# CPUIDLE 之低功耗定时器

28 May 2016

在 Linux 操作系统中，Idle 进程（又叫 Swapper 进程）的 pid 号是 0，是所有进程的祖先，它是在 Linux 初始化阶段从无到有创建的一个内核线程。`stark_kernel()` 函数初始化内核需要的所有数据结构，激活中断，创建另一个叫进程 1 的内核线程（init 进程）。新创建内核线程的 PID 为 1，并与进程 0 共享进程所有的内核数据结构。创建 init 进程后，进程 0 执行无限循环，`cpu_idle_loop()` 函数，只有当没有其它进程处于 TASK_RUNNING 状态时，调度器才会选择进程 0，也就是执行 Idle 线程，让 CPU 进入 Idle 模式。

当 CPU 进入比较深层次的 Idle 模式时，为了达到最大程度的节省功耗，CPU 会把系统时钟给关闭掉。那么怎么保证 CPU 从 Idle 模式重新进入工作模式，从而保证系统正常运行呢？本文主要探讨低功耗定时器在该情况下扮演什么样的角色。

## Idle的执行流程

这里先大概的列出 Idle 进程的执行流程，怎么执行到让 CPU 进入各级 Idle 模式。详细的分析会在以后的文章中补上。

### Idle 进程创建

Linux Kernel 会在系统启动完成后，在 Idle 进程中，处理 CPUIdle 相关的事情。在多核系统中，CPU 启动的过程是，先启动主 CPU，启动过程和传统的单核系统类似。其函数调用关系如下：

stext –> start_kernel –> rest_init –> cpu_startup_entry

而启动其它 CPU，可以有多种方式，例如 CPU hotplug 等，启动过程：

secondary_startup –> __secondary_switched –> secondary_start_kernel –> cpu_startup_entry

无论是上述的哪种启动，最终都会运行至 cpu_startup_entry 函数，在这个函数中，最终程序会掉进无限循环里 cpu_idle_loop。到此，Idle 进程创建完成，以下是 Idle 进程的代码实现（参考的是 linux-3.18 版本）：

```
static void cpu_idle_loop(void)
{
```

```
        while (1) {
                __current_set_polling();

                /* 关闭周期 tick，CONFIG_NO_HZ_IDLE 必须打开 */
                tick_nohz_idle_enter();

                /* 如果系统当前不需要调度，执行后续的动作 */
                while (!need_resched()) {
                        check_pgt_cache();
                        rmb();

                        if (cpu_is_offline(smp_processor_id())) {
                                tick_set_cpu_plugoff_flag(1);
                                arch_cpu_idle_dead();
                        }

                        /* 关闭 irq 中断 */
                        local_irq_disable();

                        /* arch 相关的 cpuidle enter，主要执行注册到
                        arch_cpu_idle_enter();

                        if (cpu_idle_force_poll || tick_check_bro

                                /* idle poll，这时先不分析 */
                                cpu_idle_poll();
                        else

                                /* 进入 CPU 的 Idle 模式，进行省电 *
                                cpuidle_idle_call();

                        /* Idle 退出，主要执行注册到 idle 的 notify
                        arch_cpu_idle_exit();
                }

                /* 如果系统当前需要调度，则退出 Idle 进程 */
                preempt_set_need_resched();

                /* 打开周期 tick */
                tick_nohz_idle_exit();
                __current_clr_polling();

                smp_mb__after_atomic();

                sched_ttwu_pending();

                /* 让出 cpu，使调度器调度其它优化级更高的进程 */
                schedule_preempt_disabled();
        }
}
```

## Tickless 和 CPUIdle 的关系

Tickless 是指动态时钟，即系统的周期 Tick 可动态地关闭和打开。这个功能可通过内核配置项 CONFIG_NO_HZ 打开，而 Idle 正是使用了这项技术，使系统尽量长时间处于空闲状态，从而尽可能地节省功耗.

打开内核配置项 CONFIG_NO_HZ_IDLE，即可让系统在 Idle 前关闭周期 Tick，退出 Idle 时重新打开周期 Tick。

那么在关闭了周期 Tick 之后，系统何时被唤醒呢?

在关闭周期 Tick 时，同时会根据时钟子系统计算下一个时钟中断到来的时间，以这个时间为基准来设置一个 hrtimer 用于唤醒系统（高精度时钟框架），而这个时间的计算方法也很简单，即在所有注册到时钟框架的定时器中找到离此时最近的那一个的时间点作为这个时间。当然，用什么定时器来唤醒系统还要根据 CPU Idle 的深度来决定，后面会介绍。

## 不同层级的 CPU Idle 对唤醒时钟源的处理

前面提到了，系统关闭周期 Tick 的同时，会计算出下一个时钟中断到来的时间，以这个时间为基准来设置一个 hrtimer 用于唤醒系统。那么，如果有些 CPU 进入的层级比较深，关闭了 CPU 中的 hrtimer，系统将无法再次被唤醒。针对这种情况，则需要低功耗 Timer 去唤醒系统，这里先以 MTK 平台为例，在 CPU 进入 dpidle 和 soidle （两种 Idle 模式）时都会关闭 hrtimer ，另外起用一个 GPT Timer，而这个 GPT Timer 的超时时间直接从被关闭的 hrtimer 中的寄存器获取。这样就保证时间的延续性。因为 GPT Timer 是以 32K 晶振作为时钟源，所以在 CPU 进入 dpidle 时可以把 26M 的主时钟源给关闭，从而达到最大程度的省电。

