**自旋锁与信号量及其区别**

为了避免并发，防止竞争。内核提供了一组同步方法来提供对共享数据的保护。 我们的重点不是介绍这些方法的详细用法，而是强调为什么使用这些方法和它们之间的差别。

Linux 使用的同步机制可以说从2.0到2.6以来不断发展完善。从最初的原子操作，到后来的信号量，从大内核锁到今天的自旋锁。这些同步机制的发展伴随 Linux从单处理器到对称多处理器的过度；伴随着从非抢占内核到抢占内核的过度。锁机制越来越有效，也越来越复杂。

目前来说内核中原子操作多用来做计数使用，其它情况最常用的是两种锁以及它们的变种:一个是自旋锁，另一个是信号量。我们下面就来着重介绍一下这两种锁机制。

**自旋锁：**

**自旋锁是专为防止多处理器并发而引入的一种锁，它在内核中大量应用于中断处理等部分(对于单处理器来说，防止中断处理中的并发可简单采用关闭中断的方式，不需要自旋锁)。  
自旋锁最多只能被一个内核任务持有，如果一个内核任务试图请求一个已被争用(已经被持有)的自旋锁，那么这个任务就会一直进行忙循环——旋转——等待锁重新可用。**要是锁未被争用，请求它的内核任务便能立刻得到它并且继续进行。自旋锁可以在任何时刻防止多于一个的内核任务同时进入临界区，因此这种锁可有效地避免多处理器上并发运行的内核任务竞争共享资源。

事实上，自旋锁的初衷就是：在短期间内进行轻量级的锁定。一个被争用的自旋锁使得请求它的线程在等待锁重新可用的期间进行自旋(特别浪费处理器时间)，所以自旋锁不应该被持有时间过长。如果需要长时间锁定的话, 最好使用信号量。  
自旋锁的基本形式如下：  
spin\_lock(&mr\_lock);  
//临界区  
spin\_unlock(&mr\_lock);

因为自旋锁在同一时刻只能被最多一个内核任务持有，所以一个时刻只有一个线程允许存在于临界区中。这点很好地满足了对称多处理机器需要的锁定服务。在单处理器上，自旋锁仅仅当作一个设置内核抢占的开关。如果内核抢占也不存在，那么自旋锁会在编译时被完全剔除出内核。

简单的说，**自旋锁在内核中主要用来防止多处理器中并发访问临界区，防止内核抢占造成的竞争。**另外自旋锁不允许任务睡眠(持有自旋锁的任务睡眠会造成自死锁——因为睡眠有可能造成持有锁的内核任务被重新调度，而再次申请自己已持有的锁)，**它能够在中断上下文中使用**。

死锁：假设有一个或多个内核任务和一个或多个资源，每个内核都在等待其中的一个资源，但所有的资源都已经被占用了。这便会发生所有内核任务都在相互等待， 但它们永远不会释放已经占有的资源，于是任何内核任务都无法获得所需要的资源，无法继续运行，这便意味着死锁发生了。自死琐是说自己占有了某个资源，然后自己又申请自己已占有的资源，显然不可能再获得该资源，因此就自缚手脚了。

**信号量：**

Linux中的信号量是一种睡眠锁。如果有一个任务试图获得一个已被持有的信号量时，信号量会将其推入等待队列，然后让其睡眠。这时处理器获得自由去执行其它代码。当持有信号量的进程将信号量释放后，在等待队列中的一个任务将被唤醒，从而便可以获得这个信号量。

信号量的睡眠特性，使得信号量适用于锁会被长时间持有的情况；只能在进程上下文中使用，因为中断上下文中是不能被调度的；另外当代码持有信号量时，不可以再持有自旋锁。信号量基本使用形式为：

static DECLARE\_MUTEX(mr\_sem);//声明互斥信号量

if(down\_interruptible(&mr\_sem))

//可被中断的睡眠，当信号来到，睡眠的任务被唤醒

//临界区

up(&mr\_sem);

**信号量和自旋锁区别：**

虽然听起来两者之间的使用条件复杂，其实在实际使用中信号量和自旋锁并不易混淆。注意以下原则:

如果代码需要睡眠——这往往是发生在和用户空间同步时——使用信号量是唯一的选择。由于不受睡眠的限制，使用信号量通常来说更加简单一些。如果需要在自旋锁和信号量中作选择，应该取决于锁被持有的时间长短。理想情况是所有的锁都应该尽可能短的被持有，但是如果锁的持有时间较长的话，使用信号量是更好的选择。另外，信号量不同于自旋锁，它不会关闭内核抢占，所以持有信号量的代码可以被抢占。这意味者信号量不会对影响调度反应时间带来负面影响。

需求                     建议的加锁方法  
低开销加锁               优先使用自旋锁  
短期锁定                 优先使用自旋锁  
长期加锁                 优先使用信号量  
中断上下文中加锁          使用自旋锁  
持有锁是需要睡眠、调度     使用信号量