协作关系

进程同步

# 进程的同步

进程同步：synchronization

指系统中多个进程中发生的事件存在某种时序关系，需要相互合作，共同完成一项任务

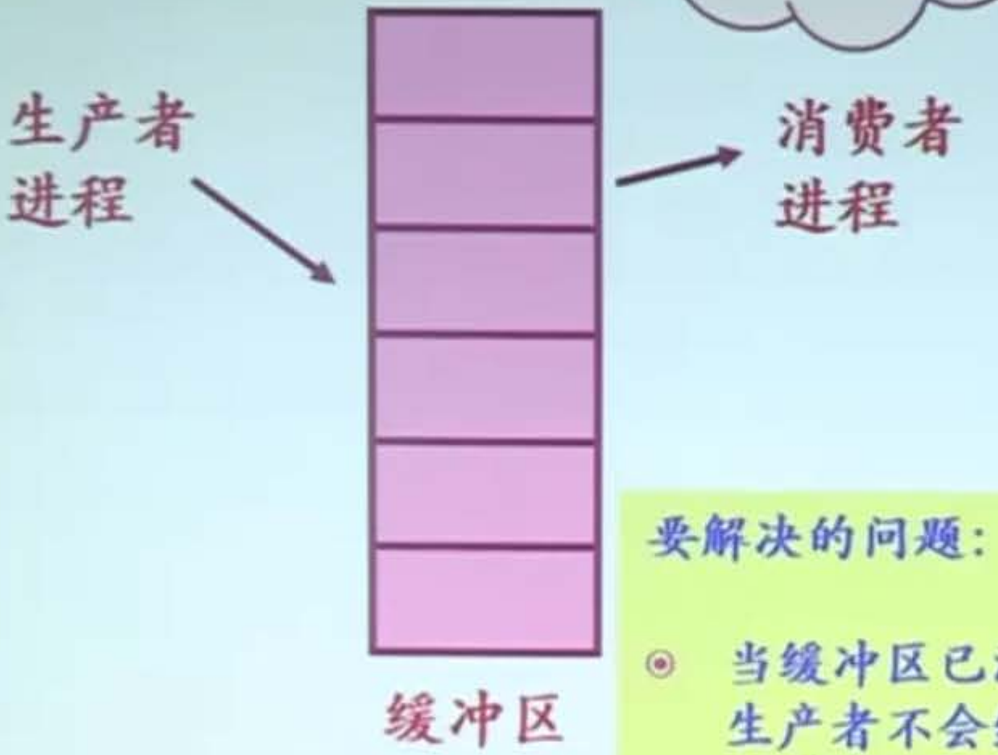
*具体地说，一个进程运行到某一点时，要求另一伙伴进程为它提供消息，在未获得消息之前，该进程进入阻塞态，获得消息后被唤醒进入就绪态*

# 生产者/消费者问题

又称为
有界缓冲
区问题

问题描述：

- 一个或多个生产者生产某种类型的数据放置在缓冲区中
- 有消费者从缓冲区中取数据，每次取一项
- 只能有一个生产者或消费者对缓冲区进行操作




要解决的问题：

- 当缓冲区已满时，生产者不会继续向其中添加数据；
- 当缓冲区为空时，消费者不会从中

- 避免忙等待

睡眠与唤醒操作(原语)

