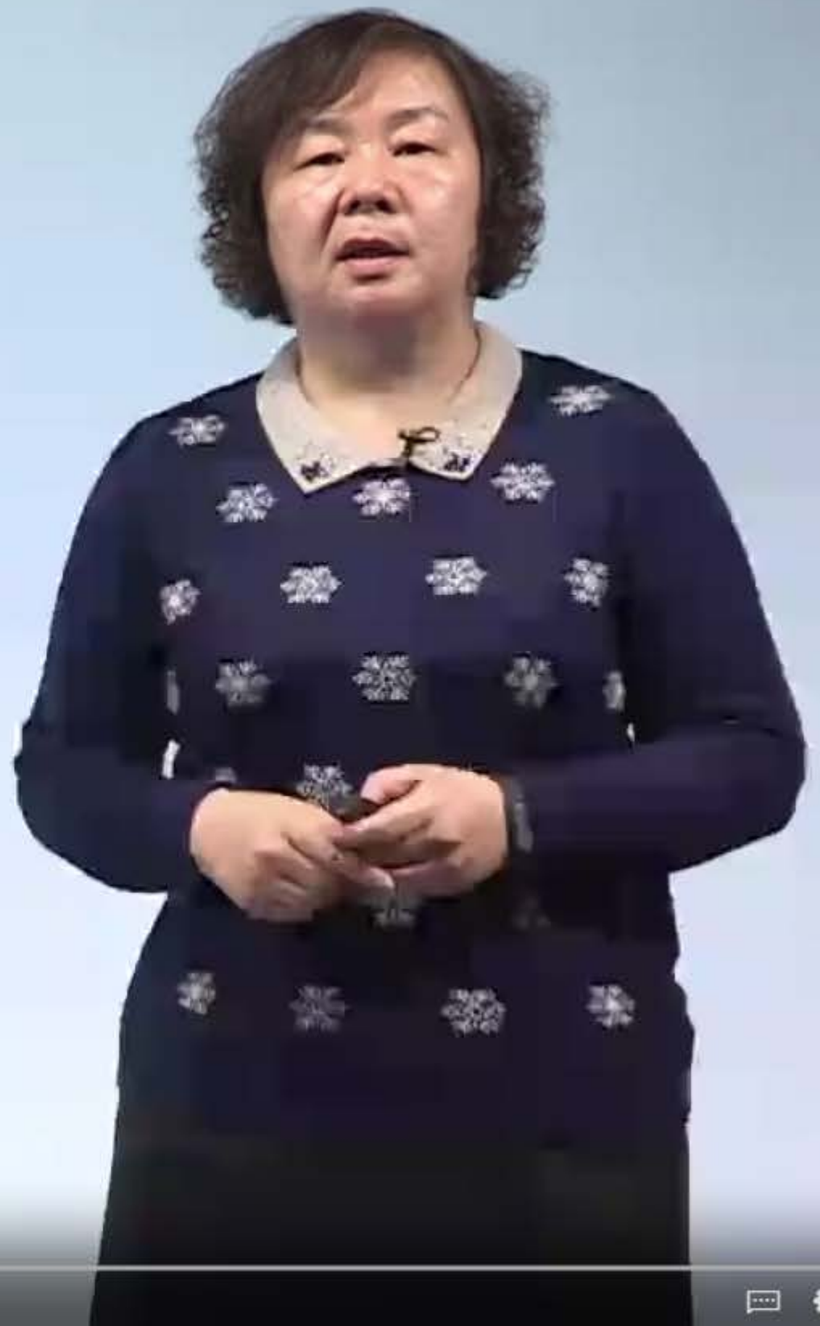
锁、条件变量

PTHREAD中的同步机制



PTHREAD中的同步机制

Thread call	Description
Pthread_mutex_init	Create a mutex
Pthread_mutex_destroy	Destroy an existing mutex
Pthread_mutex_lock	Acquire a lock or block
Pthread_mutex_trylock	Acquire a lock or fail
Pthread_mutex_unlock	Release a lock

