[蜗窝科技](http://www.wowotech.net/)

*慢下来，享受技术。*

[](http://www.wowotech.net/)

* [博客](http://www.wowotech.net/)
* [项目](http://www.wowotech.net/sort/project)
* [讨论区](http://www.wowotech.net/forum)
* [关于蜗窝](http://www.wowotech.net/about.html)
* [联系我们](http://www.wowotech.net/contact_us.html)
* [支持我们](http://www.wowotech.net/support_us.html)
* [登录](http://www.wowotech.net/admin)

﻿

**Linux电源管理(6)\_Generic PM之Suspend功能**

作者：[wowo](http://www.wowotech.net/author/2) 发布于：2014-8-22 21:40 分类：[电源管理子系统](http://www.wowotech.net/sort/pm_subsystem)

**1. 前言**

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。

内核中，Suspend及Resume过程涉及到PM Core、Device PM、各个设备的驱动、Platform dependent PM、CPU control等多个模块，涉及了console switch、process freeze、CPU hotplug、wakeup处理等过个知识点。就让我们跟着内核代码，一一见识它们吧。

**2. Suspend功能有关的代码分布**

内核中Suspend功能有关的代码包括PM core、Device PM、Platform PM等几大块，具体如下：

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）Device PM

drivers/base/power/\*----具体可参考“[Linux电源管理(4)\_Power Management Interface](http://www.wowotech.net/linux_kenrel/pm_interface.html)”的描述。

设备驱动----具体设备驱动的位置，不再涉及。

3）Platform dependent PM

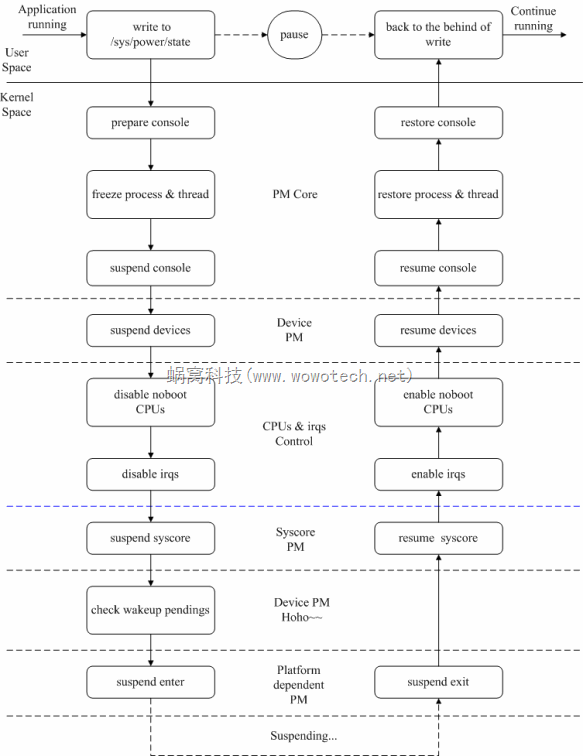
include/linux/suspend.h----定义platform dependent PM有关的操作函数集

arch/xxx/mach-xxx/xxx.c或者

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

**3. suspend&resume过程概述**

下面图片对Linux suspend&resume过程做了一个概述，读者可以顺着这个流程阅读内核源代码。具体的说明，可以参考后面的代码分析。

[](http://www.wowotech.net/content/uploadfile/201408/d7d8826b98dcd42ff63de66398709a9a20140822134032.gif)

**4. 代码分析**

**4.1 suspend入口**

在用户空间执行如下操作：

echo "freeze" > /sys/power/state

echo "standby" > /sys/power/state

echo "mem" > /sys/power/state

会通过sysfs触发suspend的执行，相应的处理代码如下：

1. **static** ssize\_t state\_store(**struct** kobject \*kobj, **struct** kobj\_attribute \*attr,
2. **const** **char** \*buf, size\_t n)
3. {
4. suspend\_state\_t state;
5. int error;
7. error = pm\_autosleep\_lock();
8. **if** (error)
9. **return** error;
11. **if** (pm\_autosleep\_state() > PM\_SUSPEND\_ON) {
12. error = -EBUSY;
13. **goto** out;
14. }
16. state = decode\_state(buf, n);
17. **if** (state < PM\_SUSPEND\_MAX)
18. error = pm\_suspend(state);
19. **else** **if** (state == PM\_SUSPEND\_MAX)
20. error = hibernate();
21. **else**
22. error = -EINVAL;
24. out:
25. pm\_autosleep\_unlock();
26. **return** error ? error : n;
27. }
29. power\_attr(state);

power\_attr定义了一个名称为state的attribute文件，该文件的store接口为state\_store，该接口在lock住autosleep功能后，解析用户传入的buffer（freeze、standby or mem），转换成state参数。