所以有两个操作，我们来看一下，用这样两个操作来解决生产者/消费者的问题

# 生产者/消费者问题

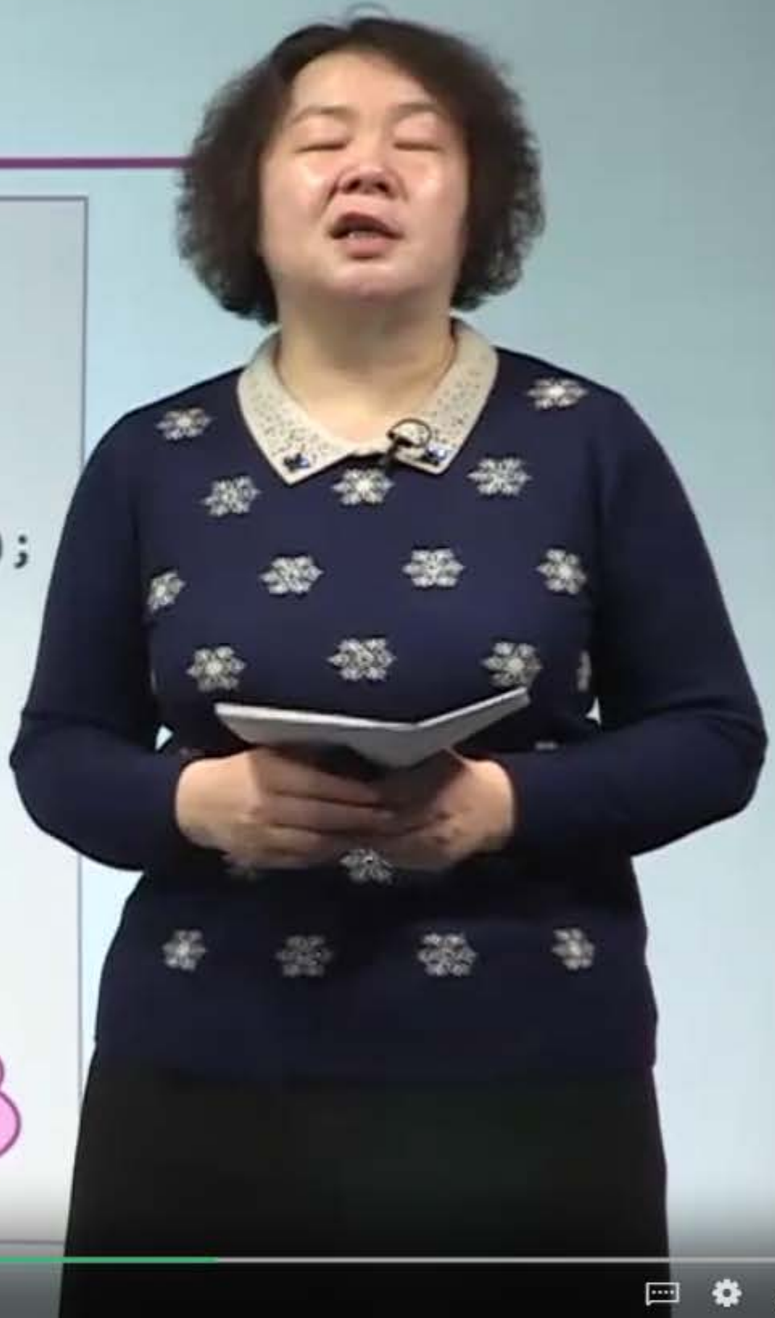
```
#define N 100
int count=0;

void producer(void)
{  int item;
   while(TRUE) {
      item=produce_item();
      if(count==N) sleep();
      insert_item(item);
      count=count+1;
      if(count==1)
       wakeup(consumer);
   }
}
```