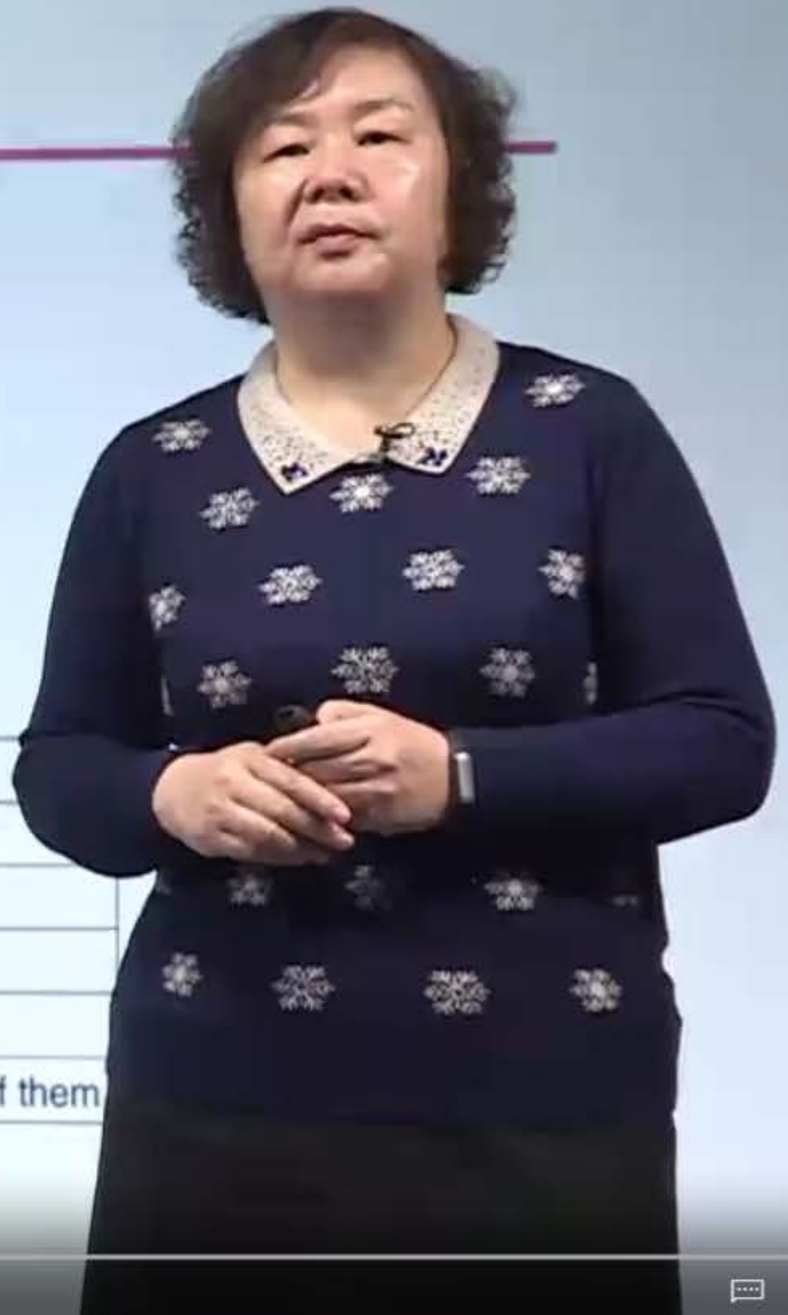
互斥量

保护临界区

Thread call	Description
Pthread_cond_init	Create a condition variable
Pthread_cond_destroy	Destroy a condition variable
Pthread_cond_wait	Block waiting for a signal
Pthread_cond_signal	Signal another thread and wake it up
Pthread_cond_broadcast	Signal multiple threads and wake all of them

