**线程通信机制之互斥锁**

1. **简介**

**OK,刚刚简单的学习了我们的线程之间的通讯信号量。也就是我们所谓的P和V的操作。它是一个同步的操作，也就是A线程执行完了之后，再来执行我们的B线程。按照一定的顺序来执行。而我们的互斥锁就是用来保护我们的临界资源的，不让其他线程访问资源。例如，全局变量，A线程给全局变量加了互斥锁之后，B和C线程都无法访问全局变量中的内容了。**

**首先，我们先来了解几个概念。**

**互斥：一个线程访问资源的时候，不允许其他的线程访问。（相互排斥）**

**临界资源：同一个时候只能有一个资源，多个人访问的时候就会出问题。**

**临界资源需要加锁来保护。**

**例如:多个线程访问全局变量，里面的内容就不知道是那个线程的值。**

**OK，到了这个时候，可能很多人就会把互斥锁和信号量搞得很混淆。我们来简单的它们的特点。**

**信号量：**

1. **用于多线程中多个任务同步的机制。一个线程完成了某一个动作就通过信号量告诉别的线程，别的线程再进行某些动作（大家没有资源的时候，就阻塞在哪里）。信号量**
2. **可用于进程或者线程之间的通信。**

**互斥锁：**

**1.互斥锁是用在多线程多任务互斥的，一个线程占用了某一个资源，那么别的线程就无法访问，直到这个线程unlock，其他的线程才开始可以利用这个资源**

**2.只能用于线程之间的通信。**

**3.线程必须先获得互斥锁才能访问临界资源，访问完临界资源之后释放该锁。如果无法获得锁。线程会阻塞，直到获得锁线程为止。**

**它们区别如下图所示：**



**互斥锁的理解如下图：**



1. **互斥锁的常用操作**

**<1>定义互斥锁**

**pthread\_mutex\_t mlock;**

**<2>初始化互斥锁**

**pthread\_mutex\_init()**

**int pthread\_mutex\_init(pthread\_mutex\_t \*restrict mutex,const pthread\_mutexattr\_t \*restrict attr);**

**功能: 初始化一把锁**

**参数:**

**@mutex 要初始化的锁**

**@attr NULL (锁的属性) (NULL默认属性的锁)**

**返回值：**

**成功 返回0**

**失败 返回错误码**

**注意：我们互斥锁初始化的方法有以下两种**

1. **静态初始化[推荐使用]**