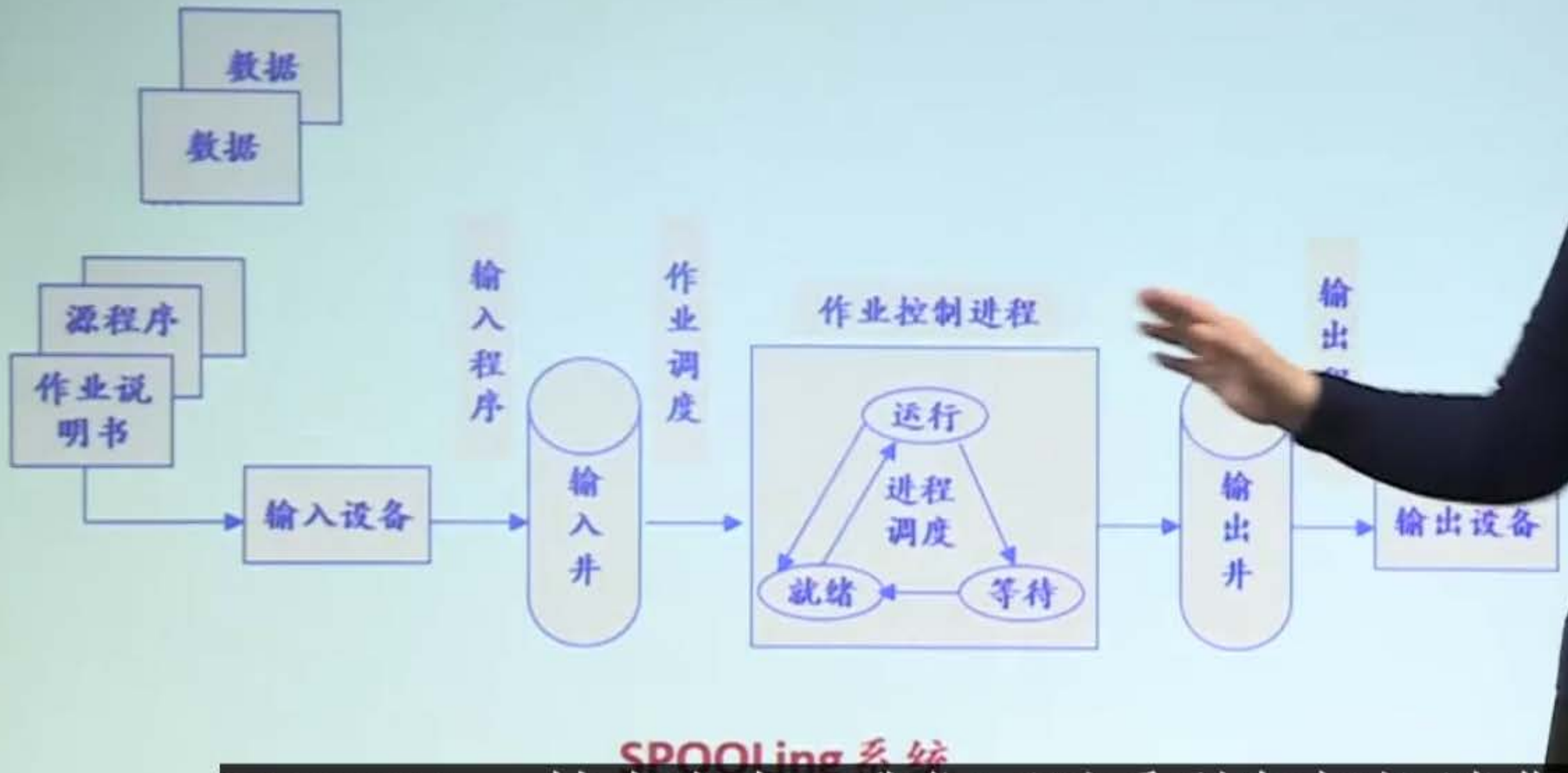
检查 count 的值

```
void consumer(void)
{
  int item;
   while(TRUE) {
      if(count==0)sleep();
      item=remove_item();
      count=count-1;
      if(count==N-1)
       wakeup(producer);
      consume_item(item);
   }
}
```

