

密级状态：绝密() 秘密() 内部资料() 公开(☒)

休眠唤醒流程

(技术研发部，底层平台中心)

文件状态： [] 草稿 [<input checked="" type="checkbox"/>] 正式发布 [] 正在修改	文件标识：	休眠唤醒流程
	当前版本：	1.0
	审 核：	
	作 者：	谢修鑫、许盛飞、陈建洪、黄小东
	完成日期：	2018-01-02

版 本 历 史

版本号	作者	修改日期	修改说明
1.0	许盛飞	2018-01-02	初稿

目录

概述.....	4
一、 基本概念.....	4
1.1、“一级休眠”（浅度休眠）：	4
1.2、 二级休眠（深度休眠）：	4
1.3、 SOC 低功耗处理：	4
二、 Suspend/resume 有关的代码分布.....	4
2.1、 PM Core.....	4
2.2、 Device PM.....	5
2.3、 Platform dependent PM.....	5
三、 suspend&resume 过程概述.....	6
四、 休眠代码流程分析.....	6
4.1、 代码调用流程.....	7
4.2、 休眠唤醒中重要的结构体.....	7
4.2.1、 struct platform_suspend_ops.....	8
4.2.2、 struct dev_pm_info.....	8
4.2.3、 struct dev_pm_ops.....	9
4.2.4、 struct syscore_ops.....	9
4.2.5、 休眠唤醒流程不同的划分方式：	9

概述

本文档主要简介休眠唤醒流程及比较重要的结构体。

一、基本概念

1.1、“一级休眠”（浅度休眠）：

“一级休眠”与 earlysuspend 不同，要实现一级休眠的设备，接收到 fb 的 FB_EARLY_EVENT_BLANK 这个 notify 后，再运行相应应的低功耗操作。

哪些设备需要实现一级休眠：lcd、tp，背光，Sensor 等设备；

一级休眠主要应用场景有：听音乐，蓝牙数据传输，wifi 后台下载等。

USB 连接电脑是不能进二级休眠，也就只能停留在一级休眠这个阶段。

1.2、二级休眠（深度休眠）：

Linux 内核提供了三种 Suspend: Freeze、Standby 和 STR(Suspend to RAM)，在用户空间向”/sys/power/state”文件分别写入”freeze”、”standby”和”mem”，即可触发它们。

当前平台只支持 freeze，mem 两种：

Freeze: 无需平台代码参与即可支持，而 freeze 暂时没有场景用到这个。

mem: 需要平台实现 platform_suspend_ops 的相关函数。

standby: 我们平台不支持，但实现的流程与 mem 类似，需要实现 platform_suspend_ops 的相关函数；

1.3、SOC 低功耗处理：

在 trust.img 中，有针对芯片级的低功耗处理。请参照《RK 平台 SOC 休眠唤醒原理》

二、Suspend/resume 有关的代码分布

2.1、PM Core

kernel/power/main.c----提供用户空间接口 (/sys/power/state)

kernel/power/suspend.c----Suspend 功能的主逻辑

kernel/power/suspend_test.c----Suspend 功能的测试逻辑

kernel/power/console.c----Suspend 过程中对控制台的处理逻辑

kernel/power/process.c----Suspend 过程中对进程的处理逻辑

2.2、Device PM

drivers/base/power/main.c---设备功耗相关函数调用接口

2.3、Platform dependent PM

针对 3.10


arch/arm/mach-rockchip/pm_rk3xxx.c 实现相关的 API 供
rockchip_pm.c 中相关 SOC 操作的流程调用;