**pthread\_mutex\_t mlock;**

**mlock = PTHREAD\_MUTEX\_INITIALIZER；(通过系统定义的宏来初始化)**

1. **动态初始化**

**pthread\_mutex\_t mlock;**

**pthread\_mutex\_init(&mutex,NULL); //通过调用函数来获得。**

**以上两种方式实质差不多，任选一种方法即可。**

**<3>获得互斥锁[上锁]**

**int pthread\_mutex\_lock(pthread\_mutex\_t \*mutex);//加锁(阻塞)**

**功能：获得互斥锁，则立即返回，否则阻塞直到获得互斥锁为止。**

**参数:**

**@mutex 初始化过后的互斥锁**

**返回值：成功返回0.失败返回错误码**

**<4>释放互斥锁[解锁]**

**int pthread\_mutex\_unlock(pthread\_mutex\_t \*mutex);//解锁**

**功能：释放互斥锁对象。**

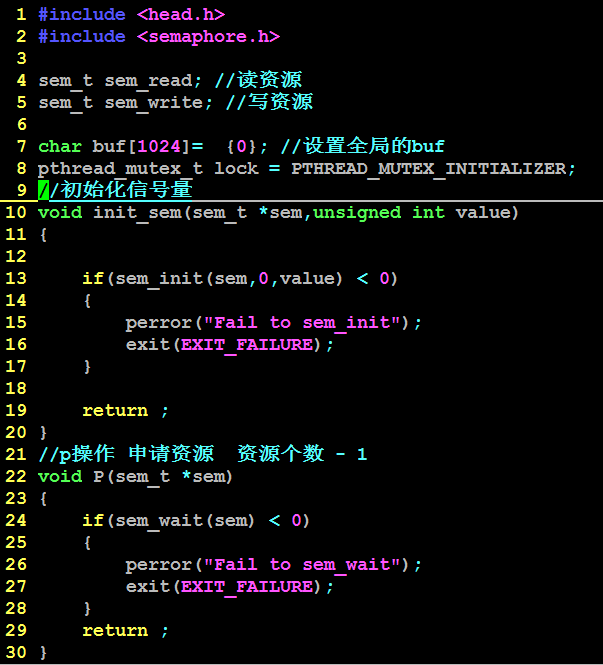
**参数:**

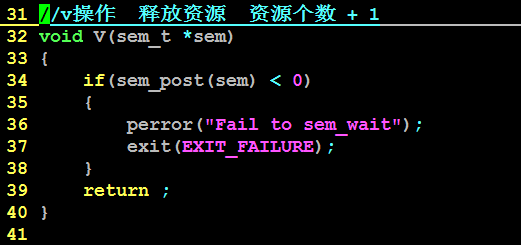
**@mutex 初始化过后的互斥锁**

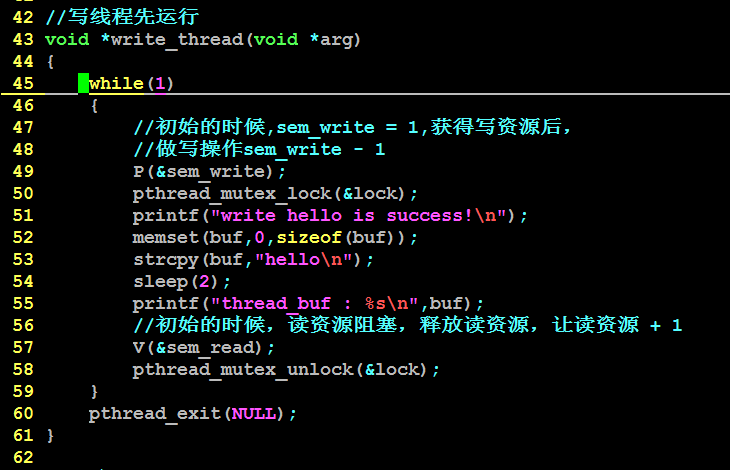
**返回值：成功返回0.失败返回错误码**

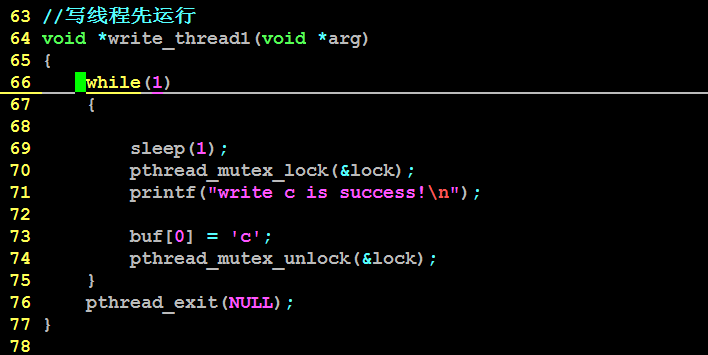
**pthread\_rw\_mutex.c**

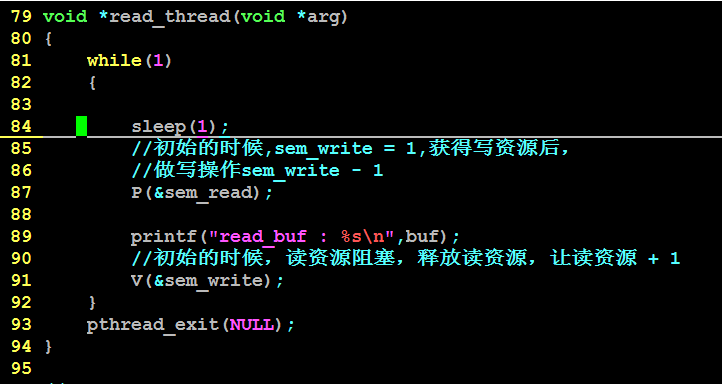
**注:以下代码若是不加互斥锁的话，我们的pthread\_buf中就有可能输出”cworld”**