state参数的类型为suspend\_state\_t，在include\linux\suspend.h中定义，为电源管理状态在内核中的表示。具体如下：

1. **typedef** int \_\_bitwise suspend\_state\_t;
3. #define PM\_SUSPEND\_ON ((\_\_force suspend\_state\_t) 0)
4. #define PM\_SUSPEND\_FREEZE ((\_\_force suspend\_state\_t) 1)
5. #define PM\_SUSPEND\_STANDBY ((\_\_force suspend\_state\_t) 2)
6. #define PM\_SUSPEND\_MEM ((\_\_force suspend\_state\_t) 3)
7. #define PM\_SUSPEND\_MIN PM\_SUSPEND\_FREEZE
8. #define PM\_SUSPEND\_MAX ((\_\_force suspend\_state\_t) 4)

根据state的值，如果不是（PM\_SUSPEND\_MAX，对应hibernate功能），则调用pm\_suspend接口，进行后续的处理。

pm\_suspend在kernel/power/suspend.c定义，处理所有的suspend过程。

**4.2 pm\_suspend & enter\_state**

pm\_suspend的实现非常简单，简单的做一下参数合法性判断，直接调用enter\_state接口，如下：

1. int pm\_suspend(suspend\_state\_t state)
2. {
3. int error;
5. **if** (state <= PM\_SUSPEND\_ON || state >= PM\_SUSPEND\_MAX)
6. **return** -EINVAL;
8. error = enter\_state(state);
9. **if** (error) {
10. suspend\_stats.fail++;
11. dpm\_save\_failed\_errno(error);
12. } **else** {
13. suspend\_stats.success++;
14. }
15. **return** error;
16. }

enter\_state代码为：

1. **static** int enter\_state(suspend\_state\_t state)
2. {
3. int error;
5. **if** (!valid\_state(state))
6. **return** -ENODEV;
8. **if** (!mutex\_trylock(&pm\_mutex))
9. **return** -EBUSY;
11. **if** (state == PM\_SUSPEND\_FREEZE)
12. freeze\_begin();
14. printk(KERN\_INFO "PM: Syncing filesystems ... ");
15. sys\_sync();
16. printk("done.\n");
18. pr\_debug("PM: Preparing system for %s sleep\n", pm\_states[state]);
19. error = suspend\_prepare(state);
20. **if** (error)
21. **goto** Unlock;
23. **if** (suspend\_test(TEST\_FREEZER))
24. **goto** Finish;
26. pr\_debug("PM: Entering %s sleep\n", pm\_states[state]);
27. pm\_restrict\_gfp\_mask();
28. error = suspend\_devices\_and\_enter(state);
29. pm\_restore\_gfp\_mask();
31. Finish:
32. pr\_debug("PM: Finishing wakeup.\n");
33. suspend\_finish();
34. Unlock:
35. mutex\_unlock(&pm\_mutex);
36. **return** error;
37. }

主要工作包括：

a）调用valid\_state，判断该平台是否支持该电源状态。

suspend的最终目的，是让系统进入可恢复的挂起状态，而该功能必须有平台相关代码的参与才能完成，因此内核PM Core就提供了一系列的回调函数（封装在platform\_suspend\_ops中），让平台代码（如arch/arm/mach-xxx/pm.c）实现，然后由PM Core在合适的时机调用。这些回调函数包含一个valid函数，就是用来告知PM Core，支持哪些state。

最后看一下valid\_state的实现（删除了无关代码）：

1. **bool** valid\_state(suspend\_state\_t state)
2. {
3. **if** (state == PM\_SUSPEND\_FREEZE) {
4. **return** **true**;
5. }
6. /\*
7. \* PM\_SUSPEND\_STANDBY and PM\_SUSPEND\_MEMORY states need lowlevel
8. \* support and need to be valid to the lowlevel
9. \* implementation, no valid callback implies that none are valid.
10. \*/
11. **return** suspend\_ops && suspend\_ops->valid && suspend\_ops->valid(state);
12. }