一种场景：消费者判断count=0后进入睡眠前被切换



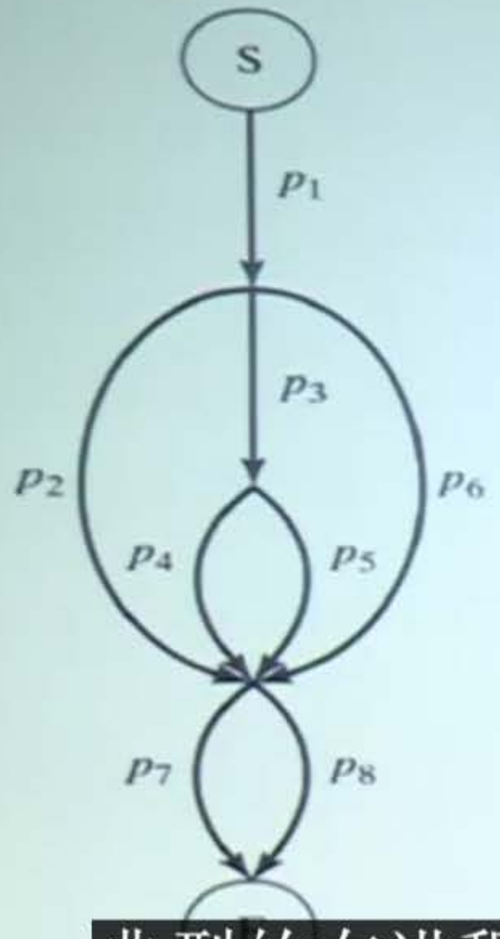
# 其他同步例子1

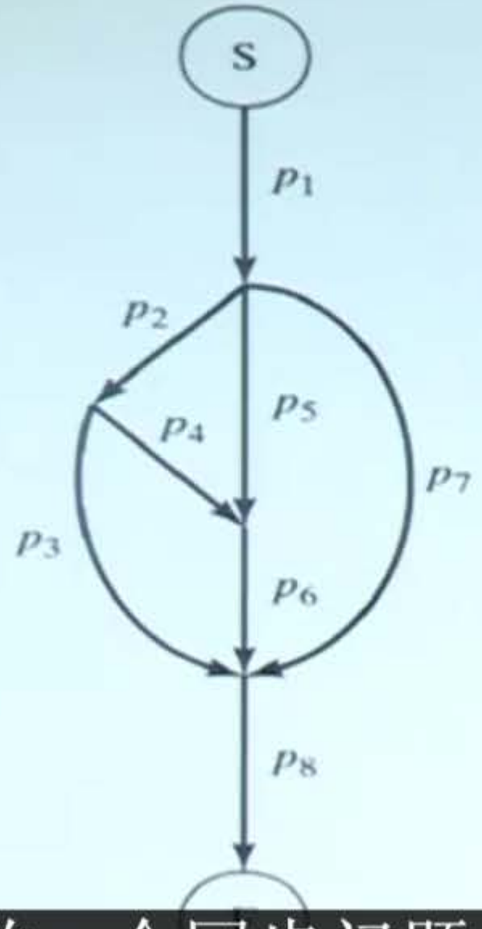



SPOOLing 系统

SPOOLing 技术当中，我们可以看到生产者/消费者的一个影子

# 其他同步例子2




典型的在进程之间的一个同步问题，刚才我们举的那个例子呢

0:00 下面我们介绍一下进程同步的概念 刚才我们介绍的是进程的互斥 它指的是进程之间具有一种竞争关系 而进程的同步呢实际上是指的多个进程之间的协作关系 那么什么是进程的同步呢？ 它是指的在多个进程中发生的事件存在着某种时序关系 需要这些进程相互合作，共同完成一项任务 更具体的一点来说，就是 有多个进程，其中一个进程执行到了某一点 它要求另外一个进程为它提供消息 而另外这个进程呢由于它们是相互合作的，所以是它的一个伙伴进程 因此第一个进程需要第二个进程给它提供消息 在没有获得这个消息的时候，也就是消息没有到达 那么这个事件还没有发生，那么前一个进程 就要进入阻塞态，就要等这个消息，一旦这个消息到达，也就是说这个事件发生了 就把前面这个进程唤醒，让它 重新就绪，实际上我们所谈的是一个进程之间的一个协作关系 所以这个事件可能是各种各样的事件，这些事件之间存在着时序关系 那么一个事件到达了某个时间点 它要等另外一个事件的发生，这样有一个时序关系，这就是进程的同步 那么像我们刚才介绍的 get,copy 和 put 这三个进程，它们就是一种同步关系 也就是说举一个例子，那么当 第一个进程 get 做完第一个循环之后，那么 copy 这个进程才能去做它的第一个循环 就是它如果 copy 的进程先上了 CPU 实际上它也是应该被阻塞的，它要等第一个进程 做完，第一个循环，也就是把某个数据从 F 里头送到 S 里头才行。 好，这是进程的同步，这样一个概念 那我们来再看一个进程同步的 问题，这个问题也是著名的生产者/消费者问题 那么我们把进程分成两类 一类叫做生产者进程，一类叫做消费者进程 那么它们呢都对一个 buffer 进行相应的操作 那么这个问题因为有一个 buffer 介入，所以有时候也称为有界缓冲区域问题 那我们来看看生产者进程和消费者进程它们有什么行为 这个问题是场景是这样的，就是 一个或多个生产者，它生产了某种数据 某种信息，然后要把这个数据或者信息 存放在缓冲区当中，而消费者呢 他的主要工作是从这个缓冲区当中取走相应的数据，然后每次取一个，基本思路是这样 那么如果 两个进程，一个生产者进程，一个消费者进程 同时要对一个 buffer 操作，这呢是不允许的，所以我们说不允许 同时生产者进程和消费者进程对同一个 buffer 进行相应的操作，这样就是对 buffer 的一个保护 所以这里头我们就会看到 在这样一个例子当中实际上有三个问题 那么主要的两个问题是 当缓冲区已经满了的时候 比如说生产者进程，很多生产者进程往 buffer 里送，缓冲区满了 那么后续的生产者进程就不能够再继续往 buffer 里送出去了 第二个问题就是如果 buffer 是空的 消费者上CPU 执行应该是取不到任何数据的，所以当 缓冲区域是空的时候，那么消费者不会从 缓冲