arch/xxx/plat-xxx/rockchip_pm.c----平台相关的电源管理操作

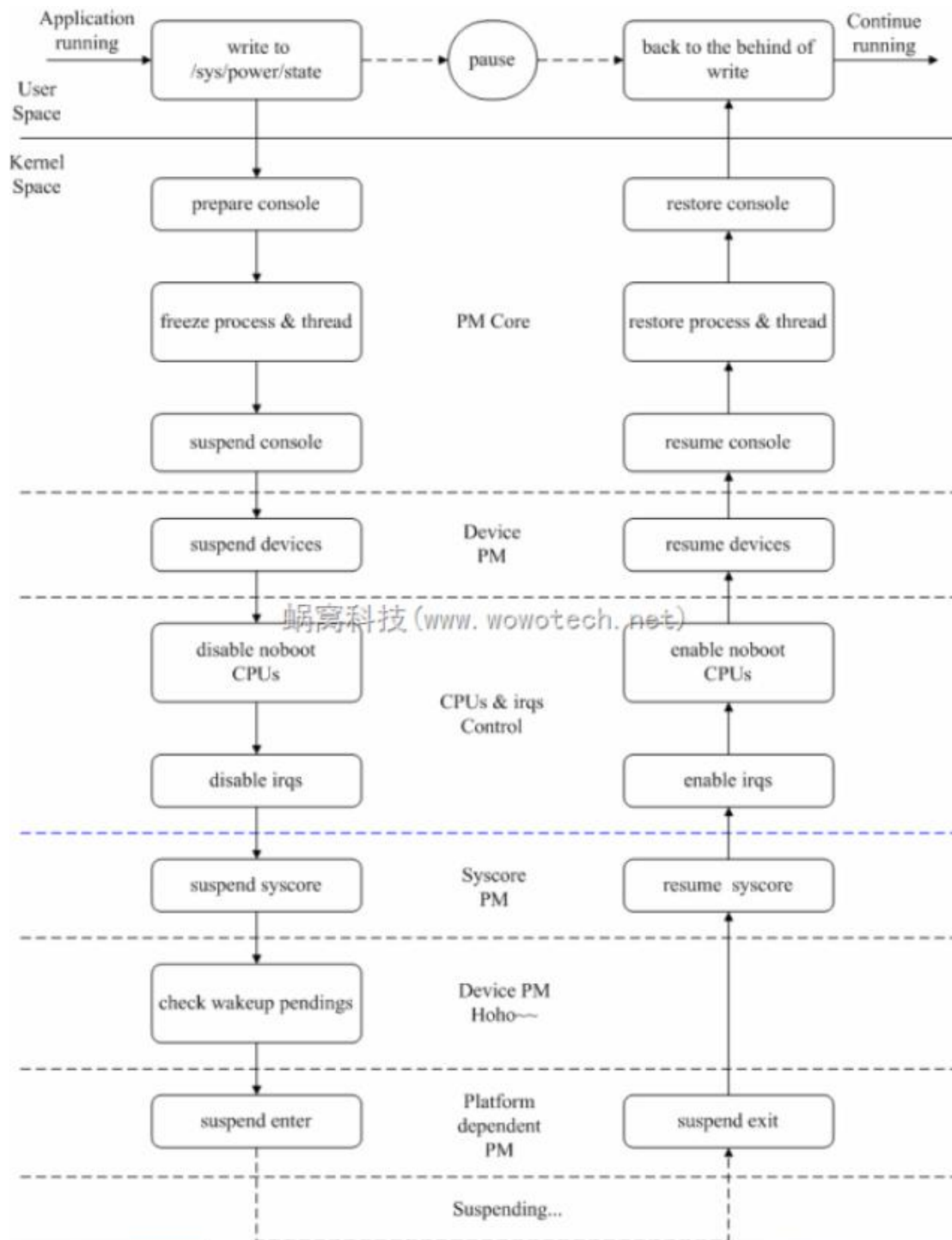
针对 4.4

通过 psci 的调用陷入 ATF 或者 OPTE(即我们烧写的 trust.img)

```
00392:
00393: static const struct platform_suspend_ops psci_suspend_ops = {
00394:     .valid      = suspend_valid_only_mem,
00395:     .enter      = psci_system_suspend_enter,
00396:     .prepare    = psci_system_suspend_prepare,
00397:     .finish     = psci_system_suspend_finish,
00398: };
00399:
```



三、 suspend&resume 过程概述



四、 休眠代码流程分析

4.1、代码调用流程

```

enter_state
---->suspend_prepare(state);
---->suspend_test(TEST_FREEZER)
----> suspend_devices_and_enter(state);
    -->platform_suspend_begin(state);(suspend_ops->begin 如果是 freeze 模式,
则是调用 freeze_ops)
    ---->suspend_console();
    ----> dpm_suspend_start(PMSG_SUSPEND);
    (#define PMSG_SUSPEND ((struct pm_message){ .event = PM_EVENT_SUSPEND, }))
    ---->dpm_prepare(state);
        --->device_prepare(dev, state);
            (pm->prepare)(prepare 阶段没有类似 pm_prepare_op 这样的
接口直接调用)
        ---->dpm_suspend(state);
            --->device_suspend(dev)
                (ops->suspend) (pm_op)
    ---->suspend_test(TEST_DEVICES)
    ---->suspend_enter(state, &wakeup);
        -->platform_suspend_prepare(state);(suspend_ops->prepare)
        ---->dpm_suspend_late(PMSG_SUSPEND);
            (ops->suspend_late) (pm_late_early_op)
        -->platform_suspend_prepare_late(state);(freeze_ops->prepare)
与 suspend_ops 无关, 这个是 freeze_ops 的调用。
        ---->suspend_test(TEST_PLATFORM)
        ---->dpm_suspend_noirq(PMSG_SUSPEND);
            (ops->suspend_noirq) (pm_noirq_op)
        -->platform_suspend_prepare_noirq(state)(suspend_ops->prepare
_late)

        ---->disable_nonboot_cpus();
        ---->suspend_test(TEST_CPUS)
        ---->arch_suspend_disable_irqs();
        ----> syscore_suspend();
        ---->suspend_test(TEST_CORE)
        ---->suspend_ops->enter(state);
    
```