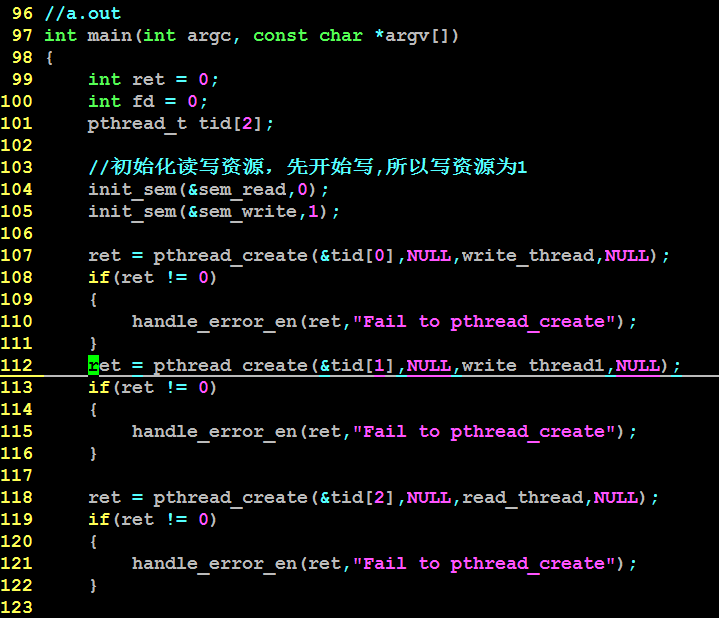


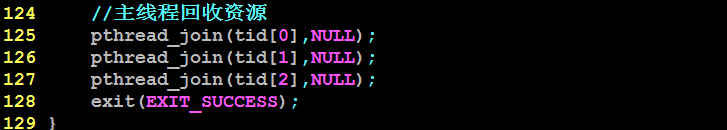




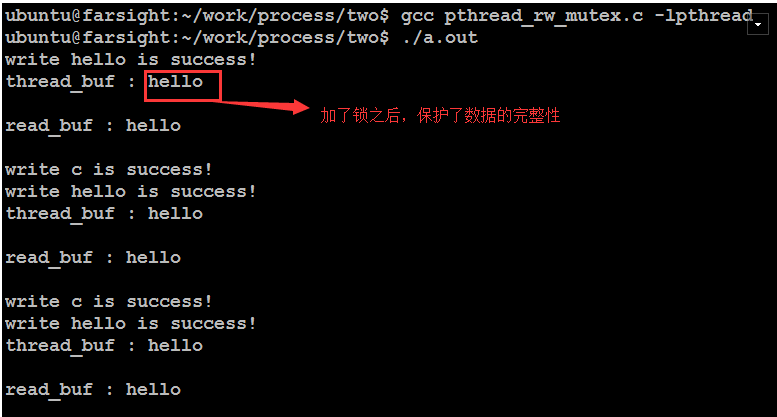








**运行结果:**



**OK，刚刚我们看到了我们的信号量和我们的互斥锁联合起来使用，相对来说这是一种比较好的方式。但是，大多数的时候。我们的互斥锁是和我们的条件变量来联合一起来使用的。我们更希望实现这样的功能，当我们读线程发现文件中没有数据的时候，就阻塞，当写线程写完数据之后就去通过读线程去进行读操作。**

