第三十章

1. 线程同步：

互斥量和条件变量。

1. 互斥量：

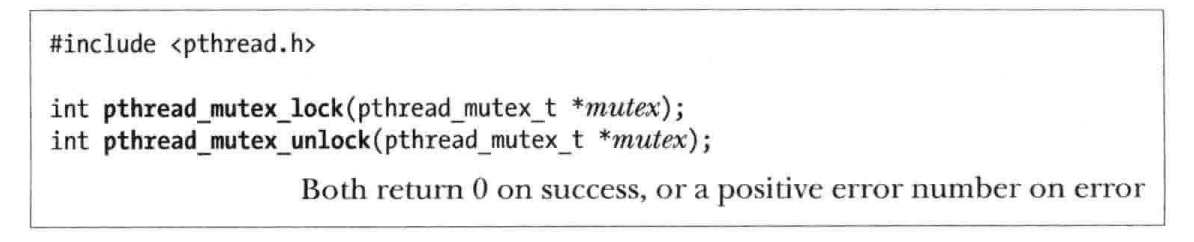
什么是临界区：指访问某一共享资源且应为原子操作的代码片段。

互斥量有两种状态：已锁定和未锁定。

1. 静态分配的互斥量：



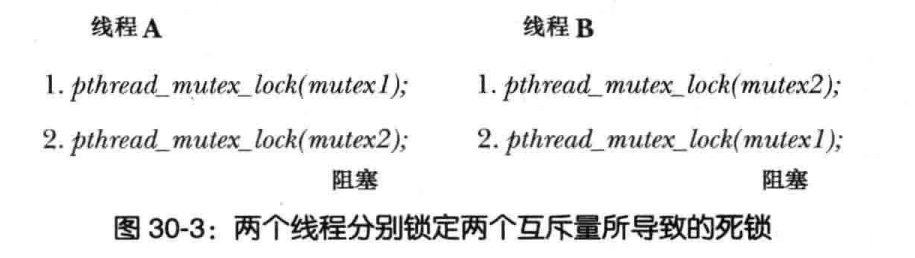
1. 加锁和解锁互斥量：



1. 互斥量的性能：

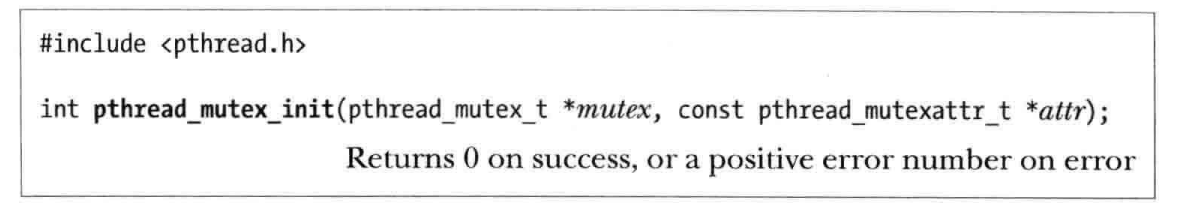
