 当然，还有另一种情况，就是up()操作和down()操作是交*出现的，一般的规律就是，如果进程在临界区期间又有进程（无论是哪个进程，新来的还是刚被唤醒的那个）进入睡眠，就会令count的值从0变为-1，从而进程在从临界区出来执行up()里就必须执行一次wake_up()，以确保所有的进程都能被唤醒，因为多唤醒几个是没关系的。如果进程在临界区期间没有别的进程进入睡眠，则从临界区出来执行up()时就用不着去执行wake_up()了（当然，执行了也没什么影响，不过多余罢了）。而为什么要把wake_up()和count 分开呢，可以从上面的up1看出来，例如，进程2第一次得到机会运行时，本来这时唤醒它的进程还没执行up()的，但有可能其它进程执行了up()了，所以真有可能会发现count==1的情况，这时它就真的不用睡觉了，令count=sleepers=0，就可以接着往下执行了。还可看出一点，一般的，(count ,sleepers)的值的取值范围为(n ,0)[n>0] 和(0 ,0)和 (1 ,-1)。下边看看spin_lock机制。 Spin_lock采用的方式是让一个进程运行，另外的进程忙等待，由于在只有一个cpu的机器(UP)上微观上只有一个进程在运行。所以在UP中，spin_lock和spin_unlock就都是空的了。在SMP中，spin_lock()和spin_unlock()定义如下： typedef struct {volatile unsigned int lock.} spinlock_t.static inline void spin_lock(spinlock_t *lock){__asm__ __volatile__("\n1:\t""lock . decb %0\n\t""js 2f\n" //lock->locklock==1"rep.nop\n\t""jle 2b\n\t""jmp

1b\n"".previous":"=m" (lock->lock) : : "memory").}static inline void spin_unlock(spinlock_t *lock){__asm__ __volatile__("movb $1,%0":"=m" (lock->lock) : : "memory"). //lock->lock=1} 一般是如此使用: #define SPIN_LOCK_UNLOCKED (spinlock_t) { 1 }spinlock_t xxx_lock = SPIN_LOCK_UNLOCKED.spin_lock_(amp.xxx_lock) 可以看出，它和semaphore机制解决的都是两个进程的互斥问题，都是让一个进程退出临界区后另一个进程才进入的方法，不过sempahore机制实行的是让进程暂时让出CPU，进入等待队列等待的策略，而spin_lock实行的却是却进程在原地空转，等着另一个进程结束的策略。 下边考虑中断对临界区的影响。要互斥的还有进程和中断服务程序之间。当一个进程在执行一个临界区的代码时，可能发生中断，而中断函数可能就会调用这个临界区的代码，不让它进入的话就会产生死锁。这时一个有效的方法就是关中断了。 #define local_irq_save(x) __asm__ __volatile__("pushfl . popl %0 . cli":"=g" (x): /* no input */ :"memory")#define local_irq_restore(x) __asm__ __volatile__("pushl %0 . popfl": /* no output */ :"g" (x):"memory")#define local_irq_disable() __asm__ __volatile__("cli": : :"memory")#define local_irq_enable() __asm__ __volatile__("sti": : :"memory")#define cpu_bh_disable(cpu) do { local_bh_count(cpu) . barrier(). } while (0)#define cpu_bh_enable(cpu) do { barrier(). local_bh_count(cpu)--. } while (0)#define local_bh_disable() cpu_bh_disable(smp_processor_id())#define local_bh_enable() cpu_bh_enable(smp_processor_id()) 对于UP来说，上面已经是

足够了，不过对于SMP来说，还要防止来自其它cpu的影响，这时解决的方法就可以把上面的spin_lock机制也综合进来了。 #define spin_lock_irqsave(lock, flags) do { local_irq_save(flags). spin_lock(lock). } while (0)#define spin_lock_irq(lock) do { local_irq_disable(). spin_lock(lock). } while (0)#define spin_lock_bh(lock) do { local_bh_disable(). spin_lock(lock). } while (0)#define spin_unlock_irqrestore(lock, flags) do { spin_unlock(lock). local_irq_restore(flags). } while (0)#define spin_unlock_irq(lock) do { spin_unlock(lock). local_irq_enable().} while (0)#define spin_unlock_bh(lock) do { spin_unlock(lock). local_bh_enable(). } while (0) 前面说过，对于UP来说，spin_lock（）是空的，所以以上的定义就一起适用于UP和SMP的情形了。 read_lock_irqsave(lock, flags) , read_lock_irq(lock), read_lock_bh(lock) 和write_lock_irqsave(lock, flags) , write_lock_irq(lock), write_lock_bh(lock) 就是spin_lock的一个小小的变型而已了。