linux 3.6 启动源码分析(四) rest_init

原创 2013年12月16日 14:25:13 1310

在内核初始化函数start_kernel执行到最后，就是调用rest_init函数，这个函数的主要使命就是创建并启动内核线程init。这个函数虽然意思为剩下的初始化，但是这个“剩下”的可是内容颇多，下面详细分析如下：

```cpp
static noinline void __init_refok rest_init(void)
{
    int pid;
    rcu_scheduler_starting();// 1.内核RCU锁机制调度启动,因为下面就要用到
    /*
     * We need to spawn init first so that it obtains pid 1, however
     * the init task will end up wanting to create kthreads, which, if
     * we schedule it before we create kthreadd, will OOPS.
     我们必须先创建init内核线程，这样它就可以获得pid为1。
     尽管如此init线程将会挂起来等待创建kthreads线程。
     如果我们在创建kthreadd线程前调度它，就将会出现OOPS。
     */
    kernel_thread(kernel_init, NULL, CLONE_FS | CLONE_SIGHAND);
    numa_default_policy();//    1.设定NUMA系统的内存访问策略为默认
    pid = kernel_thread(kthread, NULL, CLONE_FS | CLONE_FILES);
    /*
    1.创建kthreadd内核线程，它的作用是管理和调度其它内核线程。
    2.它循环运行一个叫做kthreadd的函数，该函数的作用是运行kthread_create_list全局链表中维护的内核线程。
    3.调用kthread_create创建一个kthread，它会被加入到kthread_create_list 链表中；
    4.被执行过的kthread会从kthread_create_list链表中删除；
    5.且kthreadd会不断调用scheduler函数让出CPU。此线程不可关闭。

    上面两个线程就是我们平时在Linux系统中用ps命令看到：
    $ ps -A
    PID TTY TIME CMD
    3.1 ? 00:00:00 init
    4.2 ? 00:00:00 kthreadd
    */
    rcu_read_lock();
    kthreadd_task = find_task_by_pid_ns(pid, &init_pid_ns);
    rcu_read_unlock();
    complete(&kthreadd_done);

    /*1.获取kthreadd的线程信息，获取完成说明kthreadd已经创建成功。并通过一个
    complete变量（kthreadd_done）来通知kernel_init线程。*/
    /*
     * The boot idle thread must execute schedule()
     * at least once to get things moving:
     */
    init_idle_bootup_task(current);
    schedule_preempt_disabled();
    /* Call into cpu_idle with preempt disabled */
    cpu_idle();
}
```

在以上的函数中，内核创建了两个内核线程，一个是内核线程的管理者，另一个是内核初始化线程init，后者是我们分析内核启动需要关注的，这个线程继续做系统的初始化（其中就包含了设备驱动系统）

qing_ping 关注

原创	粉丝	喜欢	评论
30	31	1	1

等级：博客 3 访问量：3万+

积分：650 排名：7万+