sleep\_on()用于进程睡眠。

其原型为

void sleep\_on(struct task\_struct \*\*p);

比如某个资源是互斥的，当资源被某一个进程占用时，其他进程便无法访问此资源。

假设资源结构为

struct \_\_xxx\_resource

{

       struct resource \*rs;

       struct task\_struct \*wait;

       ....

}xxx\_resource;

那么若某进程无法得到资源，其使用

sleep\_on(&wait) 在资源的wait队列上睡眠，注意到这里的wait只是指针变量，那么多个进程调用sleep\_on()是如何形成所谓的wait队列呢？以下是我的一些理解。

sleep\_on()函数原型如下

void sleep\_on(struct task\_struct \*\*p)

{

   struct task\_struct \*tmp;

   if (!p)

      return;

   if (current == &(init\_task.task))

      panic("task[0] trying to sleep");

   tmp = \*p;

   \*p = current;

   current->state = TASK\_UNINTERRUPTIBLE;

   schedule();

   if (tmp)

      tmp->state=0;

}

函数流程比较简单，先做了一些基本的判断，然后使用一个tmp变量指向\*p，\*p指向当前的进程，设置进程标志为TASK\_UNINTERRUPTIBLE即不可中断睡眠，随后就执行schedule()调度其他进程运行，当本进程恢复运行时，设置tmp->state为0即TASK\_RUNNING。

让我们用例子来理解sleep\_on()

假设有2个进程A,B都无法访问资源，只能执行sleep\_on(&wait)进行睡眠。

1. 进程A执行sleep\_on(&wait)，进入睡眠，tmp -> wait'(NULL)，wait'' -> 进程A task\_struct

2. 进程B执行sleep\_on(&wait)，进入睡眠，tmp -> wait''，wait''' -> 进程B task\_struct

wait上标表示wait值的不同。

可以看到wait始终指向的是最后一个执行sleep\_on()的进程task\_struct。

理解的关键在于：所有的进程共用一个公有的wait变量(因为资源只有一个)，然而内部的tmp变量是进程私有的，(tmp变量在栈中分配)，便是私有的tmp和公有的wait形成了等待队列，而wait可以看成是等待队列的第一个成员，tmp则指向等待队列的第二个成员。

当资源被释放，使用wake\_up(&wait)唤醒等待队列上的睡眠进程，过程则如下

1. wait指向进程B的task\_struct，则进程B被唤醒。

2. 进程B唤醒后，执行if (tmp) tmp->state = 0，进程B上下文中的tmp是指向进程A的task\_struct，则进程A被唤醒。

3. 进程A被进程B唤醒。

可以发现，调用了wake\_up(&wait)之后，在等待队列上睡眠的所有进程都被唤醒了，以一种“链式”的方式。

---------------------

作者：wu5795175

来源：CSDN

原文：https://blog.csdn.net/wu5795175/article/details/9156535

版权声明：本文为博主原创文章，转载请附上博文链接！