用信号量解决读者/写者问题

问题描述：

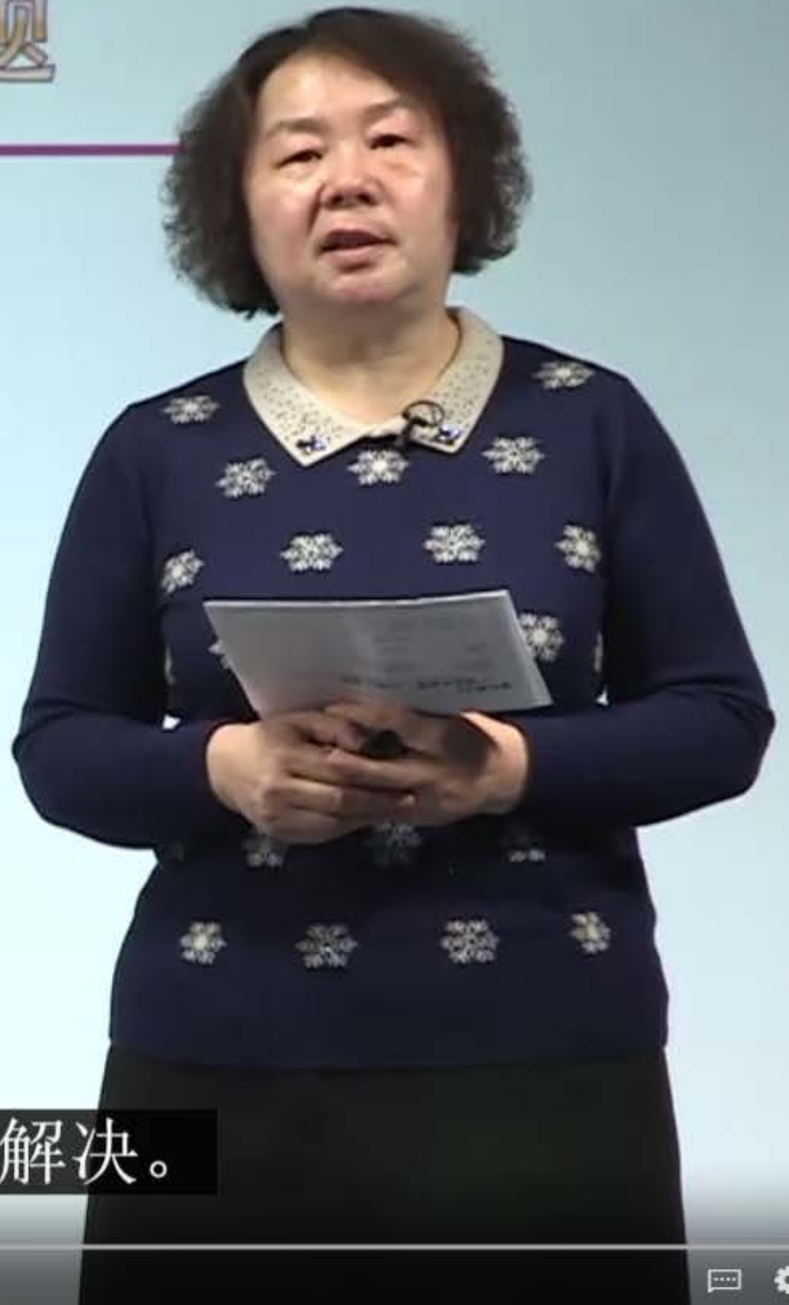
多个进程共享一个数据区，这些进程分为两组：

读者进程： 只读数据区中的数据

写者进程： 只往数据区写数据

要求满足条件：

- ✓ 允许多个读者同时执行读操作
- ✓ 不允许多个写者同时操作
- ✓ 不允许读者、写者同时操作



下面呢我们来看看这个问题怎么解决。

第一类读者写者问题：读者优先

如果读者执行：

- 无其他读者、写者，该读者可以读
- 若已有写者等，但有其他读者正在读，则该读
  也可以读
- 若有写者正在写，该读者必须等

如果写者执行：

- 无其他读者、写者，该写者可以写
- 若有读者正在读，该写者等待
- 若有其他写者正在写，该写者等待

第一类读者-写者问题的解法

```
void reader(void)
{
    while (TRUE) {
    ......
    P (w);
     读操作
    V(w);
    ......
    }
}
```