Linux上，互斥量采用futex实现，与文件锁和信号量相比，性能还行。

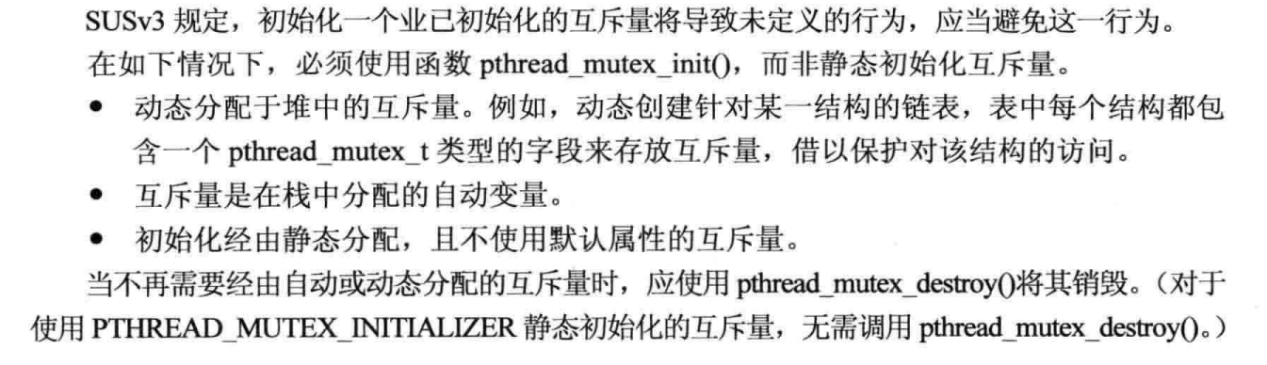
1. 互斥量的死锁：

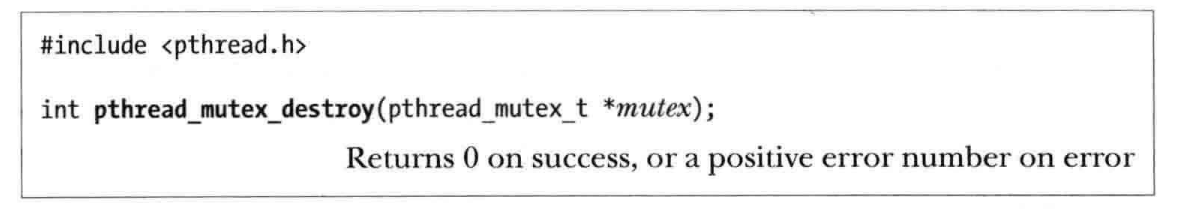


避免此类问题：可以定义互斥量的层级关系，当多个线程对一组互斥量进行操作的时候，总是以相同的顺序对该组互斥量进行锁定。另一种方法是尝试一下，使用pthread\_mutex\_trylock，如果不行，就释放所有的锁。不过这个方法不常用。

