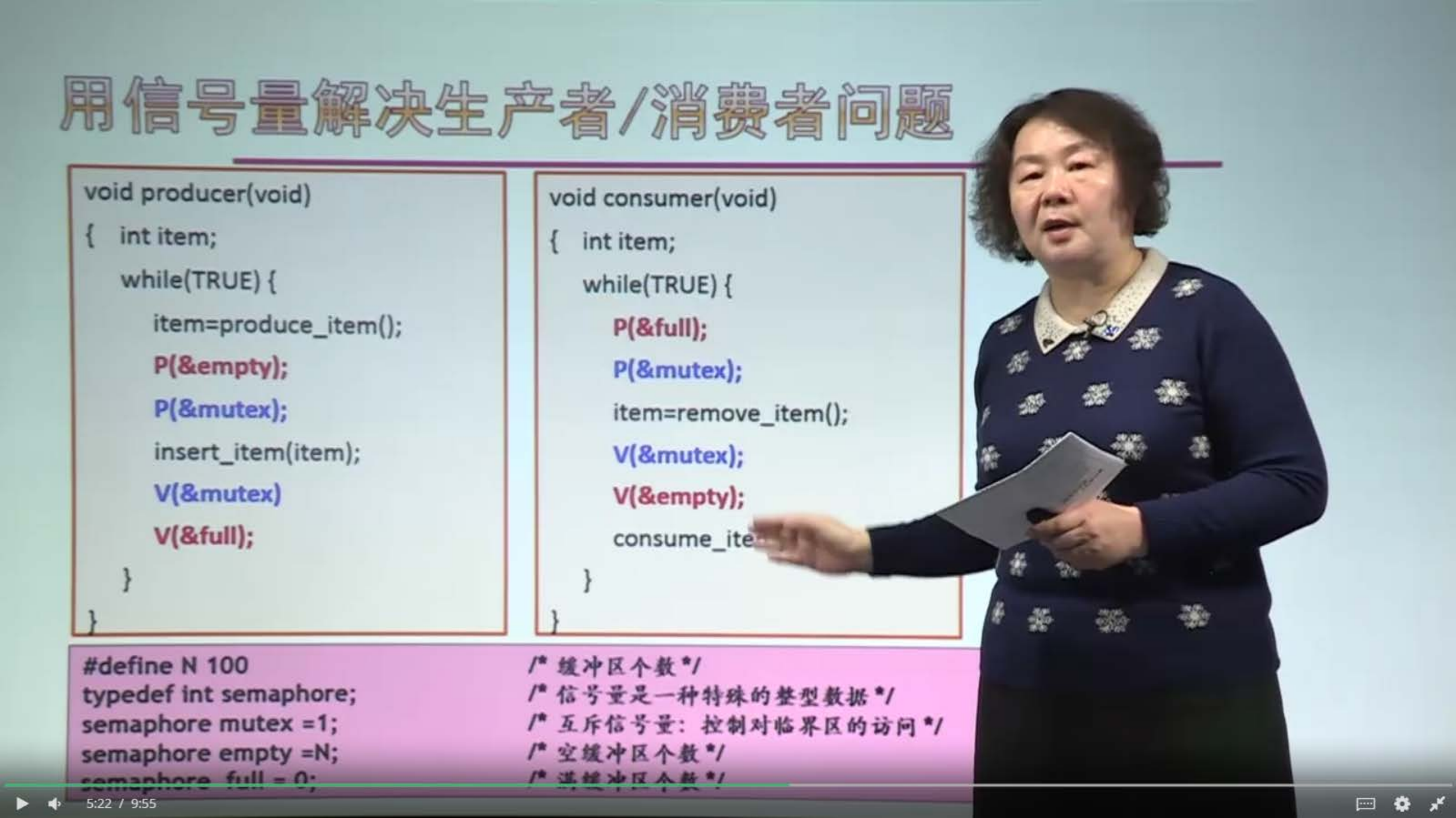
用信号量解决生产者/消费者问题

```
void producer(void)
{   int item;
    while(TRUE) {
        item=produce_item();
        P(&empty);
        P(&mutex);
        insert_item(item);
        V(&mutex)
        V(&full);
    }
}
```

```
void consumer(void)
{   int item;
    while(TRUE) {
        P(&full);
        P(&mutex);
        item=remove_item();
        V(&mutex);
        V(&empty);
        consume_ite
    }
}
```

```
#define N 100                       /* 缓冲区个数 */
typedef int semaphore;              /* 信号量是一种特殊的整型数据 */
semaphore mutex =1;                 /* 互斥信号量：控制对临界区的访问 */
semaphore empty =N;                 /* 空缓冲区个数 */
semaphore full = 0;                 /* 满缓冲区个数 */
```