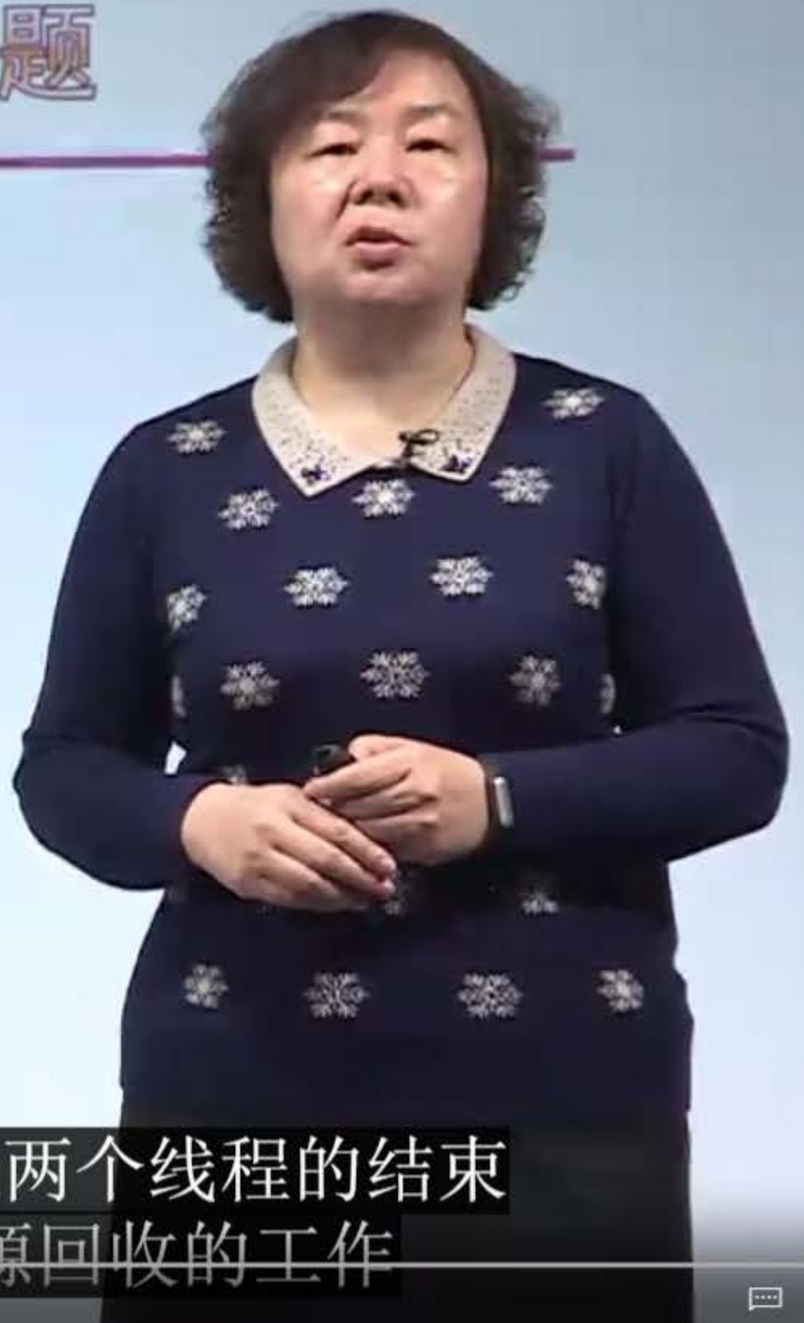
条件变量

解决同步



用PTHREAD解决生产者-消费者问题

```
int main(int argc, char **argv)
{
        pthread_t pro, con;
        pthread_mutex_init(&the_mutex, 0);
        pthread_cond_init(&condc, 0);
        pthread_cond_init(&condp, 0);
        pthread_create(&con, 0, consumer, 0);
        pthread_create(&pro, 0, producer, 0);
        pthread_join(pro, 0);
        pthread_join(con, 0);
        pthread_cond_destroy(&condc);
        pthread_cond_destroy(&condp);
        pthread_mutex_destroy(&the_mutex);
}
```

