Linux系统内核的同步机制－自旋锁（1） PDF转换可能丢失图片或格式，建议阅读原文
https://www.100test.com/kao_ti2020/144/2021_2022_Linux_E7_B3_BB_E7_BB_c103_144639.htm 自旋锁最多只能被一个可执行线程持有。自旋锁不会引起调用者睡眠，如果一个执行线程试图获得一个已经被持有的自旋锁，那么线程就会一直进行忙循环，一直等待下去，在那里看是否该自旋锁的保持者已经释放了锁，"自旋"一词就是因此而得名。 由于自旋锁使用者一般保持锁时间非常短，因此选择自旋而不是睡眠是非常必要的，自旋锁的效率远高于互斥锁。 信号量和读写信号量适合于保持时间较长的情况，它们会导致调用者睡眠，因此只能在进程上下文使用（_trylock的变种能够在中断上下文使用）；而自旋锁适合于保持时间非常短的情况，因为一个被争用的自旋锁使得请求它的线程在等待重新可用时自旋，特别浪费处理时间，这是自旋锁的要害之处，所以自旋锁不应该被长时间持有。在实际应用中自旋锁代码只有几行，而持有自旋锁的时间也一般不会超过两次上下方切换，因线程一旦要进行切换，就至少花费切出切入两次，自旋锁的占用时间如果远远长于两次上下文切换，我们就可以让线程睡眠，这就失去了设计自旋锁的意义。 如果被保护的共享资源只在进程上下文访问，使用信号量保护该共享资源非常合适，如果对共享资源的访问时间非常短，自旋锁也可以。但是如果被保护的共享资源需要在中断上下文访问（包括底半部即中断处理句柄和顶半部即软中断），就必须使用自旋锁。 自旋锁保持期间是抢占失效的，而信号量和读写信号量保持期间是可以被抢占的。自旋锁只有在内核可抢占或SMP的情况下才

真正需要，在单CPU且不可抢占的内核下，自旋锁的所有操作都是空操作。 一个执行单元要想访问被自旋锁保护的共享资源，必须先得到锁，在访问完共享资源后，必须释放锁。如果在获取自旋锁时，没有任何执行单元保持该锁，那么将立即得到锁；如果在获取自旋锁时锁已经有保持者，那么获取锁操作将自旋在那里，直到该自旋锁的保持者释放了锁。无论是互斥锁，还是自旋锁，在任何时刻，最多只能有一个保持者，也就说，在任何时刻最多只能有一个执行单元获得锁。自旋锁的实现和体系结构密切相关，代码一般通过汇编实现，定义在文件,实际用到的接口定义在文件夹 中, 自旋锁的API有： CODE:spin_lock_init(x)该宏用于初始化自旋锁x。自旋锁在真正使用前必须先初始化。该宏用于动态初始化指定的。 CODE:DEFINE_SPINLOCK(x)该宏声明一个自旋锁x并初始化它。该宏在2.6.11中第一次被定义，在先前的内核中并没有该宏。 CODE:SPIN_LOCK_UNLOCKED该宏用于静态初始化一个自旋锁。 CODE:DEFINE_SPINLOCK(x)等同于spinlock_t x = SPIN_LOCK_UNLOCKEDspin_is_locked(x) )该宏用于判断自旋锁x是否已经被某执行单元保持（即被锁），如果是，返回真，否则返回假。 CODE:spin_unlock_wait(x)该宏用于等待自旋锁x变得没有被任何执行单元保持，如果没有任何执行单元保持该自旋锁，该宏立即返回，否则将循环在那里，直到该自旋锁被保持者释放。 CODE:spin_trylock(lock)该宏尽力获得自旋锁lock，如果能立即获得锁，它获得锁并返回真，否则不能立即获得锁，立即返回假。它不会自旋等待lock被释放。 CODE:spin_lock(lock)该宏用于获得自旋锁lock，如果能够立即获得锁，它就马上返回，否则，它将

自旋在那里，直到该自旋锁的保持者释放，这时，它获得锁并返回。总之，只有它获得锁才返回。
CODE:spin_lock_irqsave(lock, flags)该宏获得自旋锁的同时把标志寄存器的值保存到变量flags中并失效本地中断。
CODE:spin_lock_irq(lock) 