# linux 3.6 启动源码分析(二) start_kernel

原创 2013年12月16日 13:38:12 2910



在构架相关的汇编代码运行完之后，程序跳入了构架无关的内核C语言代码：init/main.c中的start_kernel函数，在这个函数中Linux内核开始真正进入初始化阶段，
进行一系列与内核相关的初始化后，调用第一个用户进程 - init 进程并等待用户进程的执行，这样整个 Linux 内核便启动完毕。该函数所做的具体工作有：

1) 调用 setup_arch()函数进行与体系结构相关的第一个初始化工作；

对不同的体系结构来说该函数有不同的定义。对于 ARM 平台而言，该函数定义在arch/arm/kernel/Setup.c。它首先通过检测出来的处理器类型进行处理器内核的初始化，然后通过 bootmem_init()函数根据系统定义的 meminfo 结构进行内存结构的初始化，最后调用paging_init()开启 MMU，创建内核页表，映射所有的物理内存和 IO空间。

2) 创建异常向量表和初始化中断处理函数；

3) 初始化系统核心进程调度器和时钟中断处理机制；

4) 初始化串口控制台（serial-console）；

5) 创建和初始化系统 cache，为各种内存调用机制提供缓存，包括;动态内存分配，虚拟文件系统（Virtual File System）及页缓存。

6) 初始化内存管理，检测内存大小及被内核占用的内存情况；

7) 初始化系统的进程间通信机制（IPC）

```cpp
asmlinkage void __init start_kernel(void)
{
    char * command_line;
    extern const struct kernel_param __start___param[], __stop___param[];
    /*这两个变量为地址指针，指向内核启动参数处理相关结构体段在内存中的位置（虚拟地址）。
    声明传入参数的外部参数对于ARM平台，位于 include\asm-generic\vmlinux.lds.h*/
    /*
     * Need to run as early as possible, to initialize the
     * lockdep hash:
     lockdep是一个内核调试模块，用来检查内核互斥机制（尤其是自旋锁）潜在的死锁问题。
     */
    lockdep_init();//初始化内核依赖的关系表，初始化hash表
    smp_setup_processor_id();//获取当前CPU,单处理器为空
    debug_objects_early_init();//对调试对象进行早期的初始化,其实就是HASH锁和静态对象池进行初始化

    /*
     * Set up the the initial canary ASAP:
     初始化栈canary值
     canary值的是用于防止栈溢出攻击的堆栈的保护字 。
     */
    boot_init_stack_canary();
    /*1.cgroup: 它的全称为control group.即一组进程的行为控制.
    2.该函数主要是做数据结构和其中链表的初始化
    3.参考资料： Linux cgroup机制分析之框架分析
    */
    cgroup_init_early();

    local_irq_disable();//关闭系统总中断（底层调用汇编指令）
    early_boot_irqs_disabled = true;

/*
 * Interrupts are still disabled. Do necessary setups, then
 * enable them
 */
    tick_init();//1.初始化内核时钟系统
    boot_cpu_init();//1.激活当前CPU（在内核全局变量中将当前CPU的状态设为激活状态）
    page_address_init();//高端内存相关，未定义高端内存的话为空函数
    printk(KERN_NOTICE "%s", linux_banner);
    /*1.内核构架相关初始化函数,可以说是非常重要的一个初始化步骤。
    其中包含了处理器相关参数的初始化、内核启动参数（tagged list）的获取和前期处理、
    内存子系统的早期的初始化（bootmem分配器）。 主要完成了4个方面的工作，一个就是取得MACHINE和PROCESSOR的信息然或将他们赋值
    给kernel相应的全局变量，然后呢是对boot_command_line和tags接行解析，再然后呢就是
    memory、cach的初始化，最后是为kernel的后续运行请求资源”**/
    setup_arch(&command_line);
    /*1.初始化代表内核本身内
    存使用的管理结构体init_mm。
    2.ps：每一个任务都有一个mm_struct结构以管理内存空间，init_mm是内核的mm_struct，其中：
    3.设置成员变量* mmap指向自己，意味着内核只有一个内存管理结构;
    4.设置* pgd=swapper_pg_dir，swapper_pg_dir是内核的页目录(在arm体系结构有16k， 所以init_mm定义了整个kernel的内存空间)。
    5.这些内容涉及到内存管理子系统*/
    mm_init_owner(&init_mm, &init_task);
    mm_init_cpumask(&init_mm);//初始化CPU屏蔽字
    /*1.对cmdline进行备份和保存：保存未改变的comand_line到字符数组static_command_line［］ 中。保存 boot_command_line到字符数组saved_command_line［］中
    */
    setup_command_line(command_line);

    /*如果没有定义CONFIG_SMP宏，则这个函数为空函数。如果定义了CONFIG_SMP宏，则这个setup_per_cpu_areas()函数给每个CPU分配内存，并拷贝.data.percpu段的数据。为系统中的每个CPU的per_cpu变量申请空间。
    */
    /*下面三段1.针对SMP处理器的内存初始化函数，如果不是SMP系统则都为空函数。 (arm为空)
    2.他们的目的是给每个CPU分配内存，并拷贝.data.percpu段的数据。为系统中的每个CPU的per_cpu变量申请空间并为boot CPU设置一些数据。
    3.在SMP系统中，在引导过程中使用的CPU称为boot CPU*/
    setup_nr_cpu_ids();
    setup_per_cpu_areas();
    smp_prepare_boot_cpu(); /* arch-specific boot-cpu hooks */


    build_all_zonelists(NULL, NULL);//  建立系统内存页区(zone)链表

    page_alloc_init();//内存页初始化