1. 动态初始化互斥量：







1. 互斥量属性：

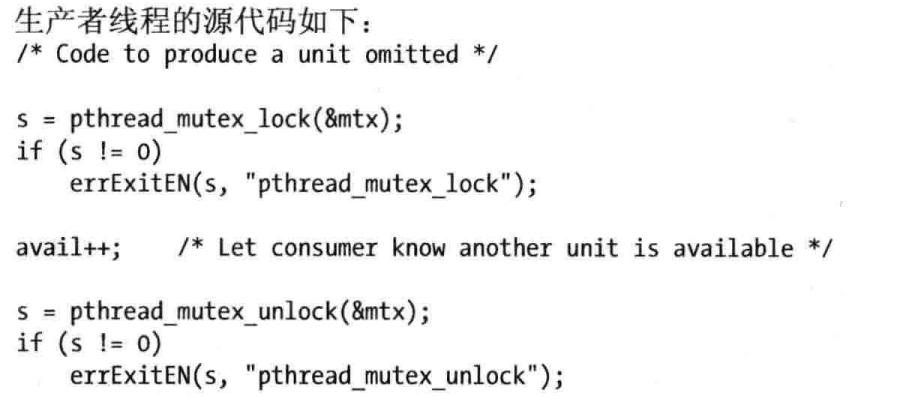
本书没有细说。

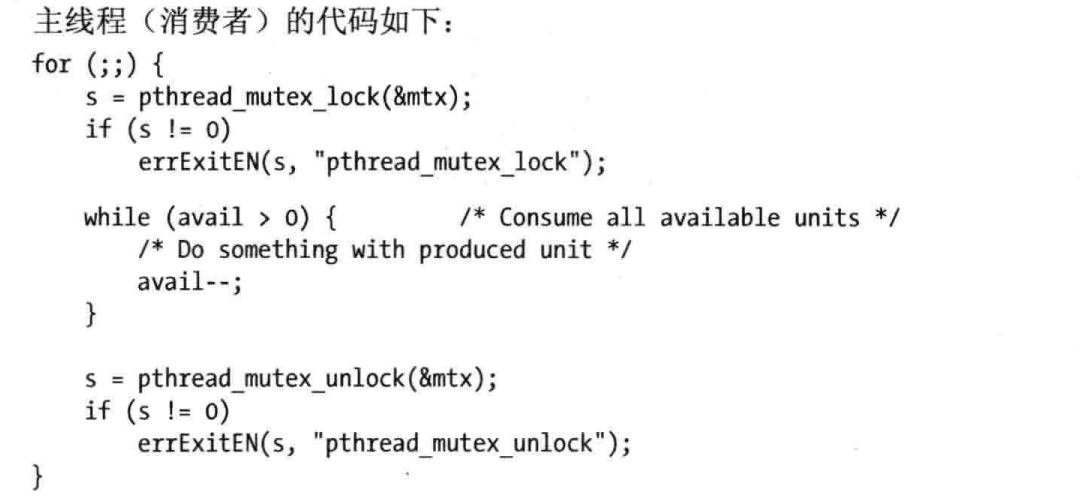
1. 互斥量的类型：

类型属于属性的一种。具体细节查看手册。

1. 通知状态的改变：条件变量

使用了一个生产者消费者的问题来说明条件变量的作用：



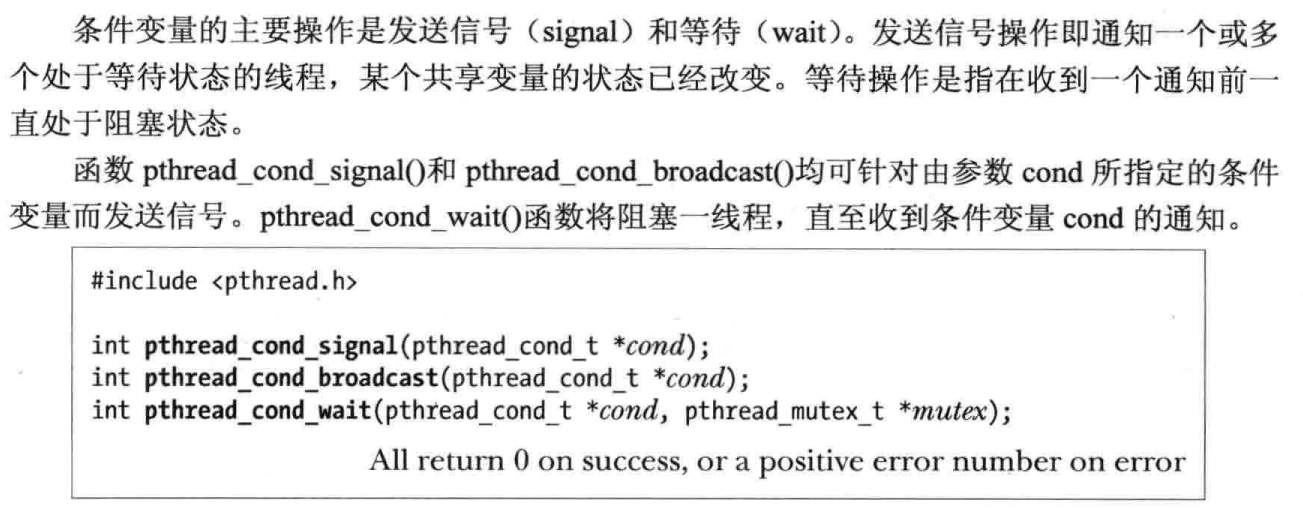


上述代码虽然可行，但是主线程会不断循环检查共享变量的状态。