**要向实现这种操作，我们就可以通过现场的条件变量来实现。**

**条件变量和信号量就像我们现实生活中的刀和剑，根据自己的需求来选择。**

1. **条件变量**

**条件变量是利用线程间共享的全局变量进行同步通信的一种机制。**

**多个线程等待某个条件满足，在条件达到时由一个线程去通知其他等待的线程。**

**常用函数：**

**<1>定义条件变量**

**Pthread\_cont\_t cond;**

**<2>初始化一个条件变量**

**int pthread\_cond\_init(pthread\_cond\_t \*restrict cond,**

**const pthread\_condattr\_t \*restrict attr);**

**初始化的方法：**

1. **静态初始化[推荐使用]**

**pthread\_cond\_t cond = PTHREAD\_COND\_INITIALIZER;**

1. **动态初始化**

**pthread\_cont\_t cond;**

**pthread\_cond\_init(&cond,NULL);**

**<3>等待条件变量**

**int pthread\_cond\_wait(pthread\_cond\_t \*cond,pthread\_mutex\_t \*mutex);**

**功能：等待别的线程唤醒它。**

**参数：**

**@cond 等待的条件变量**

**@mutex 互斥锁**

**返回值：成功返回0，失败返回错误码**

**特点：**

**(1)条件不满足的时候，阻塞调用者，阻塞前释放互斥锁**

**(2)条件满足[别人唤醒它],返回，返回的时候需要先获得互斥锁**

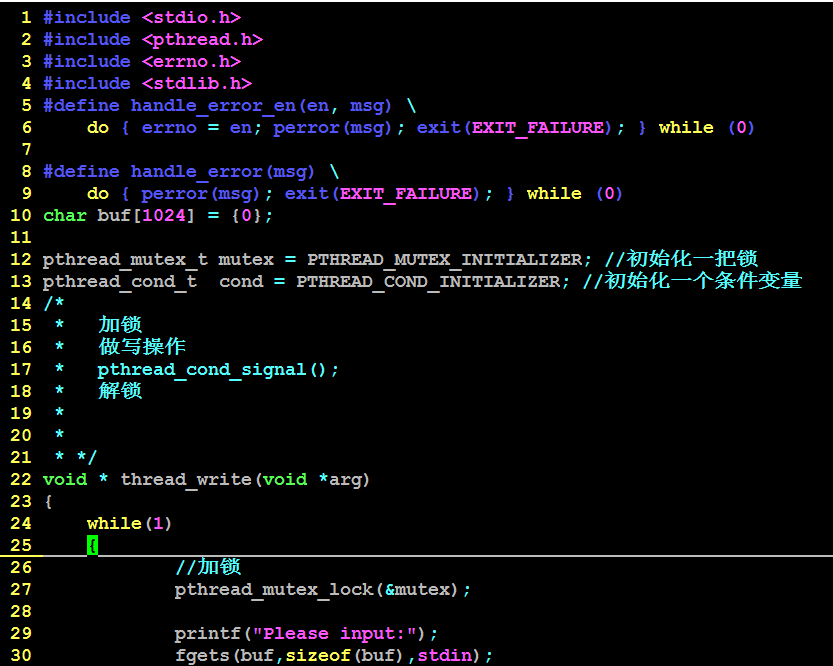
**<4>唤醒条件变量**

**int pthread\_cond\_broadcast(pthread\_cond\_t \*cond);//唤醒所有的等待线程**

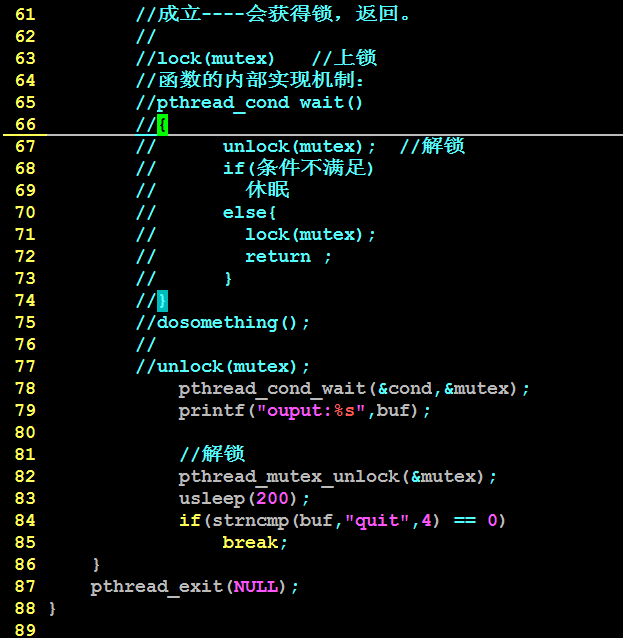
**int pthread\_cond\_signal(pthread\_cond\_t \*cond);//唤醒至少一个等待线程**

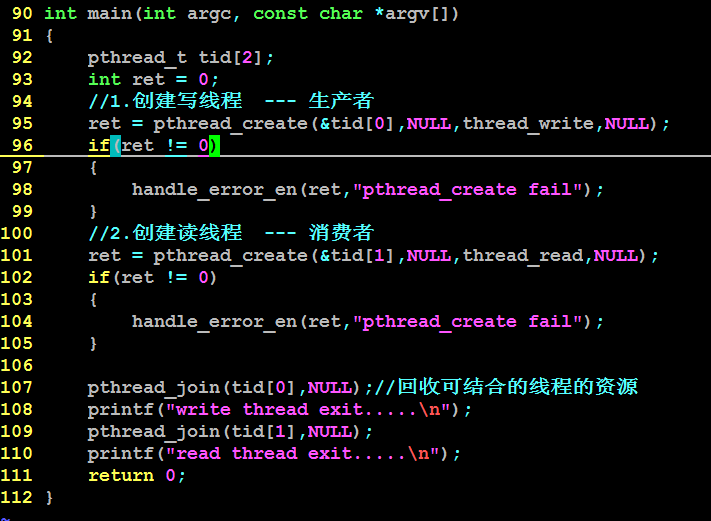
**注:被唤醒的线程一定要处于等待态。**

**pthread\_rw\_cond.c**









**运行结果：**