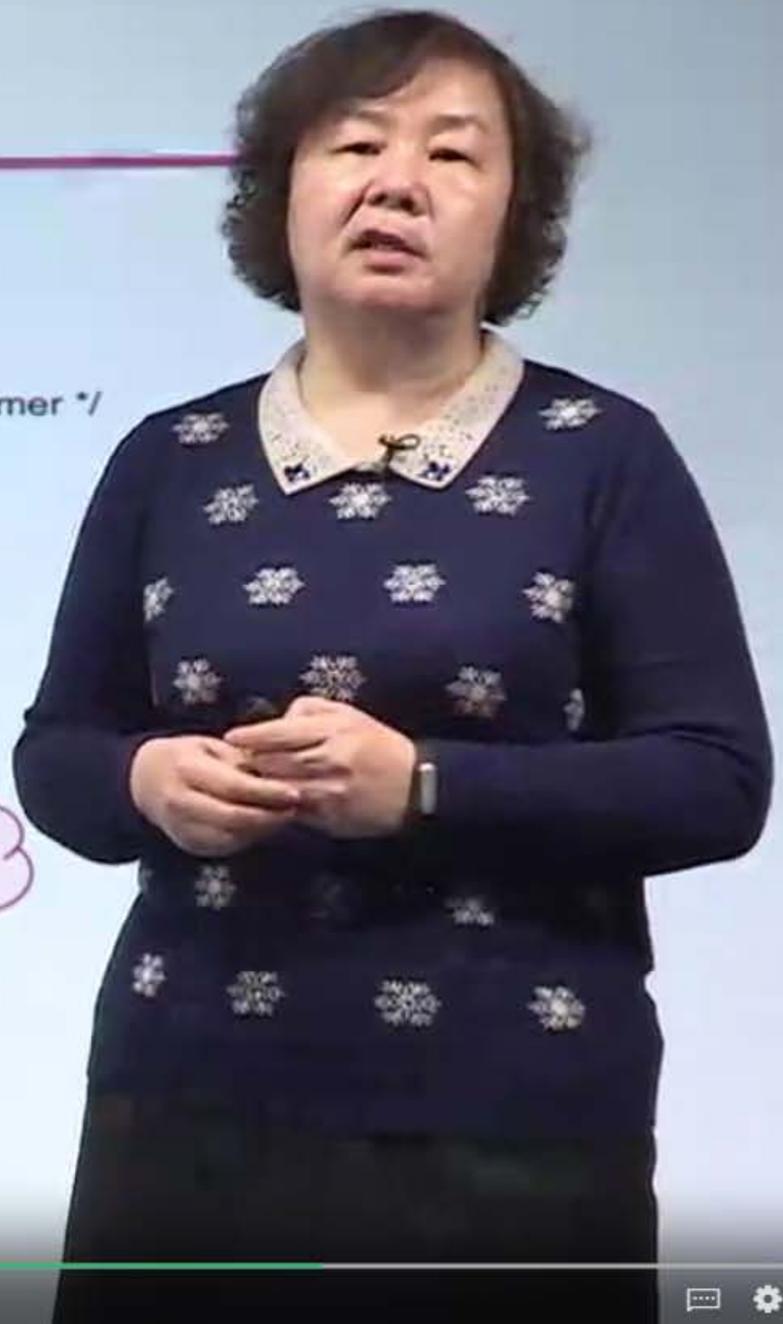


然后呢，这个主线程实际上就在等待这两个线程的结束
我们其实结束之后做一些，啊，资源回收的工作

生产者-消费者问题

```
#include <stdio.h>
#include <pthread.h>
#define MAX 1000000000                          /* how many numbers to produce */
pthread_mutex_t the_mutex;
pthread_cond_t condc, condp;
int buffer = 0;                                 /* buffer used between producer and consumer */

void *producer(void *ptr)                       /* produce data */
{       int i;

        for (i= 1; i <= MAX; i++) {
                pthread_mutex_lock(&the_mutex);   /* get exclusive access to buffer */
                while (buffer != 0) pthread_cond_wait(&condp, &the_mutex);
                buffer = i;                       /* put item in buffer */
                pthread_cond_signal(&condc);      /* wake up consumer */
                pthread_mutex_unlock(&the_mutex);/* release access to buffer */
        }
        pthread_exit(0);
}

void *consumer(void *ptr)                       /* consume data */
{       int i;

        for (i = 1; i <= MAX; i++) {
                pthread_mutex_lock(&the_mutex);   /* get exclusive access to buffer */
                while (buffer ==0 ) pthread_cond_wait(&condc, &the_mutex);
                buffer = 0;                       /* take item out of buffer */
                pthread_cond_signal(&condp);      /* wake up producer */
                pthread_mutex_unlock(&the_mutex);/* release access to buffer */
        }
        pthread_exit(0);
}
```