讨论一下

```
void producer(void)
{   int item;
    while(TRUE) {
        item=produce_item();
        P(&empty);
        P(&mutex);
        insert_item(item);
        V(&mutex)
        V(&full);
    }
}
```

```
void consumer(void)
{   int item;
    while(TRUE) {
        P(&full);
        P(&mutex);
        item=remove_item();
        V(&mutex);
        V(&empty);
        consume_item(item);
    }
}
```

思考：
若颠倒两个P
操作的顺序？

思考：
若颠倒两个V
操作的顺序？

顺序与
位置

0:00 下面呢我们用信号量来解决生产者/消费者问题。这里呢首先给出来的是生产者/消费者的代码框架。生产者的行为呢是生产一个产品并且把这个产品送到缓冲区里。而消费者呢，是从缓冲区里头取产品，并且呢把产品消费掉。那么这里头我们知道生产者是不能够往满的缓冲区里头放东西的。因此我们设置了一个信号量 empty。empty 呢实际上是表示了在缓冲区当中有多少个空缓冲区，它的个数。它的初值呢肯定是 n 也就是换句话说，刚开始的时候 n 个缓冲区都是空的。那么消费者不能够从一个空的缓冲区取东西。因此呢我们设置了另外一个信号量 full。那么 full 的初值是 0，也就是一上来消费者是取不到东西的，刚开始的时候在生产者没有生产产品的时候，那么 buffer 是空的。那么 full 实际上就代表了在 buffer 里头有哪些 buffer 是有内容的，是有产品有数据的。这是满的缓冲区的个数，当然我们还设置了一个信号量 mutex。因为我们不允许同时生产者和消费者都往同一个缓冲区里放。那么一个放一个取，不允许同时做，因此呢我们用一个 mutex 来保护这样一个 buffer。好，那么现在我们来看一下，我们有了这样几个信号量之后，那么我们怎么样把相应的信号量放在一个合适的位置呢。当生产者要把生产的产品往缓冲区里放的时候，它要判断缓冲区是不是满。因此它要通过一个 P(&empty) 来决定能放还是不能放。empty 初值是 n 也就是说它一上来可以放，那么它继续去循环可以放多少个呢？假如生产者一直在 cpu 上执行，那可以循环 n 次，也就是可以往 buffer 里放 n 个数据。当它把 buffer 放满之后，那么 empty 每次减一，每次减一减到 0 了。第 n+1 次循环的时候，那么 empty 值已经是零，再去减一，P(&empty) 再去减一。那么我们知道这个时候已经是负一了，负一那么生产者就被阻塞了，被阻塞了，这就是 empty 这个信号量，当然了如果被阻塞，什么时候把它从阻塞态进入就绪态呢，那么要看消费者。消费者取走一个产品之后，就去对 empty 做一次 V 操作，就加一，取走一个就加一。那么如果它被阻塞了，加完一之后就被释放了。所以这边呢是 P 操作，这边是 V 操作，我们已经看到了在解决这个同步问题的时候，那么 P 操作和 V 操作，对同一个信号量的 P 操作和 V 操作是分散在两个不同的进程里头的，这就是同步。同样道理，P(&full) 和 V(&full) 啊，消费者在取数据之前，先要判断能不能取，有没有内容可以取到。如果有就取，如果没有就停在这里。被阻塞，那我们可以看到 full 的初值是 0，也就是说如果在这样一个并发环境下，消费者先上 cpu 了。那很显然它应该什么都取不到，所以呢它停在这里的。那么如果上 cpu 的时候，生产者已经生产了很多的数据，那么消费者就能取到，因为每

