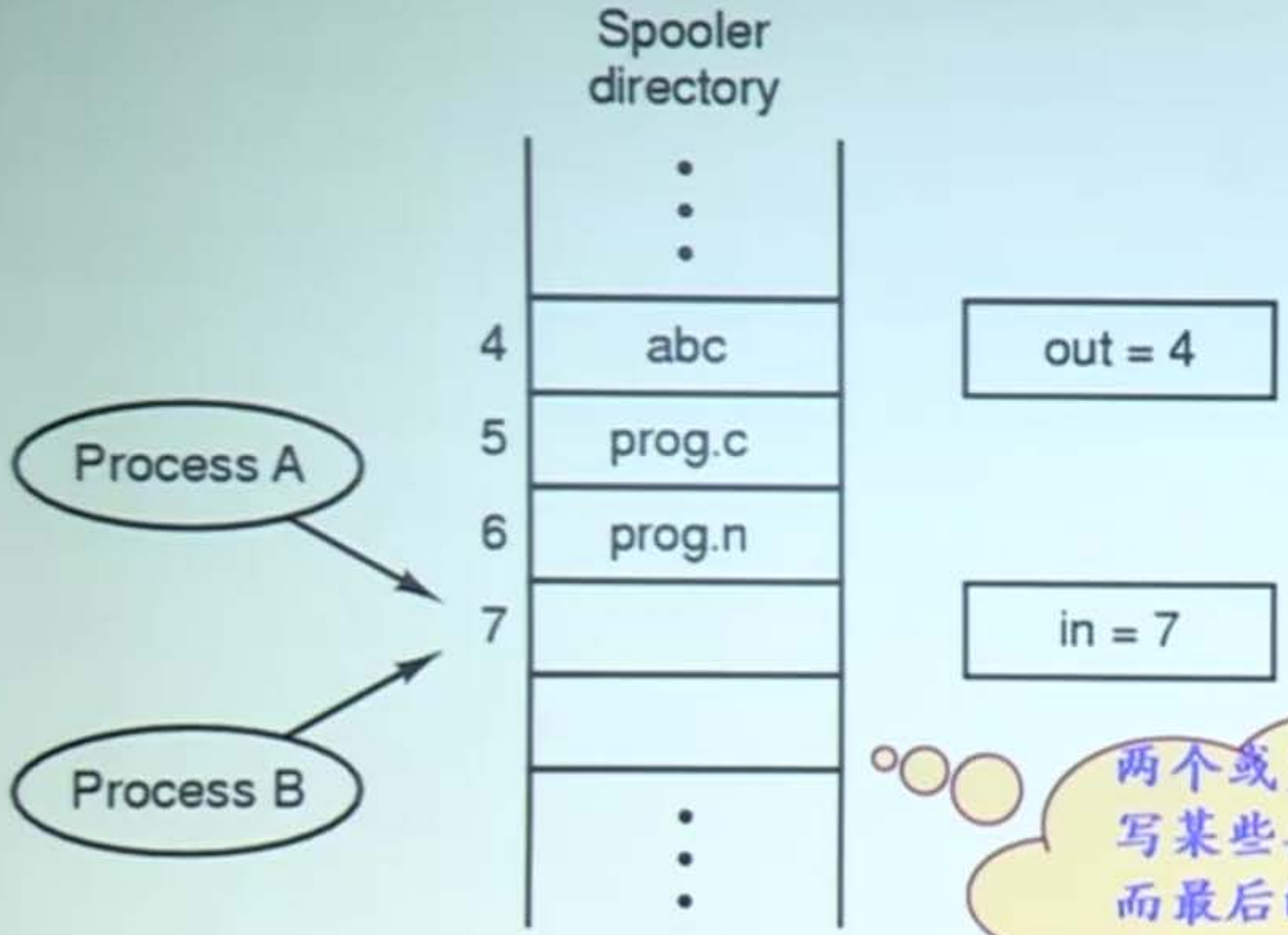
临界资源、临界区

进程互斥

下面我们来介绍一个非常重要的概念，进程互斥 首先呢

竞争条件（RACE CONDITION）



两个或多个进程读写某些共享数据，而最后的结果取决于进程运行的精确时序

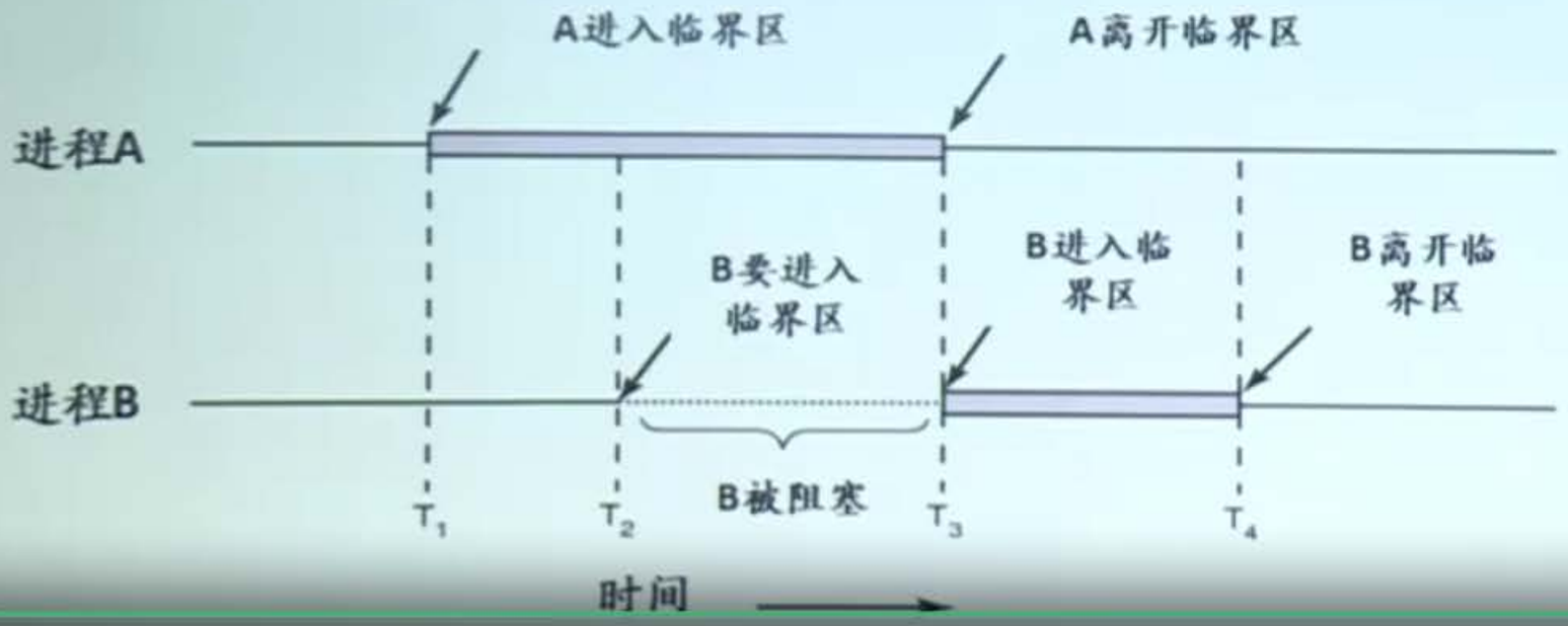
进程互斥(MUTUAL EXCLUSIVE)

- 由于各进程要求使用共享资源（变量、文件等），而这些资源需要排他性使用
  各进程之间竞争使用这些资源
  —— 这一关系称为进程互斥
- 临界资源：critical resource
  系统中某些资源一次只允许一个进程使用，称这样的资源为临界资源或互斥资源或共享变量
- 临界区（互斥区）：critical section(region)
  各个进程中对某个临界资源（共享变量）实施操作的程序片段

互为互斥区，有了互斥区

临界区（互斥区）的使用原则

- 没有进程在临界区时，想进入临界区的进程可进入
- 不允许两个进程同时处于其临界区中
- 临界区外运行的进程不得阻塞其他进程进入临界区
- 不得使进程无限期等待进入临界区