```

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+

```
72.     printk(KERN_NOTICE "Kernel command line: %s\n", boot_command_line);
73.     parse_early_param();//  解析早期格式的内核参数
74.     /*函数对Linux启动命令行参数进行在分析和处理,
75.     当不能够识别前面的命令时，所调用的函数。*/
76.     parse_args("Booting kernel", static_command_line, __start___param,
77.            __stop___param - __start___param,
78.            -1, -1, &unknown_bootoption);
79.     jump_label_init();
80.     /*
81.      * These use large bootmem allocations and must precede
82.      * kmem_cache_init()
83.      */
84.     setup_log_buf(0);
85.     /*初始化hash表，以便于从进程的PID获得对应的进程描述指针，按照开发办上的物理内存初始化pid hash表
86.     */
87.     pidhash_init();
88.     vfs_caches_init_early();//建立节点哈希表和数据缓冲哈希表
89.     sort_main_extable();//对异常处理函数进行排序
90.     trap_init();//初始化硬件中断
91.     mm_init();//    Set up kernel memory allocators      建立了内核的内存分配器
92.     /*
93.      * Set up the scheduler prior starting any interrupts (such as the
94.      * timer interrupt). Full topology setup happens at smp_init()
95.      * time - but meanwhile we still have a functioning scheduler.
96.      */
97.     sched_init();
98.     /*
99.      * Disable preemption - early bootup scheduling is extremely
100.     * fragile until we cpu_idle() for the first time.
101.     */
102.    preempt_disable();//禁止调度
103.    //  先检查中断是否已经打开，若打开，输出信息后则关闭中断。
104.    if (!irqs_disabled()) {
105.        printk(KERN_WARNING "start_kernel(): bug: interrupts were "
106.                "enabled *very* early, fixing it\n");
107.        local_irq_disable();
108.    }
109.    idr_init_cache();//创建錳dr缓冲区
110.    perf_event_init();
111.    rcu_init();//互斥访问机制
112.    radix_tree_init();
113.    /* init some links before init_ISA_irqs() */
114.    early_irq_init();
115.    init_IRQ();//中断向量初始化
116.    prio_tree_init();//初始化优先级数组
117.    init_timers();//定时器初始化
118.    hrtimers_init();//高精度时钟初始化
119.    softirq_init();//软中断初始化
120.    timekeeping_init();//   初始化资源和普通计时器
121.    time_init();
122.    profile_init();//   对内核的一个性能测试工具profile进行初始化。
123.    call_function_init();
124.    if (!irqs_disabled())
125.        printk(KERN_CRIT "start_kernel(): bug: interrupts were "
126.                "enabled early\n");
127.    early_boot_irqs_disabled = false;
128.    local_irq_enable();//使能中断
129.    kmem_cache_init_late();//kmem_cache_init_late的目的就在于完善slab分配器的缓存机制.
130.
131.    /*
132.     * HACK ALERT! This is early. We're enabling the console before
133.     * we've done PCI setups etc, and console_init() must be aware of
134.     * this. But we do want output early, in case something goes wrong.
135.     */
136.    console_init();//初始化控制台以显示printk的内容
137.    if (panic_later)
138.        panic(panic_later, panic_param);
139.
