**[semaphore 实现机制 [转]](http://www.cnblogs.com/napoleon_liu/archive/2010/01/07/1641422.html)**

相关数据结构   
首先看看信号量的相关数据结构:   
<include/linux/semaphore.h>   
struct semaphore {   
spinlock\_t lock; #lock应该是这个信号量的自旋锁   
unsigned int count; #count表示的是这个信号量的计数器   
struct list\_head wait\_list; #wait\_list顾名思义应该是等待链表了   
};   
信号量的数据结构就这么简单.那么信号量的初始化是怎么做的呢?还有大家熟悉的信号量P,V操作是怎么实现的呢?   
----------------------------------------------------------------------------------------------------------------   
信号量的初始化:   
#define DECLARE\_MUTEX(name) \   
struct semaphore name = \_\_SEMAPHORE\_INITIALIZER(name, 1)   
#define \_\_SEMAPHORE\_INITIALIZER(name, n) \   
{ \   
.lock = \_\_SPIN\_LOCK\_UNLOCKED((name).lock), \ #初始化自旋锁   
.count = n, \ #将信号量计数器赋值为n   
.wait\_list = LIST\_HEAD\_INIT((name).wait\_list), \ #初始化等待队列   
}   
也可以用以下方法来初始化信号量:   
static inline void sema\_init(struct semaphore \*sem, int val)   
{   
static struct lock\_class\_key \_\_key;   
\*sem = (struct semaphore) \_\_SEMAPHORE\_INITIALIZER(\*sem, val);   
lockdep\_init\_map(&sem->lock.dep\_map, "semaphore->lock", &\_\_key, 0);   
}   
#define init\_MUTEX(sem) sema\_init(sem, 1)   
#define init\_MUTEX\_LOCKED(sem) sema\_init(sem, 0)   
-------------------------------------------------------------------------------------------------------------------   
PV操作:   
首先看看P操作   
<kernel/semaphore.c>   
void up(struct semaphore \*sem)   
{   
unsigned long flags;   
spin\_lock\_irqsave(&sem->lock, flags);   
if (likely(list\_empty(&sem->wait\_list)))#如果等待链表为空,表示没有正在等待此信号量的进程,count++就行   
sem->count++;   
else   
\_\_up(sem);   
spin\_unlock\_irqrestore(&sem->lock, flags);   
}   
如果等待链表为空,表示没有正在等待此信号量的进程,count++就行   
如果等待链表不为空,也就是还有进程在等待此信号量,那么此时进入到\_\_up()这个函数中:   
<kernel/semaphore.c>   
static noinline void \_\_sched \_\_up(struct semaphore \*sem)   
{   
#取出等待链表中的最前面的那个进程   
struct semaphore\_waiter \*waiter = list\_first\_entry(&sem->wait\_list,   
struct semaphore\_waiter, list);   
list\_del(&waiter->list); #将这个进程从等待链表中删除   
waiter->up = 1;   
wake\_up\_process(waiter->task); #唤醒这个进程   
}   
此函数从等待列表中移出最前面的那个进程,然后唤醒它   
<kernel/semaphore.c>   
struct semaphore\_waiter {   
struct list\_head list; #链表项   
struct task\_struct \*task; #进程结构,也可以叫做进程PCB   
int up;   
};   
这个结构主要就是用来辅助把进程放到信号量的等待队列,后面的V操作会再介绍它   
----------------------------------------------------------------------------------------------------------------   
V操作:   
<kernel/semaphore.c>   
void down(struct semaphore \*sem)   
{   
unsigned long flags;   
spin\_lock\_irqsave(&sem->lock, flags); #要对sem进行操作了,加锁   
if (likely(sem->count > 0)) #如果count>0,直接count--就行了   
sem->count--;   
else   
\_\_down(sem); #调用\_\_down()   
spin\_unlock\_irqrestore(&sem->lock, flags); #对sem操作完毕,释放锁   
}   
如果count是<=0的,那么调用\_\_down()来把进程放进等待队列里,现在看看\_\_down()是怎么实现的   
<kernel/semaphore.c>   
static noinline void \_\_sched \_\_down(struct semaphore \*sem)   
{   
\_\_down\_common(sem, TASK\_UNINTERRUPTIBLE, MAX\_SCHEDULE\_TIMEOUT);   
}   
static inline int \_\_sched \_\_down\_common(struct semaphore \*sem, long state, long timeout)   
{   
struct task\_struct \*task = current; #得到当前正在运行的进程   
struct semaphore\_waiter waiter;   
list\_add\_tail(&waiter.list, &sem->wait\_list); #将waiter加到链表sem->wait\_list后面   
waiter.task = task; #waiter的task设置成正在运行的这个进程   
waiter.up = 0;   
for (;;) {   
if (signal\_pending\_state(state, task)) #如果进程被一个意外的信号中断,直接放弃等待(从等待链表中删除),返回-EINTR错误号   
goto interrupted;   
if (timeout <= 0) #如果等待时间到,从等待链表中删除之,返回-ETIME错误号   
goto timed\_out;   
\_\_set\_task\_state(task, state); #设置进程状态为TASK\_UNINTERRUPTIBLE   
spin\_unlock\_irq(&sem->lock);   
timeout = schedule\_timeout(timeout); #[调度](http://cms.hit.edu.cn/mod/quiz/view.php?id=2010)  
spin\_lock\_irq(&sem->lock);   
if (waiter.up)   
return 0;   
}   
timed\_out:   
list\_del(&waiter.list);   
return -ETIME;   
interrupted:   
list\_del(&waiter.list);   
return -EINTR;   
}   
该函数首先申请一个semaphore\_waiter结构,   
<kernel/semaphore.c>   
struct semaphore\_waiter {   
struct list\_head list;   
struct task\_struct \*task;   
int up;   
};   
然后将这个结构加入到sem信号量的等待链表sem->wait\_list中,   
接着将当前进程task填入waiter.task中,   
然后在一个循环中进行进程调度schedule\_timeout,每进行一次[调度](http://cms.hit.edu.cn/mod/quiz/view.php?id=2010),timeout会减小   
如果当前进程task的状态位就绪,则跳转到interrupt,从信号量列表sem->wait\_list中删除这个任务对应的节点   
如果遇到意外的信号或者等待时间到,从信号量列表sem->wait\_list中删除这个任务对应的节点,返回相应的错误号