内核同步之自旋锁和信号量Linux认证考试 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/644/2021_2022__E5_86_85_E6_A0_B8_E5_90_8C_E6_c103_644931.htm 1. 自旋锁 Linux内核中最常见的锁是自旋锁。一个自旋锁就是一个互斥设备，它只能有两个值："锁定"和"解锁"。如果锁可用，则"锁定"位被设置，而代码继续进入临界区；相反，如果锁被其他进程争用，则代码进入忙循环并重复检查这个锁，直到锁可用为止。这个循环就是自旋锁的"自旋"。自旋锁最多只能被一个可执行的线程持有。如果一个执行线程试图获得一个被争用的自旋锁，那么该线程就会一直进行忙循环-旋转-等待锁重新可用。注意，同一个锁可以用在多个位置。缺点：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时自旋(特别浪费处理器时间)。所以，自旋锁不应该被长时间持有。当然，可以采用另外的方式处理对锁的争用：让请求线程睡眠，直到锁重新可用时在唤醒它。但是，这里有两次明显的上下文切换，被阻塞的线程要换入或换出。因此，持有自旋锁的时间最好小于完成两次上下文切换的耗时。 注意： 1) 如果禁止内核被抢占，那么在编译时自旋锁会被完成剔除出内核。 2) Linux内核实现的自旋锁是不可递归的。小心自加锁。 3) 调试自旋锁， 加上配置选项CONFIG_DEBUG_SPINLOCK。 4）自旋锁可以使用在中断处理程序中(此处不能使用信号量，因为它们会导致睡眠)，在中断处理程序中使用自旋锁时，一定要在获取锁之前，首先禁止本地中断 (在当前处理器上的中断请求)，否则中断处理程序就会打断正持有锁的内核代码，有可能试图去争用这个已经被持有的自旋锁。 5) 加锁是对数

据不是对代码。 6) 所有自旋锁的等待在本质上都是不可中断的。 1.1. 自旋锁API介绍自旋锁实现与体系结构密切相关，定义在gt.中。在编译时对自旋锁的初始化： spinlock_t my_lock=SPIN_LOCK_UNLOCKED. 或者在运行时： void spin_lock_init(spinlock_t *lock). 进入临界区： spin_lock(amp.my_lock). 内核提供禁止中断同时请求锁的接口： spinlock_t my_lock=SPIN_LOCK_UNLOCKED. unsigned long flags. spin_lock_irqsave(amp.my_lock, flags).函数spin_lock_irqsave保存了中断的当前状态，并禁止了本地中断，然后在获取指定的锁；函数 spin_unlock_irqrestore对指定的锁解锁，然后让中断恢复到加锁前的状态。 内核提供的自旋锁的接口： void spin_lock_irq (spinlock_t *lock). void spin_unlock_irq (spinlock_t *lock). void spin_lock_bh (spinlock_t *lock). void spin_unlock_bh (spinlock_t *lock). 如果能够确保没有任何其他代码禁止本地处理器的中断，也就是说，能够确保在释放自旋锁时应该启用中断，这可以使用spin_lock_irq函数，而无需跟踪标志。函数spin_lock_bh在获得锁之前禁止软件中断，但是会让硬件中断保持打开。 