4.2、休眠唤醒中重要的结构体

与休眠唤醒关系比较紧密的数据结构有以下 4 个：

device_pm_ops 和 platform_pm_ops 就是电源管理（suspend）的全部，只要在合适的地方，实现合适的回调函数，即可实现系统的电源管理

4.2.1、struct platform_suspend_ops

与平台相关，系统如果要支持休眠唤醒的功能，必须初始化这个结构并调用 suspend_set_ops() 初始化 suspend_ops，在休眠的过程中会判断 suspend_ops 且当前的模式是否有效

(suspend_ops && suspend_ops->valid && suspend_ops->valid(state);)。

```
static const struct platform_suspend_ops psci_suspend_ops = {
    .valid      = suspend_valid_only_mem,
    .enter      = psci_system_suspend_enter,
    .prepare    = psci_system_suspend_prepare,
    .finish     = psci_system_suspend_finish,
};

static void __init psci_init_system_suspend(void)
{
    int ret;

    if (!IS_ENABLED(CONFIG_SUSPEND))
        return;

    ret = psci_features(PSCI_FN_NATIVE(1_0, SYSTEM_SUSPEND));

    if (ret != PSCI_RET_NOT_SUPPORTED)
        suspend_set_ops(&psci_suspend_ops);
}
```

4.2.2、struct dev_pm_info

将该设备纳入电源管理的范围，记录电源管理的一些信息,针对每个设备的 pm 相关的信息都这个这个结构体中，在休眠唤醒的过程中会更新对应的状态值；power 变量主要保存 PM 相关的状态，如当前的 power_state、是否可以被唤醒、是否已经 prepare 完成、是否已经 suspend 完成等等。涉及的内容非常多。

```
struct device {
    struct device *parent;

    struct device_private *p;

    struct kobject kobj;
    const char *init_name; /* initial name of the device */
    const struct device_type *type;

    struct mutex mutex; /* mutex to synchronize calls to
                        * its driver.
                        */

    struct bus_type *bus; /* type of bus device is on */
    struct device_driver *driver; /* which driver has allocated this
                                device */

    void *platform_data; /* Platform specific data, device
                        core doesn't touch it */
    void *driver_data; /* Driver data, set and get with
                        dev_set/get_drvdata */

    struct dev_pm_info power;
    struct dev_pm_domain pm_domain;
};
```

4.2.3、struct dev_pm_ops

设备电源管理的核心数据结构，用于封装和设备电源管理有关的所有操作该结构基本上该有的东西都有，主要分为几类：

传统 suspend 的常规路径，prepare/complete、suspend/resume、freeze/thaw、poweroff、restore；

传统 suspend 的特殊路径，early_suspend/late_suspend、noirq；

runtime PM: runtime_suspend/runtime_resume/runtime_idle。

各类 driver 需要做的事情就是实现必要的回调函数，并保存在合适的位置。

4.2.4、struct syscore_ops

syscore 作为低功耗流程的一部分，其涉及的文件主要有 syscore_ops.h 和 syscore.c，这一级别的回调函数是在**完全屏蔽中断**的场景下进行的。

4.2.5、休眠唤醒流程不同的划分方式：

休眠按/sys/power/pm_test 节点分可以分成一下几个部分：

core processors platform devices freezer(这个 freezer 与之前的 freezer mem 那边的是不一样的)

按 struct platform_suspend_ops 这个结构体的调用流程我们可以分成：

Begin prepare prepare_late enter

针对 struct dev_pm_ops 的角度来说 suspend/resume 的过程中，PM Core 会依次调用流程

“**prepare**——>**suspend**——>**suspend_late**——>**suspend_noirq**----->**wakeup**----->**resume_noirq**---> **resume_early**——>**resume**----->**complete**”