区域中取数据。 第三个问题其实就是前面 已经说过的，就是生产者进程和消费者进程不能同时对 一个 buffer 进行操作，一个里头写，一个去读，这是不允许的 好，那么我们来看这个问题怎么解决。 前面我们已经讲过了 要避免忙等待，如果避免忙等待就是说当我 发现这个条件不成立，不能够继续执行的时候呢，我不会去在那循环地做测试 我会让出 CPU，所以呢我们设计了两个操作 一个呢叫水面，让出 CPU，我自己去睡眠了 另外一个呢是唤醒，当了这是另外一个进程 做完了相应的工作之后，把刚才进入睡眠的这个进程唤醒 所以有两个操作，我们来看一下，用这样两个操作来解决生产者/消费者的问题 我们来简单看一下，对于生产者 那么他首先要做的事情呢是生产一个产品，至于用什么来生产，生产 什么？我们不关心，生产一个产品，把这个数据放到一个变量里头 然后下面要做的事情呢就是要把这个数据送到 缓冲区里头，但是它要看缓冲区是不是 满了，如果满了就不能送了，所以这里判断如果缓冲区 满了，我们用count 来表示缓冲区里头现在 已经装了多少数据的个数，所以 count 初值是 0，然后每送一个加 1，因此如果 count = N，也就是缓冲区满了，这个时候生产者就要睡眠，所以生产者去调用 sleep 来睡眠 那么消费者也是一样，如果消费者上 CPU 它首先要判断有没有数据可以取到，所以呢判断一下 count 是不是 0 没有数据可以取，那么它呢也是调用 sleep，然后睡眠 当然了，当一个进程睡眠就必须有另外一个进程把它从 睡眠状态变成就绪状态，因此我们来看一下，当生产者往 buffer 里 放了一个数据之后，那么它呢要把 这个 count 加 1，增加了一个，那么如果原来 count 是0 ，加完 1 之后 缓冲区里就有数据了，所以判断一下，如果 count = 1，也就是刚才 是 0，刚才是 0，就有可能有消费者在睡眠，因为它是 0 而睡眠 所以呢如果 count 是等于 1 ，就要做一个 wakeup 操作 所以，生产者有义务把某个消费者唤醒。 同样的道理，消费者 消费者消费了一个产品之后，消费了一个 数据之后，那么他去减 1，count 减 1，所以要判断 减完 1 之后是不是等于 N-1，也就是刚才是满的，消费完一个之后呢，变成 N-1 那么刚才是满的，就有可能有生产者睡眠，所以它要根据 count 的值来决定是不是要唤醒一个生产者 所以这是一个生产者/消费者问题， 用这个 sleep 和 wakeup 这样两个原子操作呢来解决 那这里头我们会看到在这里头分别对 count 值进行了判断 那么这个解法会有问题吗？我们来看一个场景 这个场景呢就是说消费者 判断 count 是不是等于 0 ，count 是不是等于0，如果 count 等于 0，它要进入睡眠 但是我