140.    lockdep_info();//   如果定义了CONFIG_LOCKDEP宏，那么就打印锁依赖信息，否则什么也不做
141.
142.    /*
143.     * Need to run this when irqs are enabled, because it wants
144.     * to self-test [hard/soft]-irqs on/off lock inversion bugs
145.     * too:
146.     */
147.    locking_selftest();
148. #ifdef CONFIG_BLK_DEV_INITRD
149.    if (initrd_start && !initrd_below_start_ok &&
150.        page_to_pfn(virt_to_page((void *)initrd_start)) < min_low_pfn) {
151.        printk(KERN_CRIT "initrd overwritten (0x%08lx < 0x%08lx) - "
152.            "disabling it.\n",
153.            page_to_pfn(virt_to_page((void *)initrd_start)),
154.            min_low_pfn);
155.        initrd_start = 0;
156.    }
157. #endif
158.    page_cgroup_init();
159.    debug_objects_mem_init();
160.    kmemleak_init();
161.    setup_per_cpu_pageset();
162.    numa_policy_init();
163.    if (late_time_init)
164.        late_time_init();
165.    sched_clock_init();
166.    calibrate_delay();//校准延时函数的精确度
167.    pidmap_init();//进程号位图初始化，一般用一个鎊age来表示所有进程的鎊id占用情况
168.    anon_vma_init();    //  匿名虚拟内存域（ anonymous VMA）初始化
169. #ifdef CONFIG_X86
170.    if (efi_enabled)
171.        efi_enter_virtual_mode();
172. #endif
173.    thread_info_cache_init();//获取thread_info缓存空间，大部分构架为空函数（包括ARM
174.    cred_init();    //任务信用系统初始化。详见：Documentation/credentials.txt
175.    fork_init(totalram_pages);  //进程创建机制初始化。为内核"task_struct"分配空间，计算最大任务数。
176.    proc_caches_init(); //初始化进程创建机制所需的其他数据结构，为其申请空间。
177.    buffer_init();  //缓存系统初始化，创建缓存头空间，并检查其大小限时。
178.    key_init(); //内核密钥管理系统初始化
179.    security_init();    //内核安全框架初始?
180.    dbg_late_init();
181.    vfs_caches_init(totalram_pages);vfs_caches_init(totalram_pages);//虚拟文件系统（VFS）缓存初始化
182.    signals_init();//信号管理系统初始化
183.    /* rootfs populating might need page-writeback */
184.    page_writeback_init();//页写回机制初始化
185. #ifdef CONFIG_PROC_FS
186.    proc_root_init();//proc文件系统初始化
187. #endif
188.    cgroup_init();//control group正式初始化
189.    cpuset_init();//CPUSET初始化。 参考资料：《多核心計算環境—NUMA與CPUSET簡介》
190.    taskstats_init_early(); //任务状态早期初始化函数：为结构体获取高速缓存，并初始化互斥机制。
191.    delayacct_init();   //任务延迟初始化
```

```
192.
193.     check_bugs();//检查CPU BUG的函数，通过软件规避BUG
194.     acpi_early_init(); /* before LAPIC and SMP initACPI早期初始化函数。 ACPI - Advanced Configuration an
d Power Interface高级配置及电源接口 */
195.     sfi_init_late();//功能跟踪调试机制初始化，ftrace 是 function trace 的简称
196.
197.     if (efi_enabled)
198.         efi_free_boot_services();
199.     ftrace_init();
200.     /* Do the rest non-__init'ed, we're now alive */
201.     rest_init();// 虽然从名字上来说是剩余的初始化。但是这个函数中的初始化包含了很多的内容
202. }
```

在看完上面的代码之后，你会发现内容很多。但是归纳起来，我认为需要注意的有以下几点：

1.内核启动参数的获取和处理

2.setup_arch(&command_line);函数

3.内存管理的初始化（从bootmem到slab）

4.各种内核体系的初始化

5.rest_init();函数