以下我们通过源码探讨一下 MTK 的 CPU Idle 的实现，到底哪里设置 GPT Timer。其实很多平台为了实现 CPU 达到最省电的效果，都是使用这种做法。

MTK 的 CPU 一般有以下几种 Idle 模式

- rgidle，浅度 Idle 模式，即 WFI
- soidle，亮屏 Idle 模式
- dpidle，灭屏 Idle 模式

先以 dpidle 模式为例子分析 CPU 如何在关闭所有系统时钟的情况下保证 Idle 和系统正常运行。

CPU 进入 dpidle 的实现接口如下:

```
int dpidle_enter(int cpu)
{
        int ret = IDLE_TYPE_DP;

        /* 记录 dpidle 的开始时间 */
```

```
        idle_ratio_calc_start(IDLE_TYPE_DP, cpu);

        /* 关闭一些与平台相关的定时器，hps，thermal */
        dpidle_pre_handler();

        /* 进入 dpidle */
        spm_go_to_dpidle(slp_spm_deepidle_flags, (u32)cpu, dpidle_

        /* 打开一些与平台相关的定时器，hps，thermal */
        dpidle_post_handler();

        /* 记录 dpidle 的退出时间，从而计算出 CPU 进入 dpidle 的总时间
        idle_ratio_calc_stop(IDLE_TYPE_DP, cpu);


        idle_warn_log("DP:timer_left=%d, timer_left2=%d, delta=%d"
                      dpidle_timer_left, dpidle_timer_left2, dp

        /* For test */
        if (dpidle_run_once)
                idle_switch[IDLE_TYPE_DP] = 0;

        return ret;
}
```

真正进入 dpidle 的实现在 spm_go_to_dpidle 函数

```
wake_reason_t spm_go_to_dpidle(u32 spm_flags, u32 spm_data, u32 d
{
        ...

        /* 更新 spm 的标志 */
        update_pwrctrl_pcm_flags(&spm_flags);
        /* 设置 spm 标志位 */
        set_pwrctrl_pcm_flags(pwrctrl, spm_flags);

        /* 设置 GPT4 定时器，超时时间为下一个 Timer 的唤醒时间，并开启
        spm_dpidle_before_wfi(cpu);

        lockdep_off();
        spin_lock_irqsave(&__spm_lock, flags);
        /* 屏蔽 GIC 控制器中的所有中断 */
        mt_irq_mask_all(&mask);
        /* 打开 GIC 中 SPM_IRQ0_ID 号中断，用于唤醒 */
        mt_irq_unmask_for_sleep(SPM_IRQ0_ID);

        if (request_uart_to_sleep()) {
                wr = WR_UART_BUSY;
                goto RESTORE_IRQ;
        }
```

```
        ...

        /* 关闭系统时钟 */
        spm_dpidle_pre_process();
        /* 进入 Idle */
        spm_trigger_wfi_for_dpidle(pwrctrl);

        ...

RESTORE_IRQ:
        mt_irq_mask_restore(&mask);
        spin_unlock_irqrestore(&__spm_lock, flags);
        lockdep_on();
        spm_dpidle_after_wfi(cpu, wakesta.debug_flag);

        return wr;
}
```

在 spm_dpidle_before_wfi 函数中会去设置 GPT Timer 的定时时间，并且开启这个 Timer，设置完毕后关闭 GIC 的所有中断，只打开 GPT Timer IRQ，保证 CPU 在定时时间到期时被 GPT Timer 唤醒。最后调用 spm_dpidle_pre_process 和 spm_trigger_wfi_for_dpidle 函数关闭系统时钟并进入 dpidle。

```
void spm_dpidle_before_wfi(int cpu)
{
        bus_dcm_enable();
        faudintbus_pll2sq();

        /* 从 localtimer 中获取定时器计数 */
        dpidle_timer_left2 = localtimer_get_counter();

        if ((int)dpidle_timer_left2 <= 0)
                /* Trigger GPT4 Timeout imediately */
                gpt_set_cmp(idle_gpt, 1);
        else
                /* 把从 localtimer 中获取到的定时器计数设置到 GPT Tim
                gpt_set_cmp(idle_gpt, dpidle_timer_left2);

        /* 启动 GPT Timer */
        start_gpt(idle_gpt);
}
```

localtimer_get_counter 函数中其实就是读取 hrtimer 的定时器里的剩下计数 count，然后把 count 设置到 GPT Timer 中，这种做法非常简便和巧妙。这样就能保障系统的正常运行又能达到最省电的效果了。

qkhhyga

Linux Engineer



电源管理 2 cpuidle 1 hrtimer 2 tickless 2 Linux 7

← 上一篇 文章列表 下一篇 →