一个缓冲区



讨论：PTHREAD_COND_WAIT

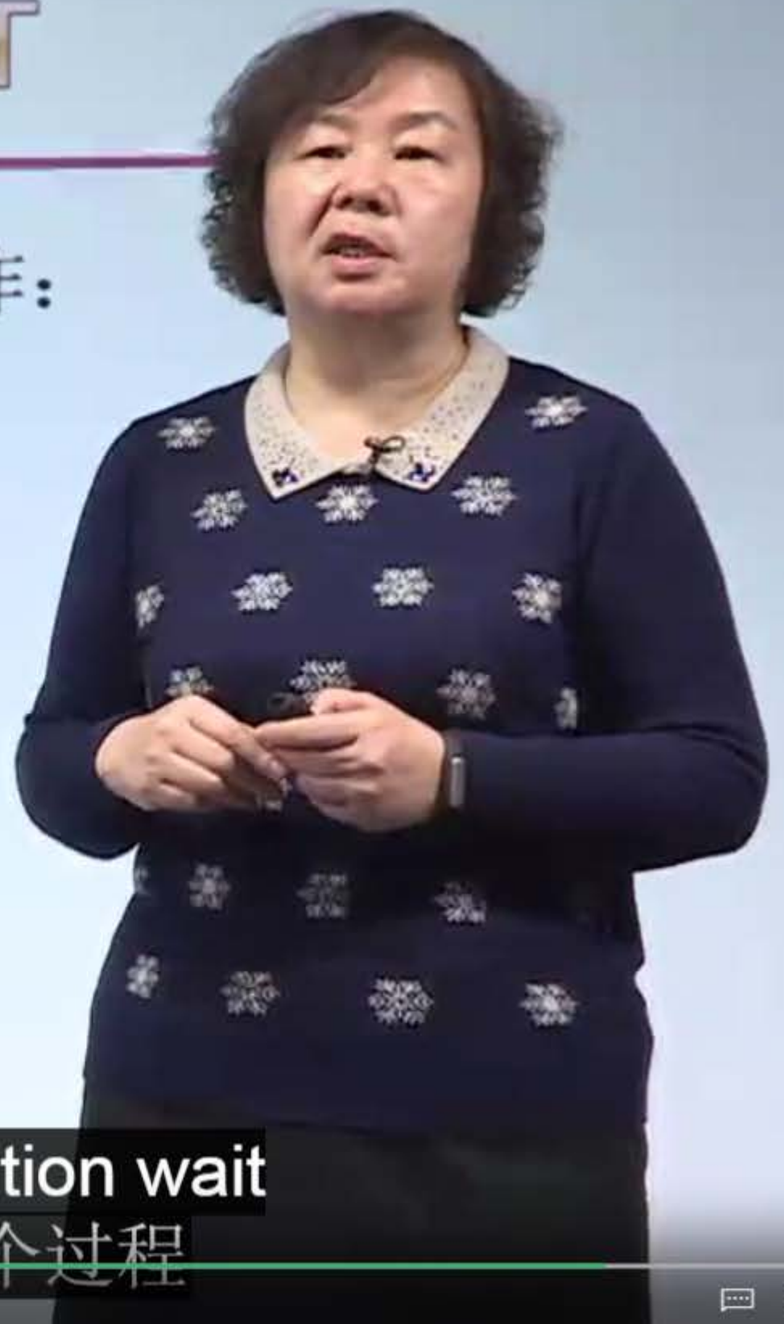
- pthread_cond_wait的执行分解为三个主要动作：

  1、解锁

  2、等待

  当收到一个解除等待的信号（pthread_cond_signal或者pthread_cond_broad_cast）之后，pthread_cond_wait马上需要做的动作是：

