start_kernel()的最后是rest_init()

```
asmlinkage __visible void __init start_kernel(void)
{

......

/* Do the rest non-__init'ed, we're now alive */
        rest_init();
}
```

```
static __initdata DECLARE_COMPLETION(kthreadd_done);

static noinline void __init_refok rest_init(void)
{
int pid;

        rcu_scheduler_starting();
/*
         * 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.
         */
        kernel_thread(kernel_init, NULL, CLONE_FS);     （0）
        numa_default_policy();                                                  (0.01)
        pid = kernel_thread(kthreadd, NULL, CLONE_FS | CLONE_FILES);
        rcu_read_lock();
        kthreadd_task = find_task_by_pid_ns(pid, &init_pid_ns);
        rcu_read_unlock();
        complete(&kthreadd_done);         （1）

/*
         * The boot idle thread must execute schedule()
         * at least once to get things moving:
         */
        init_idle_bootup_task(current);
        schedule_preemptstart_kernel()的最后是rest_init()
asmlinkage __visible void __init start_kernel(void)
{

        ......

        /* Do the rest non-__init'ed, we're now alive */
        rest_init();
}_disabled();
```

```
27.   /* Call into cpu_idle with preempt disabled */
28.           cpu_startup_entry(CPUHP_ONLINE);
29.   }
```

static int __ref kernel_init(void *unused)和

int kthreadd(void *unused)

分别在独立的thread中运行，但两者间还是有同步关系的。

```
1.  static int __ref kernel_init(void *unused)
2.  {
3.          int ret;
4.
5.          kernel_init_freeable();            (0.1)
6.          /* need to finish all async __init code before freeing the memory */
7.          async_synchronize_full();
8.          free_initmem();
9.          mark_rodata_ro();
10.         system_state = SYSTEM_RUNNING;
11.         numa_default_policy();
12.
13.         flush_delayed_fput();
14.
15.         if (ramdisk_execute_command) {
16.                 ret = run_init_process(ramdisk_execute_command);
17.                 if (!ret)
18.                         return 0;
19.                 pr_err("Failed to execute %s (error %d)\n",
20.                        ramdisk_execute_command, ret);
21.         }
22.
23.         /*
24.          * We try each of these until one succeeds.
25.          *
26.          * The Bourne shell can be used instead of init if we are
27.          * trying to recover a really broken machine.
28.          */
29. if (execute_command) {                          (2)
30.                 ret = run_init_process(execute_command);
31.                 if (!ret)
```

```
32.                         return 0;
33.                 pr_err("Failed to execute %s (error %d).  Attempting defaults...\
    n",
34.                         execute_command, ret);
35.         }
36.         if (!try_to_run_init_process("/sbin/init") ||
37.             !try_to_run_init_process("/etc/init") ||
38.             !try_to_run_init_process("/bin/init") ||
39.             !try_to_run_init_process("/bin/sh"))
40.                 return 0;
41.
42.         panic("No working init found.  Try passing init= option to kernel. "
43.               "See Linux Documentation/init.txt for guidance.");
44. }
```

```
1.  static noinline void __init kernel_init_freeable(void)
2.  {
3.          /*
4.           * Wait until kthreadd is all set-up.
5.           */
6.          wait_for_completion(&kthreadd_done);        (0.2)
7.
8.          /* Now the scheduler is fully set up and can do blocking allocations */
9.          gfp_allowed_mask = __GFP_BITS_MASK;         (1.1)
10.
11.         /*
12.          * init can allocate pages on any node
13.          */
14.         set_mems_allowed(node_states[N_MEMORY]);
15.         /*
16.          * init can run on any cpu.
17.          */
18.         set_cpus_allowed_ptr(current, cpu_all_mask);
19.
20.         cad_pid = task_pid(current);
21.
22.         smp_prepare_cpus(setup_max_cpus);
23.
24.         do_pre_smp_initcalls();
25.         lockup_detector_init();
26.
27.         smp_init();
28.         sched_init_smp();
29.
30.         do_basic_setup();                           (1.2)
31.
```

```
32.          /* Open the /dev/console on the rootfs, this should never fail */
33.          if (sys_open((const char __user *) "/dev/console", O_RDWR, 0) < 0)
34.                  pr_err("Warning: unable to open an initial console.\n");
35.
36.          (void) sys_dup(0);
37.          (void) sys_dup(0);
38.          /*
39.           * check if there is an early userspace init.  If yes, let it do all
40.           * the work
41.           */
42.
43.          if (!ramdisk_execute_command)
44.                  ramdisk_execute_command = "/init";
45.
46.          if (sys_access((const char __user *) ramdisk_execute_command, 0) != 0) {
47.                  ramdisk_execute_command = NULL;
48.                  prepare_namespace();
49.          }
50.
51.          /*
52.           * Ok, we have completed the initial bootup, and
53.           * we're essentially up and running. Get rid of the
54.           * initmem segments and start the user-mode stuff..
55.           */
56.
57.          /* rootfs is available now, try loading default modules */
58.          load_default_modules();
59.  }
```

(0) _____ (0.1)_____(0.2)____ (1) _____(1.1)_____(1.2)_____(2)

|_____ (0.01)_____ (1)

所以create init process (2)总在driver初始化(1.2)之后。