如果是freeze，无需平台代码参与即可支持，直接返回true。对于standby和mem，则需要调用suspend\_ops的valid回掉，由底层平台代码判断是否支持。

b）加互斥锁，只允许一个实例处理suspend。

c）如果state是freeze，调用freeze\_begin，进行suspend to freeze相关的特殊动作。我会在后面统一分析freeze的特殊动作，这里暂不描述。

d）打印提示信息，同步文件系统。

e）调用suspend\_prepare，进行suspend前的准备，主要包括switch console和process&thread freezing。如果失败，则终止suspend过程。

f）然后，调用suspend\_devices\_and\_enter接口，该接口负责suspend和resume的所有实际动作。前半部分，suspend console、suspend device、关中断、调用平台相关的suspend\_ops使系统进入低功耗状态。后半部分，在系统被事件唤醒后，处理相关动作，调用平台相关的suspend\_ops恢复系统、开中断、resume device、resume console。

g）最后，调用suspend\_finish，恢复（或等待恢复）process&thread，还原console。

**4.3 suspend\_prepare**

suspend\_prepare的代码如下：

1. **static** int suspend\_prepare(suspend\_state\_t state)
2. {
3. int error;
5. **if** (need\_suspend\_ops(state) && (!suspend\_ops || !suspend\_ops->enter))
6. **return** -EPERM;
8. pm\_prepare\_console();
10. error = pm\_notifier\_call\_chain(PM\_SUSPEND\_PREPARE);
11. **if** (error)
12. **goto** Finish;
14. error = suspend\_freeze\_processes();
15. **if** (!error)
16. **return** 0;
18. suspend\_stats.failed\_freeze++;
19. dpm\_save\_failed\_step(SUSPEND\_FREEZE);
20. Finish:
21. pm\_notifier\_call\_chain(PM\_POST\_SUSPEND);
22. pm\_restore\_console();
23. **return** error;
24. }

主要工作为：

a）检查suspend\_ops是否提供了.enter回调，没有的话，返回错误。

b）调用pm\_prepare\_console，将当前console切换到一个虚拟console并重定向内核的kmsg（需要的话）。该功能称作VT switch，后面我会在稍微详细的介绍一下，但Linux控制台子系统是相当复杂的，更具体的分析，要在控制台子系统的分析文章中说明。

c）调用pm\_notifier\_call\_chain，发送suspend开始的消息（PM\_SUSPEND\_PREPARE），后面会详细描述。

d）调用suspend\_freeze\_processes，freeze用户空间进程和一些内核线程。该功能称作freezing-of-tasks，我会专门用一篇文章去分析它。本文就不再详细说明了。

e）如果freezing-of-tasks失败，调用pm\_restore\_console，将console切换回原来的console，并返回错误，以便能终止suspend。

**4.4 suspend\_devices\_and\_enter**

suspend\_devices\_and\_enter的过程较为复杂，代码实现如下：

1. int suspend\_devices\_and\_enter(suspend\_state\_t state)
2. {
3. int error;
4. **bool** wakeup = **false**;
6. **if** (need\_suspend\_ops(state) && !suspend\_ops)
7. **return** -ENOSYS;
9. trace\_machine\_suspend(state);
10. **if** (need\_suspend\_ops(state) && suspend\_ops->begin) {
11. error = suspend\_ops->begin(state);
12. **if** (error)
13. **goto** Close;
14. }
15. suspend\_console();
16. ftrace\_stop();
17. suspend\_test\_start();
18. error = dpm\_suspend\_start(PMSG\_SUSPEND);
19. **if** (error) {
20. printk(KERN\_ERR "PM: Some devices failed to suspend\n");
21. **goto** Recover\_platform;
22. }
23. suspend\_test\_finish("suspend devices");
24. **if** (suspend\_test(TEST\_DEVICES))
25. **goto** Recover\_platform;
27. **do** {
28. error = suspend\_enter(state, &wakeup);
29. } **while** (!error && !wakeup && need\_suspend\_ops(state)
30. && suspend\_ops->suspend\_again && suspend\_ops->suspend\_again());
32. Resume\_devices:
33. suspend\_test\_start();
34. dpm\_resume\_end(PMSG\_RESUME);
35. suspend\_test\_finish("resume devices");
36. ftrace\_start();
37. resume\_console();
38. Close:
39. **if** (need\_suspend\_ops(state) && suspend\_ops->end)
40. suspend\_ops->end();
41. trace\_machine\_suspend(PWR\_EVENT\_EXIT);
42. **return** error;
44. Recover\_platform:
45. **if** (need\_suspend\_ops(state) && suspend\_ops->recover)
46. suspend\_ops->recover();
47. **goto** Resume\_devices;
48. }

a）再次检查平台代码是否需要提供以及是否提供了suspend\_ops。

b）调用suspend\_ops的begin回调（有的话），通知平台代码，以便让其作相应的处理（需要的话）。可能失败，需要跳至Close处执行恢复操作（suspend\_ops->end）。

c）调用suspend\_console，挂起console。该接口由"kernel\printk.c"实现，主要是hold住一个lock，该lock会阻止其它代码访问console。

d）调用ftrace\_stop，停止ftrace功能。ftrace是一个很有意思的功能，后面再介绍。

e）调用dpm\_suspend\_start，调用所有设备的->prepare和->suspend回调函数（具体可参考“[Linux电源管理(4)\_Power Management Interface](http://www.wowotech.net/linux_kenrel/pm_interface.html)”的描述），suspend需要正常suspend的设备。suspend device可能失败，需要跳至 Recover\_platform，执行recover操作（suspend\_ops->recover）。