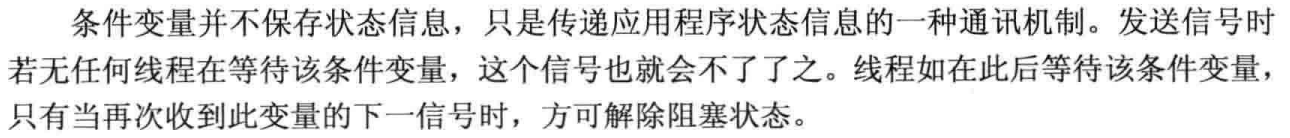
1. 由静态分配的静态变量：



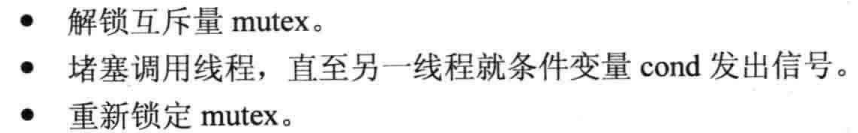
1. 等待和通知条件变量：



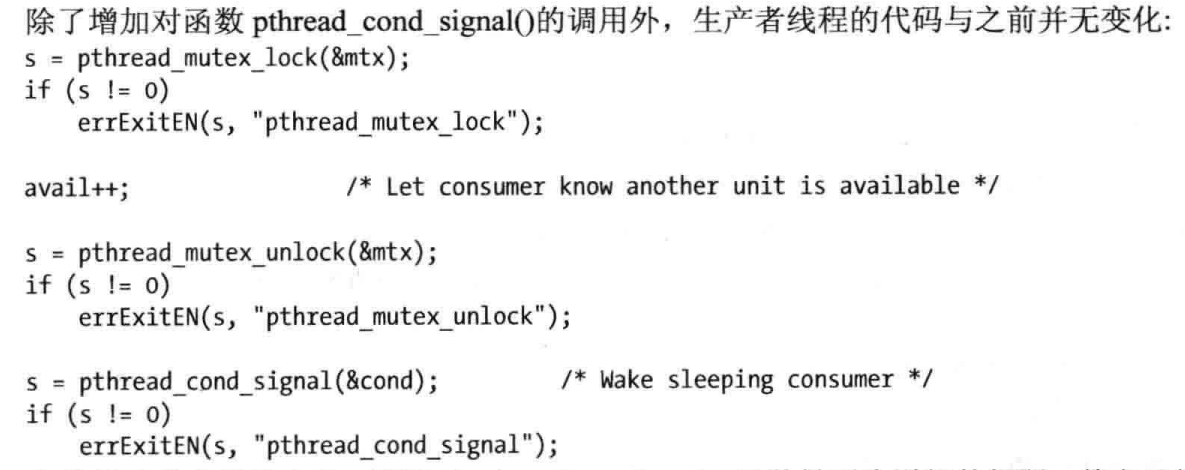
至于pthread\_cond\_signal和pthread\_cond\_broadcast之间的区别，我在Linux上试了一下，后者会唤醒所有的线程然后共同竞争互斥量，如果获得互斥量就执行，别的线程就等待互斥量的释放。