  3、上锁

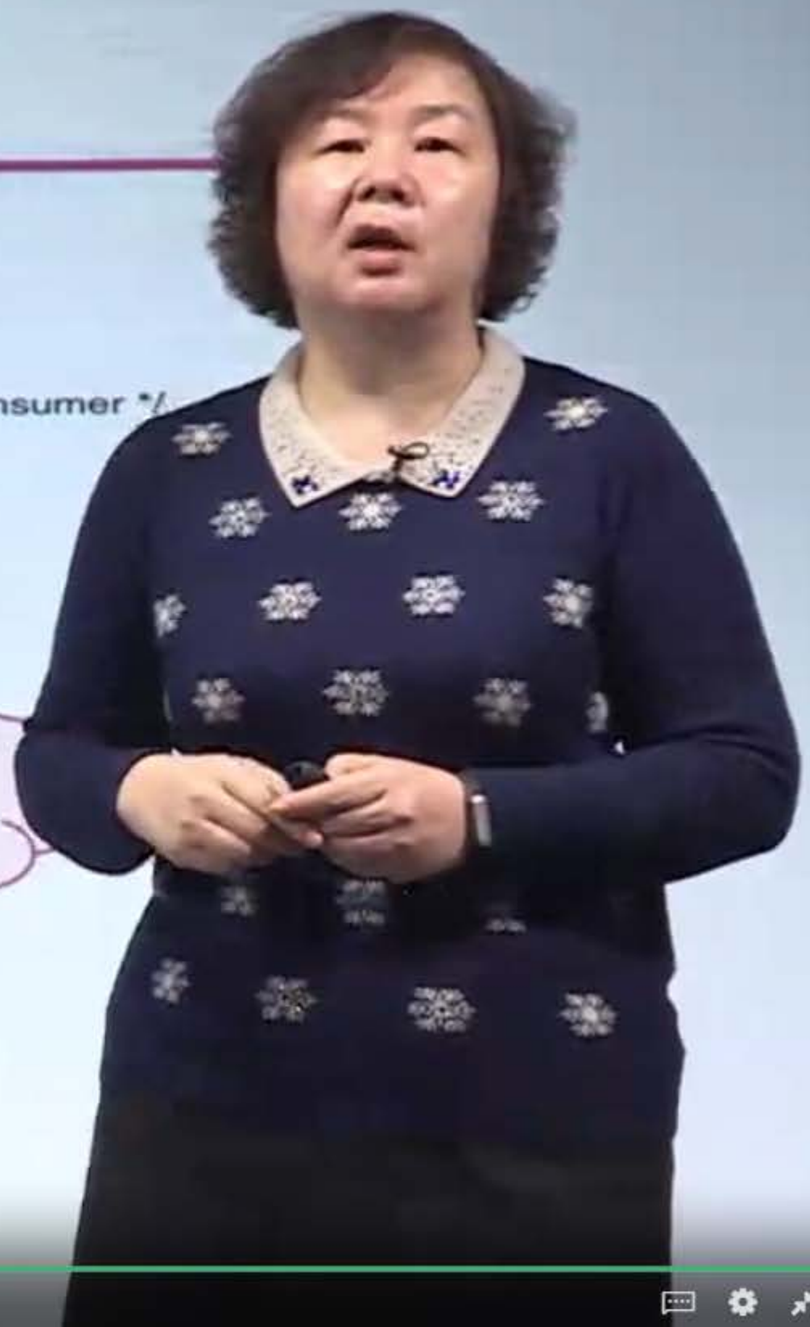


所以这里头我们可以看到，condition wait
是这样一个过程，啊，这样一个过程

生产者-消费者问题

```
#include <stdio.h>
#include <pthread.h>
#define MAX 1000000000                          /* how many numbers to produce */
pthread_mutex_t the_mutex;
pthread_cond_t condc, condp;
int buffer = 0;                                 /* buffer used between producer and consumer */

void *producer(void *ptr)                       /* produce data */
{       int i;

        for (i= 1; i <= MAX; i++) {
                pthread_mutex_lock(&the_mutex);     /* get exclusive access to buffer */
                while (buffer != 0) pthread_cond_wait(&condp, &the_mutex);
                buffer = i;                         /* put item in buffer */
                pthread_cond_signal(&condc);        /* wake up consumer */
                pthread_mutex_unlock(&the_mutex);/* release access to buffer */
        }
        pthread_exit(0);
}

void *consumer(void *ptr)                       /* consume data */
{       int i;

        for (i = 1; i <= MAX; i++) {
                pthread_mutex_lock(&the_mutex);     /* get exclusive access to buffer */
                while (buffer ==0 ) pthread_cond_wait(&condc, &the_mutex);
                buffer = 0;                         /* take item out of buffer */
                pthread_cond_signal(&condp);        /* wake up producer */