不需要每个读者都做

```
void  writer(void)
{
    while (TRUE) {
    ......
    P(w);
     写操作
    V(w);
    ......
    }
}
```

因此为了满足前面我们所谈的
条件，我们应该让第一个读者去做P(w)，让最后一个读者做V(w)

第一类读者-写者问题的解法

```
void reader(void)
  {
     while (TRUE) {
     P(mutex);
     rc = rc + 1;
     if (rc == 1)  P (w);
     V(mutex);

      读操作

     P(mutex);
     rc = rc - 1;
     if (rc == 0)  V(w);
     V(mutex);

      其他操作
  }
```

第一个读者

临界区

最后一个读者

```
void  writer(void)

 {
      while (TRUE) {
      ......
      P(w);

      写操作

      V(w);
    }
 }
```

在应用当中，有很多类似的这样一个场景存在。

LINUX提供的读写锁

- 应用场景

  如果每个执行实体对临界区的访问或者是读或者是写共享变量，但是它们都不会既读又写时，读写锁是最好的选择

- 实例：Linux的IPX路由代码中使用了读-写锁，用ipx_routes_lock的读-写锁保护IPX路由表的并发访问

要通过查找路由表实现包转发的程序需要请求读锁；需要添加和删除路由表中入口的程序必须获取写锁（由于通过读路由表的情况比更新路由表的情况多得多，使用读-写锁提高了性能）

本讲重点

- 基本概念
  - 竞争条件、临界区
  - 进程同步、进程互斥
  - 自旋锁（忙等待）
- 信号量、PV操作
- 经典问题模型及解决方案
  - 生产者-消费者问题、读者-写者问题

两个经典的问题模型，一个是生产者-消费者模型，一个是读者-写者模型

本周要求

- 重点阅读教材

  第2章相关内容：2.3.1~2.3.5，2.5.2

- 重点概念

竞争条件　与时间有关的错误　忙等待
临界资源　临界区　进程互斥　进程同步
信号量　P、V操作　锁　自旋锁
生产者消费者问题　读者写者问题

需要大家重点掌握的概念。

0:00 下面呢，我们用信号量，及PV操作，来解决另外一个非常经典的 问题：读者/写者问题。 问题的场景是这样的。 首先有多个进程，它们共享一个数据区 但是这些进程呢，我们把它分成两组 第一组，我们把它称之为"读者进程" 因为它们只去从数据区里头读数据。 另一组呢，我们称之为"写者进程" 它呢，主要呢，只往 数据区里头写数据，所以呢，它们不会 同时有一个进程，既读又写。 那么它们就是分成这两组，读者进程和写者进程。 这是我们面临的一个问题的描述。 下面我们来看一看，对于这样一个场景，我们有哪些 要求，要满足什么条件？那么第一个条件是：容许多个读者同时执行读操作 因为我们知道读操作不会改变数据，所以 所以我们允许多个读者同时执行读操作 但是我们不允许多个写者同时执行 你写我也写，同时写，这肯定是出错的。 也不允许读写同时，就是 有一个读者在读，还有一个写者在往里头写，这个读者是 读的是写之前的呢还是写之后的呢？很显然是不对的。 所以我们给出了这样一个条件。 下面呢我们来看看这个问题怎么解决。 在解决这个问题的时候呢，我们先要明确 它是哪一类读者写者问题。 因为读者写者问题有 好几类，我们这里头介绍的是第一类读者写者问题。 这类问题呢，它呢归结为叫"读者优先"，就是优先读者的这样一类读者写者问题。 我们来看一下 如果读者来执行，那么他可能面临几种情况。 第一种情况 没有其他的读者和写者，那么这个读者就可以读。 第二种情况，是他 来了之后，已经有写者在等待了。 但是呢，有其他的读者在读。 这样的话呢 新来的这个读者，他也可以继续读，也就是说 这就是"读者优先"的这样一个描述。 就是，如果 一个读者到来，有写者在等，但是其他读者已经在读了，还没读完呢，那么新来的读者就可以去读。 第三个情况是，如果他来了一个读者 已经有写者在那写了，那当然了，这个读者就得等待。 好，如果写者来了呢？没有其他的读者写者当然他就可以去写。 如果有读者在读，那么他就得等。 如果有其他的写者在写，他也得等。 所以我们看到，写者是 和另外的写者是互斥的，和读者也是互斥的。 所以，读者写者问题呢，本质上呢，我们可以看到它是一个互斥的问题。 下面我们来看一下，第一类读者写者的一个解法。 那么这个解法呢，由于 读操作和写操作实际上就是一个临界区，所以 非常简单地，就实现了一个临界区的保护 读者呢，在进临界区之前要做一个P(w) 做完了读操作，去执行一个V(w)，写者呢 在写之前做一个P(w)，写完了之后做一个V(w) 那么这就是一个临界区的，一般的临界区的保护。 但是我们所描述的，第一类读者写者问题呢，实际上是要解决的是多个读者可以同时读，因此 我们会看到，不需要每个读者都去做P(w)或做V(w) 那么究竟是谁来