次 生产者生产完数据放到缓冲区之后就要做一次 V 操作，所以 full 的值就会加一加一加一。 那么消费者在上来的时候可以取到数据。 那么我们在往 buffer 里送 数据或者从 buffer 里取数据的时候呢不允许同时啊 又送又取，因此呢我们对它进行了一个 互斥，意思就是说对 buffer 的操作实际上是一个互斥的。 我们把用 mutex 啊，P 操作 V 操作呢实际上是 保护这个 buffer ，那大家可以看到 互斥的操作，那么一个 P 操作，一个 V 操作，它们是在同一个进程里头。 也就是先做 P 操作，临界区出去之后再去做 V 操作。 那么同步呢，可以看到是两个不同的进程，一个进程做 P 操作，一个进程做 V 操作。 下面呢我们讨论一下刚才的生产者、 消费者解决方案。 我们从两个角度来讨论，一个是顺序， 一个是位置，我们看到， 这里有两个 P 操作，一个是同步的一个是互斥的。 消费者也有两个，那么 这两个 P 操作，它们的顺序是不是可以 改变一下呢，变换一下呢，所以大家思考一下， 这两个 P 操作的顺序是不是可以颠倒。 我们假设有这样一个场景， 把它颠倒了， 把消费者的 P(&mutex) 先执行，再执行 P(&full)。 假如说这个时候 buffer 是空的， 那么消费者上来 先执行的 P(&mutex) ， mutex 值等于 0 了，然后 接着执行 P(&full) ，而这个时候我们知道 消费者就等在 full 这个信号量上了，因为 buffer 现在是空的。 而这时候如果让出了 cpu 之后， 那么假设生产者来生产，那么生产者呢 P(&empty) 那么一看，哎有空 buffer 可以生产。 那么它要接着往 buffer 里送的时候，我们知道 它要执行 P(&mutex)，可是 mutex 已经是 0 了，再去执行 P(&mutex) ，那么 mutex 等于负一， 所以按照 P 操作定义，那么生产者 也要等待，当然是等待进入临界区。 一方面消费者等产品， 一方面生产者呢， 就要想放产品，但是要进不了临界区，所以它们两个进程谁也不能往前执行。 这就是出现了死锁问题。 好，那我们来看看 两个 V 操作的顺序可不可以颠倒呢？ 因为 V 操作呢，只是把信号量的值加一。 然后看一看有没有进程等在队列里头，如果有就把它释放。 因此 V 操作， 不会使得调用 V 操作的这个进程 进入等待状态，所以这两个的顺序是可以颠倒的。 那么颠倒了结果呢， 可能会带来其他的一些问题，比如说，如果 我没有，我先做的是 V(&empty) 再去做 V(&mutex) 那么临界区里头就会多一点点指令。 那么其他的进程想进临界区可能会稍微晚一点进临界区。 也就是不会出错，所以当然这样的顺序 是最理想的，因为它把临界区界定在最小的范围内。 那我们再来看看位置，按照刚才的说法，那我们说 消费一个产品，啊这是有一个一堆语句。

那么消费一个产品能不能放在这个位置呢？ 放在取完产品之后就立刻消费呢？ 肯定没有错误，但是也是把临界区的范围扩大了。 那么也就是说临界区的范围扩大就在临界区待的更长的时间。 其实它可以不需要在临界区里做的事情尽量不要在临界区里做。 所以， 作为顺序来讲，作为位置来讲，那么像 消费产品的这样一堆代码尽量不要往这儿放。 那么同样生产产品呢也不要放在这个位置，放在这个位置呢 也会扩大临界区，造成在临界区的时间过长。