                pthread_mutex_unlock(&the_mutex);/* release access to buffer */
        }
        pthread_exit(0);
}
```

一个缓冲区



8:19 / 8:21

0:00 下面呢，我们来介绍一下线程库 Pthread 当中，它的同步机制 也就是它如何来解决同步互斥问题 那我们看一下，解决同步互斥问题 那么，互斥的问题呢，在 Pthread 当中呢，是用一个互斥量 啊，这样一个概念，那么通过对互斥量提供相应的操作呢，来保护 临界区。 我们来看一下这里头这是创建，啊，一个互斥量 那么还可以把它，啊，销毁啊，销毁 那么在互斥量上我们可以实施的操作呢 加锁，啊，就实际上是加锁 然后呢，解锁，啊，解锁 那么这是两个典型的加锁解锁啊 用互斥量，实际上就是对临界区，进临界区之前加锁，出临界区解锁 那么这里头还有一些其他的操作比如说 试图加锁，如果加上了，那么这个锁就锁住了 如果没加上呢，原来那么我们说，啊 如果加锁没加上，那我们就可能会去等待，啊，会去等待 那么这里头呢就是说，如果没有加上的话呢，我们提供的就是你呢就 出错返回，啊出错返回，啊就是说汇报说没加上，啊就可以 返回，那么用户编程序的时候呢，可以灵活利用，啊，这样一些函数 那么解决同步问题呢，它就用的是 条件变量，以及在条件变量上的各种操作 条件变量上呢这个操作呢，我们知道它是解决同步问题的 这里头我们可以看到，那么这里可以创建一个条件变量，也可以 啊，销毁一个条件变量，那么在条件变量的操作呢，可以是 wait signal 或者是 broadcast，就是我们前面 已经大致上，啊，都介绍了这些的做法，那么 wait signal 这一对，那么也可以用 broadcast 啊，来 广播，啊，广播。 所以这是 Pthread 上的同步机制 那我们依然用生产者消费者 为例，看看用 Pthread 库当中提供的同步机制来解决生产者消费者问题 这里头呢我们来看一下，这是一个，啊初始化的和一些啊 前面的一些过程，比如说我可以初始化一个互斥量，啊 the mutex 就是我的互斥量，啊，用于保护临界区用的 然后呢，我有两个条件变量，一个 condc 一个 condp 啊，这样是两个条件变量，啊，将它初始化，啊，初始化 然后创建了生产者线程和消费者线程 然后呢，这个主线程实际上就在等待这两个线程的结束 我们其实结束之后做一些，啊，资源回收的工作 那我们来看一下这个具体的，啊这个 解决方案，我们来看一下。 这个假设啊，只有一个缓冲区，因为前面我们介绍是多个缓冲区 为了简化啊，我们主要是看这个同步机制是怎么用的，所以我们就设定了一个缓冲区 好，我们来看一下，这几个这是 互斥量，这两个是条件变量 那我们来看一下这个使用，它的这个使用 方法啊，应用的方法，和前面是有点，有所不同的 因为首先呢，我们来看，作为一个这是生产者 生产者呢，他要往 buffer 里头放，啊 数据，那么放数据如果是 buffer 满了，啊，那么就 必须啊，等待啊，等待，所以它这里头调用了 condition wait 啊，这样的函数