做P(w)谁来做V(w)呢？那么经过分析我们看到，第一个读者 他到来的时候，如果没有其他的读者和写者，他可以去读 他在读之前首先要把临界区保护起来，所以第一个读者是要做P(w)的 而后续的读者，只要前面有读者在读，那么后续来的读者都可以去读。 当所有的读者都读完了， 那么就会把临界区还回来，那么这样的话呢就是最后一个读者 去做V(w)操作。 因此为了满足前面我们所谈的 条件，我们应该让第一个读者去做P(w)，让最后一个读者做V(w) 我们把代码改成这样一个 版本，大家可以看到，在 读者里头我们引入了一个计数器，rc 记录了现在有几个读者。 来了一个读者rc + 1 判断一下他是不是第一个读者 如果rc=1，表示他是第一个读者，他就去做P(w)的工作 那么当 我们所看到，就是最后一个读者走的时候 因为每个读者离开都要去rc- 1 当最后一个读者做完rc- 1， rc就等于0了 所以最后一个读者就要去做V(w)操作 好，那么看似 解决了第一类读者写者的问题，但是会引出新的问题。 因为我们是多个读者，所以多个读者都对rc进行相应的操作 所以rc就成为了一个新的临界资源 那么rc=rc+1判断rc，或者rc=rc-1判断rc 都是一个临界区。 因此我们还要针对这样一个临界区 再增加一个互斥的信号量。 因此我们要对rc 它的进行保护，啊，rc这段代码，进行保护。好，那么这就是第一类读者写者问题的一个解决方案。那么读者写者问题，为什么要有这样一个问题呢？ 实际上呢，是因为，在我们的操作系统当中 在应用当中，有很多类似的这样一个场景存在。 我们来举一个例子，在LINUX当中 其实就是根据读者写者问题的解决方案呢 提供了一个锁，叫读写锁 读写锁的应用场景呢，是这样一个场景 每一个执行的实体，比如说执行的进程 对临界区的访问，它或者是读，或者是写 它不会既读又写 如果是在这样一个场景的情况下，读写锁是一个最好的选择 我们来看一个例子，路由器 经常在网络上需要查路由表 查路由表干什么呢，比如说选择一个，我要发一个包 到网络的另外一端，那么我需要通过路由表来选择我的路径 那我要去读这个路由表，所以读的工作是非常多的，读表的工作是非常多的 那么如果一旦我的这个网络拓扑发生了改变， 那么我就要去重新修改路由表，要添加一些路径或者是删除一些路径 因此，改路由表的工作也会有，但是呢不常发生。 Linux实际上就提供了一个读写锁，如果我们用读写锁，也就是读者写者问题的解决方案 来对路由表进行相应的操作，如果读路由表的时候我们就用读锁 写路由表的时候我们就用写锁 由于读的时间多，读的各种各样的情况， 更多地发生，因此我们用不同的锁 来对路由表进行保护呢，实际

上呢，是提高了这样一个读写操作的这样的一个 性能，像Linux里头，实际上就是提供了这样一个 读写锁，来在对路由器的路由表 的操作过程中呢来并发访问。 那么本讲的重点呢， 主要有这几个基本概念，竞争条件 临界区，进程同步，进程互斥， 还有一个，基于忙等待这样一个思路的自旋锁 本讲当中还有一个非常重要的是介绍了一种同步机制 非常经典：信号量及PV操作 并且呢，用信号量及PV操作解决了 两个经典的问题模型，一个是生产者-消费者模型，一个是读者-写者模型 这里给出了重点阅读的教材，和需要大家重点掌握的概念。 本讲的内容呢就介绍到这里，谢谢大家。