Pthread\_cond\_wait的实际工作：

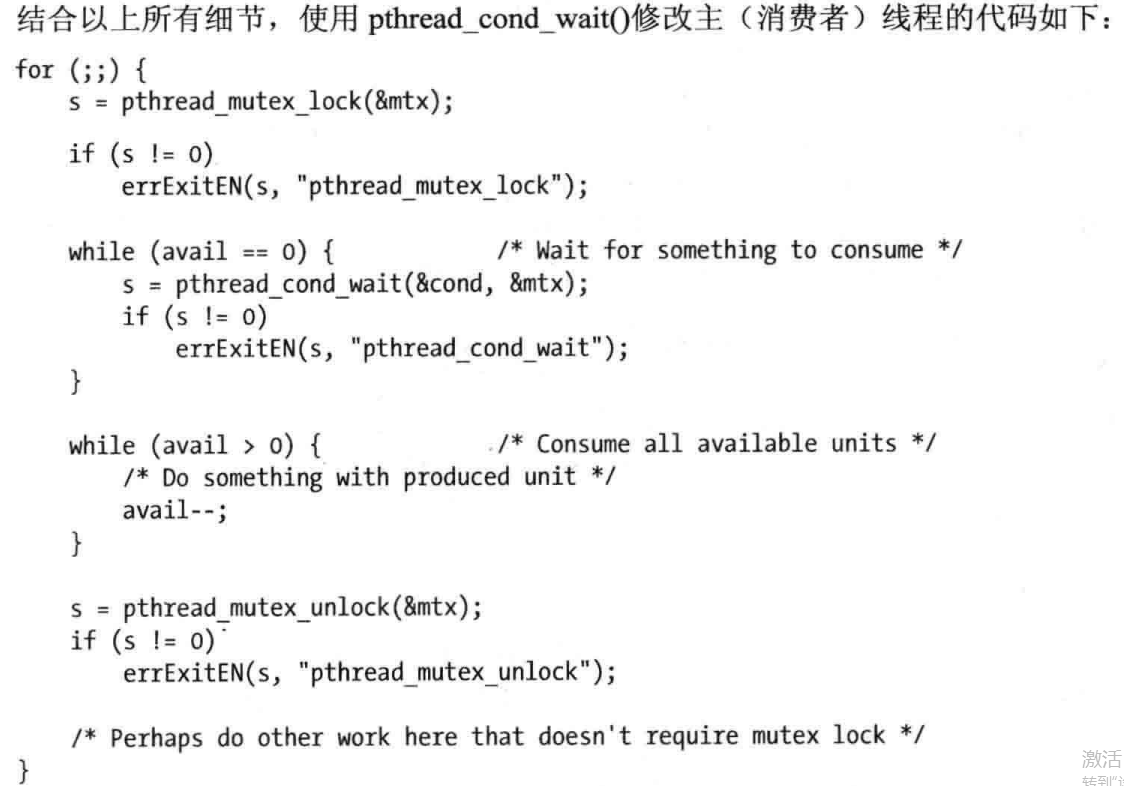


使用了条件变量的生产者和消费者代码：



上为生产者

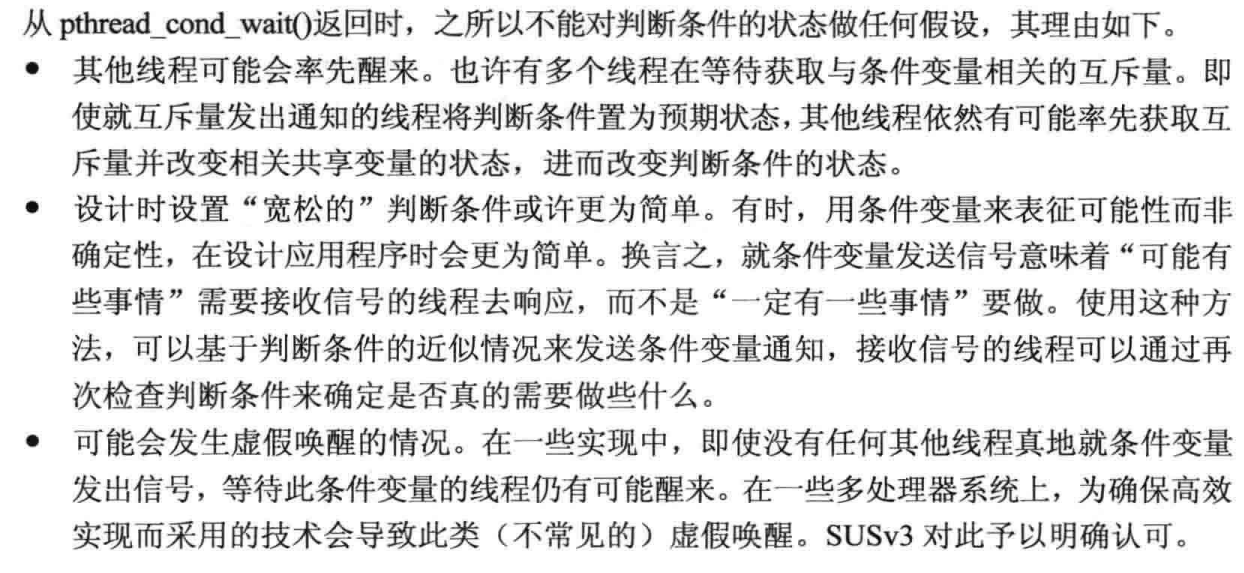
下为消费者



1. 测试条件变量的判断条件：

测试条件变量的判断条件必须是循环。

由第三点原因可以得知虚假的唤醒是内部问题。



1. 使用条件变量连接所有线程：

这是非常好的一个例子。

不过我感觉书本上的实例是有问题的。有bug的。问题在于传参给线程函数那里。

1. 动态分配的条件变量：