那么我们看在之前，因为 在之前它就给这个加锁了，啊，加锁了，也就是用互斥量给加锁 那么为什么要先加锁，再做这件事情呢？ 因为我们如果当一个进程不能，线程不能继续执行 那么它要进到等待队列里头，条件变量里头去等待的话 如果同时，在它没有进入到，啊，条件变量队列 里头等待，那么就被切换下去 CPU，那它还没进去呢，那不就会出现问题么？ 所以在这时候，我们需要用一个 锁，来锁住，啊，这样一个后续的操作 当中所要使用的这个数据结构啊，就是队列啊，队列 因此，先要用 the mutex 来加锁 然后再去判断条件是不是成立。 条件不成立 那只有进入等待，但是注意，进入等待的时候，按照我们前面对管程的介绍 虽然它不是管程，但它的这整个的思路应该是一样的 它也在它等待的时候，它必须把 这个权力放开，也就是要把锁打开，因此这里头我们看到 condition wait 的时候，除了对条件变量操作之外，还跟上这样一个互斥量 而这个互斥量就是在 等待的时候应该把这个互斥量 打开，也就是锁打开，啊，这就是 我们看到用这个 Pthread 这个库的提供的这个机制 来解决生产者消费者问题，和我们前面所说的语言当中的管程机制 来解决生产者消费者问题所不同的地方。 它是先用一个 互斥量或者锁，先上锁，然后再去做相应的工作 但在做相应的工作，条件不成立，进入等待之前呢，那么它要把这个锁呢要打开 打开，这样的话，别的啊，线程才能继续啊，做相应的操作 同样道理我们也可以看到，这里就也是一样啊，先用 把锁加上，然后在条件不成立啊，进入等待的时候，要把锁打开啊。 这就是 这样一个区别啊，这样一个区别，我们在对这个 condition wait 这个函数呢，再进一步地，啊，阐述一下，啊 也就说，condition wait 这个函数它实际上是包含了三个主要的动作，第一个主要的动作呢 是要先解锁，啊，解锁 解锁完了之后呢，那么它会 使得这个进程等在条件变量上 啊，条件变量队列里头，然后等到它收到了一个等待的信号的时候呢 它再去做什么呀？再上锁 所以它是先解锁然后就睡眠了，然后再 上锁，啊，等到它从睡眠被唤醒以后呢 第一件事情就是上锁。 所以这里头我们可以看到，condition wait 是这样一个过程，啊，这样一个过程 我们再来看一下，啊，同样刚才这段代码 这里头我们重点看一下，while 啊，判断缓冲区是不是满了 或者是判断缓冲区是不是空的，我们是用的是 while 那么说明什么？说明我们在这里头，实现的是一个 Mesa管程，Mesa管程，也就换句话说，Pthread 线程库里头它所实现的这个条件变量 在条件变量的这个 signal 操作，啊或者是，那么它实际上是一个什么？Mesa管程的

这样一个语义，啊，管程语义 所以呢，我们这里头在一个被 啊，重新，啊，得到通知重新，啊，进入就绪的这样一个 线程，它要再次要去判断一下，啊，条件是不是成立