们知道这条语句变成了 汇编，变成了指令一级的时候，就会变成多条指令 所以如果在它判断它 count = 0 还没有去调用 sleep 之前，那么这个时候消费者被切换下 CPU 那么会发生什么情况呢？那么这个时候 count 是 0 ，假设生产者又生产了一个数据，上 CPU 又生产了一个数据 因此这里我们可以看到就不断地去生产 就放到这，就是生产一个数据，肯定不等于 N 吧，然后就放就放，这里头我们可以看到 刚才是 0，现在呢生产者生产了一个数据之后 加完 1 之后，count 加完 1 之后，count 就等于 1 了 所以这个时候生产者一看，count 等于 1，所以做了一个 wakeup 那么由于刚才的 消费者还没有 sleep，所以你做的这个 wakeup 实际上做了一个 空操作，因为没有进程在睡眠，所以就 继续接着执行，生产者继续生产 如果生产者被切换下 CPU，消费者一上来，那么它肯定首先要做的事情先做 sleep 那大家会想到说 这个 sleep 的进程刚才 wakeup 已经做完了，所以就不会再被唤醒了，不会再被唤醒了。 所以在这种情况下呢就有错误发生了 所以呢，这个呢也是没有完全解决生产者/消费者问题 那么关于同步呢，我们还看一些例子，对同步有一个印象 比如说我们前面介绍的 SPOOLing 系统 在 SPOOLing 系统当中呢，我们有很多的进程，比如说有输入 进程，有作业调度，有作业控制进程，还有输出进程 但实际上我们知道这个输入级 使输入进程把一些作业，批处理的作业往输入进程里送 作业调度呢是从输入井去选作业，让它们去运行 那我们可以看到输入程序，如果输入井 输入井的这个区域设置的是一个有限的区域 那么可能源源不断的作业来了之后，就把输入井填满了 输入井如果填满了，那么输出进程实际上就不能再往里头送东西了 这就是说生产者，输入程序在这里就是生产者的作用 而作业调度程序呢实际上就是一个消费者 如果没有作业进入输入井，那么调度程序也没有可选的，所以它也要睡眠 因此在这里头，输入程序、 作业调度是一对生产者/消费者，那同样的道理 我们看到作业控制进程实际上是和输出进程，也是一对生产者和消费者，这个 buffer 缓冲区呢实际上就是输出井 其实呢，作业调度和作业控制进程它也可以看成是一对生产者和消费者 所以呢，这是一个 SPOOLing 技术当中，我们可以看到生产者/消费者的一个影子 那么关于进程同步呢，还有这样一些 情况，我们来看一下，这就是一个纯粹的一个同步问题 我们看说这个，这有几个进程，P1 一直到 P8，8 个进程，这 8 个 进程呢必须满足这样一个关系。 也就是说 P1 全部执行完了 P2，P3，P6 才能执行 然后 P3 执行完了，P4，P5 才能执行 P2，P4，P5，P6 都执行完了，才

能够 P7 或者 P8 它们两有一个执行。它们两可以同时，谁先执行，先后执行都无所谓 但是必须要求 P2，P4，P5，P6 都执行完了 那么也就是说当 P1 到 P8 这 8 个进程 同时并发在系统中执行的过程中，谁都可能先上 CPU 那么你要满足这个同步关系，就不能够让 P8，也就是说在其他都没执行完的时候P8 就执行了，不允许 如果你要满足这个，看看用什么样的解决方案能这样一个同步关系 那这是另外一个同步关系，大家可以看一下，这就是 典型的在进程之间的一个同步问题，刚才我们举的那个例子呢 实际上是进程的中间的某个事件和另外一个进程的某个事件的一个同步关系 这个呢是另外一种例子