in init/main.c

```
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);
7.
8.          /* Now the scheduler is fully set up and can do blocking allocations */
9.          gfp_allowed_mask = __GFP_BITS_MASK;
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();
                                                                    ①
25.         lockup_detector_init();
26.
27.         smp_init();

               ②
28.         sched_init_smp();
29.
30.         do_basic_setup();
                                                                          ③
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.     }
```

①

static void __init do_pre_smp_initcalls(void)

{

initcall_t *fn;

for (fn = __initcall_start; fn < __initcall0_start; fn++)

do_one_initcall(*fn);

}

in vmlinux.lds

__initcall_start = .; *(.initcallearly.init) **__initcall0_start** = .; *(.initcall0.init) *(.initcall0s.init) __initcall1_start = .; *(.initcall1.init) *(.initcall1s.init) __initcall2_start = .; *(.initcall2.init) *(.initcall2s.init) __initcall3_start = .; *(.initcall3.init) *(.initcall3s.init) __initcall4_start = .; *(.initcall4.init) *(.initcall4s.init) __initcall5_start = .; *(.initcall5.init) *(.initcall5s.init) __initcallrootfs_start = .; *(.initcallrootfs.init) *(.initcallrootfss.init) __initcall6_start = .; *(.initcall6.init) *(.initcall6s.init) __initcall7_start = .; *(.initcall7.init) *(.initcall7s.init) __initcall_end = .;

in vmlinux.lds.h

#define INIT_CALLS \

```
        VMLINUX_SYMBOL(__initcall_start) = .;                   \
        *(.initcallearly.init)                                  \
        INIT_CALLS_LEVEL(0)                                     \
        INIT_CALLS_LEVEL(1)                                     \
        INIT_CALLS_LEVEL(2)                                     \
        INIT_CALLS_LEVEL(3)                                     \
        INIT_CALLS_LEVEL(4)                                     \
        INIT_CALLS_LEVEL(5)                                     \
        INIT_CALLS_LEVEL(rootfs)                                \
        INIT_CALLS_LEVEL(6)                                     \
        INIT_CALLS_LEVEL(7)                                     \
        VMLINUX_SYMBOL(__initcall_end) = .;
```

在__initcall_start与__initcall0_start之间也就有early_initcall()定义的function table了。

```
/*
 * Early initcalls run before initializing SMP.
 *
 * Only for built-in code, not modules.
 */
#define early_initcall(fn)          __define_initcall(fn, early)
```

只有用early_initcall()定义的function才会在bootable core(single core)上运行，其他init level (从0到7都在SMP下运行)

②

在这儿是single core 与 mulyi-core的分界点，此前code运行在bootable cpu core(single core)上，此后SMP开工了。

③

built-in driver的初始化(比如driver的probe function)就在这里运行，所以driver framework的initialization及各个built-in driver initialization都是在SMP状况下运行的。