实现进程互斥的方案

- 软件方案
  - Dekker解法、Peterson解法
- 硬件方案
  - 屏蔽中断、TSL(XCHG)指令

那么待会儿呢，我们会来介绍 软件方案和硬件方案

0:00 下面我们来介绍一个非常重要的概念，进程互斥 首先呢 我们来引入一个竞争条件这样一个概念 那么这张图呢 其实是反映了一个 Spooling 技术，用于 打印机，啊，这样一个，这个场景的这么一个示意 那我们知道 Spooling 技术用于打印，那么就需要 维护这么一个打印的目录，啊，就叫 Spooler 的一个目录 那么有一个打印机的守护进程 那么随时呢，看管着这个目录 这里头呢，就存放了所有的进程，需要打印的这种文件的文件名 当这个缓冲区里头 有这个文件名的时候，那么这个打印进程就工作，没有 文件名的时候，打印机可以去睡眠，啊是这样一个场景 好，我们来看一下，那么对于这样一个目录，我们需要用一个 in 这个变量 来表示当前哪一个缓冲区的槽是空的 这样的话呢，那么进程可以把要打印的文件名 送入到这个槽中，那么假设有这么一个情况啊 进程 A 和进程 B 都需要打印了，好，我们来看一下 假如说进程 A 它先从 in 这个变量里 读到了当前的这个，可以存放啊，文件名的这个 啊索引，比如说是 7 ，那么把它这个值呢，读入到自己的这个 局部变量里头，然后呢 它就去把文件名 啊送到了这个 7 啊这个单元 那么这个时候，它应该把 7 更新到 8 才对，可是在这个更新之前，进程 A 呢 啊被切换下 CPU 它还没有把 in 更新成更新为 8 那么这个时候呢，它下 CPU 之后 假如说啊进程 B 要上 CPU 进程 B 上 CPU 的时候呢，它也要打印，所以呢 它也从 in 里头得到了一个 7 这个值 因此，进程 B 就会把它要打印的文件名呢，也送到 7 号啊这个单元 那么就把刚才进程 A 送的那个文件名就覆盖掉了 那么进程 B 呢 把 in 改成 8 之后，去做别的事情去了 那我们可以看到，那么在 7 这个单元，先放了进程 A 要打印的文件名 然后随后被进程 B 要打印的文件名给冲掉了 因此呢，进程 A 就再也得不到自己的打印的结果了 那么这就是一种竞争的条件所带来的一个错误的结果 那么什么是竞争条件呢？竞争条件的 定义是这样的，就是说两个或多个进程 在读写某些共享数据的时候，像这个 in 就是共享数据 而最后的结果取决于进程运行的一个精确的时序 就是时间序列，就是它一跟时间是相关的 那么这就是带来了竞争条件 所以竞争条件呢，是由于有这样一个共享的资源，共享的数据，而多个进程 都对这个数据进行相应的操作带来的 那么这样就产生了这样一个概念，啊，进程互斥 由于在一个并发环境里头 多个进程，都要使用一些共享资源，像 一些变量啊，像一些文件啊，而这些资源呢 具有这样一个性质，它需要排他性的使用，也就是说我用 那么另外一个进程就不能用，这是一种排他性的使用 因此，各个进程之间对这个资源的使用 是一种竞争，啊是一种竞争，而这种竞争呢 我们就称之为进程的互斥 啊当一个进程用，另外一个进程用不了，所以它们是排

他的，互斥的 那么这个共享资源是一个核心 这个共享资源呢，我们给它起了一个名字叫做临界资源 有的时候也叫互斥资源 强调它的是互斥使用的，有的时候也就直接用共享资源，共享变量 那么它的特点是一次啊 只能给一个进程使用，那么这种资源，这种 就是共享资源，或者叫临界资源 那么当多个进程 都要使用某一个，同一个共享资源的时候 那么在它的代码里头，会有相应的操作 那么这些代码我们给它起一个名字呢，就叫做临界区 临界区，也叫互斥区，所谓临界区和互斥区呢 是多个进程 多个进程当中，对某一个共享变量，或者临界资源 实施操作的程序片段，那么这才叫做 临界区，也就是说这些程序片段，分散在不同的 进程里头，那么它们的共同的特点，是对同一个共享变量进行一些操作 那么这一段代码，和另外一个进程的这一段代码，他们互为临界区 互为互斥区，有了互斥区 之后呢，实际上我们会看到在互斥区的使用上，是要满足一定的条件的 比如说，当 A 进程要进入 临界区的时候，在临界区里头它还没有出临界区，如果 B 进程 上 CPU 之后也想进临界区 应该不能够让它进去，因为如果 B 进程也进临界区，就会出现 关键活动的交叉，就会出现前面我们所介绍的各种各样带来的错误 因此，当 B 进程想要进临界区的时候呢 由于 A 进程还在临界区里头没有出来，所以 B 进程只能够被阻塞，不能进临界区 大家再回顾一下啊，上一讲我们所介绍的优先级反转问题 那么一个低优先级的进程，它呢 进到了临界区 那么这个时候因为它优先级比较低，那么有更高优先级的进程 就绪的时候呢，就会抢占它的 CPU ，可是它在临界区里头 更高优先级的进程，也上不了 CPU 运行，因为上 CPU 运行它进不了这个 临界区，所以呢 就被阻塞，而在高优先级和低优先级 之间，又源源不断的会有一些中级优先级的，但是呢 又 非常耗时的这样的一些进程在执行，使得低优先级的进程上不了 CPU ，也就不能够 让高优先级的进程尽快上 CPU ，因为它被阻塞，啊所以这个问题就是和我们这个场景相关的 因此，等到 A 进程离开临界区 B 进程才能够进入它的那段代码，啊临界代码 好，这就是一个，描述的一个场景，就是当 A 进程 在临界区的时候， B 进程呢 要想进临界区，是会被阻塞的 那么下面我们给出临界区的使用原则 第一个原则是，如果没有进程在临界区 想进临界区的进程就可以进入临界区 而且不允许两个进程 同时处于临界区中，因此我们这里头就要保证 A 在临界区的时候 B 不能在临界区。 另外一个原则是 临界区外运行的进程，不能阻塞 其他的进程进入临界区。 第四个 使用原则是不得使进程无限期的等待进入临界区，也就是说 你想进入临界区的进

程，应该在有限的时间内，满足它的这个请求 那么在实现 进程互斥的解决方案当中呢，通常呢 我们分为两大类 一类呢 是软件方案，一类呢 我们称之为硬件方案 软件方案实际上是各种各样的解法，那么当然了对编程的技巧要求比较高 而硬件解决方案呢，强调的是用 特定的指令来完成进程互斥这样一个保护的作用 那么待会儿呢，我们会来介绍 软件方案和硬件方案