**由进程睡眠函数sleep\_on()中的堆栈变量tmp引发的思考 关于进程内核堆栈**

置顶 2014年07月22日 14:29:48 [i为伱而搁浅](https://me.csdn.net/u014292052) 阅读数：798

**sleep\_on()**

**[cpp]**[**view plaincopy**](http://blog.csdn.net/geekcome/article/details/6393701)

1. **/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**
2. **/\* 功能：当前进程进入不可中断睡眠态，挂起在等待队列上                        \*/**
3. **/\* 参数：p 等待队列头                                                       \*/**
4. **/\* 返回：（无）                                                               \*/**
5. **/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**
6. **void sleep\_on(struct task\_struct \*\*p)**
7. **{**
8. **struct task\_struct \*tmp;        // tmp用来指向等待队列上的下一个进程**
9. **if (!p)         // 无效指针，退出**
10. **return;**
11. **if (current == &(init\_task.task))   // 进程0不能睡眠**
12. **panic("task[0] trying to sleep");**
13. **tmp = \*p;           // 下面两句把当前进程放到等待队列头，等待队列是以堆栈方式**
14. **\*p = current;       //  管理的。后到的进程等在前面**
15. **current->state = TASK\_UNINTERRUPTIBLE;   // 进程进入不可中断睡眠状态**
16. **schedule();     // 进程放弃CPU使用权，重新调度进程**
17. **// 当前进程被wake\_up()唤醒后，从这里开始运行。**
18. **// 既然等待的资源可以用了，就应该唤醒等待队列上的所有进程，让它们再次争夺**
19. **// 资源的使用权。这里让队列里的下一个进程也进入运行态。这样当这个进程运行**
20. **// 时，它又会唤醒下下个进程。最终唤醒所有进程。**
21. **if (tmp)**
22. **tmp->state=0;**
23. **}**
24. **关于此函数中的tmp临时指针变量，当通过它隐式构建出一条等待队列时，由于是多个不同的进程调用此函数从而睡眠，所以每个进程中堆栈中必然保存着各自的tmp变量，无疑，tmp是保存在进程的内核堆栈空间中的。**
25. **自然想到赵炯博士内核注释中的一段话:"每当任务执行内核程序而需要使用其内核栈时，CPU就会利用TSS结构把它的内核态堆栈设置成由这两个值构成（即TSS中的tss.ss0和tss.esp0，指明了内核堆栈，其值是静态的，初始化后不可变）。在任务切换时，老任务的内核堆栈指针不会被保存。对CPU来讲，这两个值是只读的。因此每当一个任务进入内核态执行时，其内核态堆栈总是空的。"**
26. **联想到这段话，突然觉得，既然"在任务切换时，老任务的内核堆栈指针不会被保存、其内核态堆栈总是空的"，那么进程睡眠之后，再被唤醒从而重新执行的时候，由于上次睡眠时切换到了别的进程，所以该进程上次的内核堆栈指针没有被保存下来，所以此时被唤醒后，内核堆栈应该是空的啊，那么tmp变量不是不复存在了吗？**
27. **经过一番思索，查资料，反复翻看内核注释相关内容，最后终于感觉明白了。在任务切换时，老任务的内核堆栈指针的确不会被保存，但一定也保存了内核堆栈指针。书上的这句话我理解错了，其实，对于一个正在运行中的进程来说，其堆栈指针寄存器只有一对ss和esp，只不过是在用户态其指向的是用户态堆栈，在内核态指向的是内核堆栈。实际上并不存在用于指向进程内核堆栈的专用寄存器组。系统是靠在tss中保存不同级别的静态堆栈指针ss和esp，从而在进程运行级别发生变化时，动态的装入进程的ss和esp寄存器，达到切换进程堆栈的目的。进程的内核态堆栈指针保存在了进程tss结构中的tss.ss0和tss.esp0中，值是静态的，即初始化后便不可再改变，所以书上说:"在任务切换时，老任务的内核堆栈指针不会被保存",这句话应当是针对tss.ss0和tss.esp0的。而"每当一个任务进入内核态执行时，其内核态堆栈总是空的",这句话应当是说每当进程运行级别发生变化时，才会动态改变进程ss和esp的值，具体而言就是进程执行系统调用或者中断时，进程从用户级3转到系统级0时，由于进程运行级别发生变化，导致进程堆栈切换，切换到哪个堆栈呢？由进程tss中相应的值确定。由于这个值是确定的，不变的，因此每次进程切换堆栈时，堆栈都是空的。**
28. **但是，tmp的值的确在进程切换时被保存了，保存在了进程tss.ss(段选择符)和tss.esp(通用寄存器)中，即进程切换时，当前进程堆栈指针值保存在了tss.ss和tss.esp中，进程切换只发生在内核态，所以每次进程切换时，保存的都是进程的内核态堆栈，但是，用户态堆栈呢？保存在了进程内核堆栈空间中，在进程进入内核态执行时，进程的用户态堆栈指针已经压入内核栈了。所以，这样来，进程的所有堆栈都保存了下来。当进程再次执行时，CPU会将进程tss中的ss和esp值装入CPU的ss和esp寄存器中，从而恢复进程原状态。**
29. **PS:赵炯博士的话:"在任务切换时，老任务的内核堆栈指针不会被保存。对CPU来讲，这两个值是只读的。因此每当一个任务进入内核态执行时，其内核态堆栈总是空的",我觉得这样说更好:"在任务切换时，老任务的TSS结构中保存的tss.ss0和tss.esp0不会被更新。对CPU来讲，这两个值是只读的。因此每当一个任务（从用户态）进入内核态执行时，其内核态堆栈总是空的。""。**
30. **从这里也可以看出，进程每次切换保存的都是进程内核堆栈指针，因为进程切换只发生在进程在内核空间执行时，也同样因为这个原因，进程再度执行时，恢复的都是内核堆栈指针，都是先恢复到内核态中继续运行的，之后再返回到用户空间运行。**