相应的操作 在信号量上只能实施的操作有三个。 第一个 给这个值初始化，给信号量的值初始化 第二个是 P 操作，第三个是 V 操作 只能实施这三类操作 而 P 和 V 呢是荷兰语 test 和 increment 的 这个荷兰语当中的那个词的首字母 这就是 P 和 V，Dijkstra 刚刚开始提出来 信号量的时候呢，他可能把它叫做信号灯。 它和铁路的 这个信号红和绿，让车通过、不让车通过有关系，所以叫信号灯 那现在呢大部分情况下，我们还是叫信号量 那我们来看看在信号量上实施的两个操作 第一个呢是 P 操作，第二个是 V 操作。 P 操作呢主要做两件事情，第一件事情呢 要给信号量的值减 1，就是这个整数值减 1 然后去判断这个值是不是小于 0 了 如果小于 0，那么 这个进程的状态就变成了阻塞态，说谁调用了 P 操作，这个进程的状态变成阻塞态 并且把它送到相应的等待队列的末尾，也就是刚才我们说的这个信号量上有一个队列 就等待队列的末尾，那这样的话呢，这个进程实际上就让出了 CPU 接着就要重新调度，选另外一个进程上 CPU 了，所以这是 P 操作。 当然如果信号量的值减完 1 之后 减完 1 之后，还是不小于 0 的，那么这种情况下就 实施 P 操作的进程呢就继续实行 那么 V 操作呢实际上是信号量的值加 1 然后判断一下信号量的值是不是小于等于 0 如果小于等于 0 ，说明原来这个信号量上有进程在等，因此呢就唤醒 这个信号量队列上等待的这个进程 唤醒其中一个进程，然后把这个进程的状态改成就绪态 然后把它送到就绪队列，然后执行 V 操作的进程呢可以继续执行 这就是 V 操作。 那么我们这个介绍的实现方案呢 和我们教材上介绍的呢略有不同 那么只是说这两个操作在实现上的不同，在使用上 是相同的。 所以呢我们希望大家比较一下这俩有什么不同 给大家这么一个机会去比较一下。 在我们的书上呢 还有一个不一样的地方就是说在我们的书上，P 操作呢叫 down V 操作呢叫 up，就是加 1、 减 1 这样一个含义 在其他的一些教

材里头呢还有 semWait 和 semSignal 这样的这个操作的名称 那我们叫 P 和 V，因为 P 和 V 呢第一比较简单，大写比较简单 第二呢很多的教材都用 P 和 V 操作，那么大家呢取得一个共同的这么一个术语 大家也要看书的时候，也就 down 就是 P 操作，up 就是 V 操作 下面呢我们对信号量及 P、V 操作做一个简要说明 首先， P 操作和 V 操作是原语操作，也是原子操作 在执行过程中呢不允许被中断，所以 整个刚才我们所介绍的那一段代码呢，实际上是在执行过程中是分中断执行的 把中断关闭，然后呢执行，然后再打开中断 在信号量上呢，刚才我们已经介绍过了 实施三个操作，一个是初始化，通常是非负值 然后 P 操作和 V 操作。 在 Dijkstra 首次提出这个信号量这个概念的时候呢 他叫的二元信号量，也就是两个值 0 或者是 1，那么用二元信号量主要当初解决的是互斥问题 后来发现如果把信号量的值 从 0 和 1 可以往正数 推，或者是往负数，那么给它 扩大，那么实际上呢就是叫做一般信号量，或者叫做多值信号量 也叫计数信号量，有这样一些名称 用于解决什么问题呢？就可以用于解决同步问题 那么现在呢不是很强调二元信号量或者一元 一般信号量、多值信号量或者计数信号量，直接就叫信号量。 因为 根据它的值，我们可以用它来解决同步问题，也可以来解决互斥问题 下面呢我们来介绍用 P、V 操作 解决进程之间的互斥问题。 这里我们给出 解决互斥问题的几个基本的步骤。 首先呢 我们来分析一下并发进程 中间的一些关键活动，也就是涉及到了这些共享变量的这些 代码语句。 而找到了这样一些 涉及到了共享变量的代码和语句之后呢，就把它划定为临界区 针对多个进程的临界区，我们设置一个信号量 mutex。 那么 mutex 这个信号量，它的初值是 1 我们这里简单说明一下，这个mutex 是大家常用于解决互斥问题的时候 给信号量起的这样一个变量名 那么它呢是一个 mutual exclusive，就是互斥这两个词的 前面几个字母的一个拼出来的一个，杜撰出来的一个 词，但是呢现在呢大家一看到这个词基本都认为、都知道 这是互斥量，互斥的意思。 好 设置为初值，也就是 0 和 1 ，二值，那么 允许进临界区还是不允许进，在临界区的前面 要实施 P 操作，P(mutex)，在 出了临界区的时候，要实施 V 操作 V(mutex)，这就是用 P、V 操作来解决进程间互斥问题的一个基本的步骤 我们来看例子，假定有三个进程 P1、P2、P3，那么它们都对同一个 共享变量或者是临界资源进行相应的操作 那么这操作呢分别在进程的不同的部分 那么这段就称之为临界区，所以我们划定好临界区 然后我们设定了一个信号量 mutex，初值是 1 在临界区的前和后，我

们把 P、V 操作加上 我们简单来看一下，如果 不是一般性，我们假设 P1 先上 CPU 那么它在做 P 操作的时候呢，把 mutex 减 1 了，mutex 现在是 0 那么 0 不小于 0，所以 P1 进程就可以进入临界区 如果 P1 进程在临界区的期间被中断了 那么 P2 进程正好上 CPU 它也想进临界区，它也要做 P(mutex)，而 mutex 刚才是 0 了 现在再减 1 就变成负 1 了。 因此，根据我们 的定义，那么 P2 进程就等在 mutex 的这个队列上 那么让出 CPU 之后，假设 P3 进程又上 CPU 了 它呢也要进临界区 又把 mutex 又减去了一次 1，就变成了负 2 了 因此，P3 进程也等在 这个信号量上，等在 P2 的后面 让出了 CPU。 那么假设 P1 又上 CPU 了 然后它就在临界区里头完成了它的工作，出临界区了。出临界区它执行了一个 V(mutex) V(mutex) 呢那么加 1 ，刚才是负 2，加 1 变成负 1，那这个时候 信号量的值呢还是小于等于 0，因此呢这个 V 操作就会到队列里头找到一个进程 P2 把它送到了就绪队列，然后 P1 接着做别的事情。 如果待会 P2 上 CPU 了，那么它就 下一个就进入临界区了。 因为 P 操作已经执行完了，所以它接着就进临界区进临界区，当它出临界区又做一次 V 操作，mutex 就变成 0 了 变成 0 了之后还是小于等于 0 ，所以呢 V 操作就会把 队列里等的 P3 让它进入就绪，就是这样一个过程。 所以用 P、V 操作解决临界区的这种互斥问题是非常简单的