f）以上都是suspend前的准备工作，此时，调用suspend\_enter接口，使系统进入指定的电源状态。该接口的内容如下：

1. **static** int suspend\_enter(suspend\_state\_t state, **bool** \*wakeup)
2. {
3. int error;
5. **if** (need\_suspend\_ops(state) && suspend\_ops->prepare) {
6. error = suspend\_ops->prepare();
7. **if** (error)
8. **goto** Platform\_finish;
9. }
11. error = dpm\_suspend\_end(PMSG\_SUSPEND);
12. **if** (error) {
13. printk(KERN\_ERR "PM: Some devices failed to power down\n");
14. **goto** Platform\_finish;
15. }
17. **if** (need\_suspend\_ops(state) && suspend\_ops->prepare\_late) {
18. error = suspend\_ops->prepare\_late();
19. **if** (error)
20. **goto** Platform\_wake;
21. }
23. **if** (suspend\_test(TEST\_PLATFORM))
24. **goto** Platform\_wake;
26. /\*
27. \* PM\_SUSPEND\_FREEZE equals
28. \* frozen processes + suspended devices + idle processors.
29. \* Thus we should invoke freeze\_enter() soon after
30. \* all the devices are suspended.
31. \*/
32. **if** (state == PM\_SUSPEND\_FREEZE) {
33. freeze\_enter();
34. **goto** Platform\_wake;
35. }
37. error = disable\_nonboot\_cpus();
38. **if** (error || suspend\_test(TEST\_CPUS))
39. **goto** Enable\_cpus;
41. arch\_suspend\_disable\_irqs();
42. BUG\_ON(!irqs\_disabled());
44. error = syscore\_suspend();
45. **if** (!error) {
46. \*wakeup = pm\_wakeup\_pending();
47. **if** (!(suspend\_test(TEST\_CORE) || \*wakeup)) {
48. error = suspend\_ops->enter(state);
49. events\_check\_enabled = **false**;
50. }
51. syscore\_resume();
52. }
54. arch\_suspend\_enable\_irqs();
55. BUG\_ON(irqs\_disabled());
57. Enable\_cpus:
58. enable\_nonboot\_cpus();
60. Platform\_wake:
61. **if** (need\_suspend\_ops(state) && suspend\_ops->wake)
62. suspend\_ops->wake();
64. dpm\_resume\_start(PMSG\_RESUME);
66. Platform\_finish:
67. **if** (need\_suspend\_ops(state) && suspend\_ops->finish)
68. suspend\_ops->finish();
70. **return** error;
71. }

        f1）该接口处理完后，会通过返回值告知是否enter成功，同时通过wakeup指针，告知调用者，是否有wakeup事件发生，导致电源状态切换失败。

        f2）调用suspend\_ops的prepare回调（有的话），通知平台代码，以便让其在即将进行状态切换之时，再做一些处理（需要的话）。该回调可能失败（平台代码出现意外），失败的话，需要跳至Platform\_finish处，调用suspend\_ops的finish回调，执行恢复操作。

        f3）调用dpm\_suspend\_end，调用所有设备的->suspend\_late和->suspend\_noirq回调函数（具体可参考“[Linux电源管理(4)\_Power Management Interface](http://www.wowotech.net/linux_kenrel/pm_interface.html)”的描述），suspend late suspend设备和需要在关中断下suspend的设备。需要说明的是，这里的noirq，是通过禁止所有的中断线的形式，而不是通过关全局中断的方式。同样，该操作可能会失败，失败的话，跳至Platform\_finish处，执行恢复动作。

        f4）调用suspend\_ops的prepare\_late回调（有的话），通知平台代码，以便让其在最后关头，再做一些处理（需要的话）。该回调可能失败（平台代码出现意外），失败的话，需要跳至Platform\_wake处，调用suspend\_ops的wake回调，执行device的resume、调用suspend\_ops的finish回调，执行恢复操作。

        f5）如果是suspend to freeze，执行相应的操作，包括冻结进程、suspended devices（参数为PM\_SUSPEND\_FREEZE）、cpu进入idle。如果有任何事件使CPU从idle状态退出，跳至Platform\_wake处，执行wake操作。

        f6）调用disable\_nonboot\_cpus，禁止所有的非boot cpu。也会失败，执行恢复操作即可。

        f7）调用arch\_suspend\_disable\_irqs，关全局中断。如果无法关闭，则为bug。

        f8）调用syscore\_suspend，suspend system core。同样会失败，执行恢复操作即可。有关syscore，我会在另一篇文章中详细描述。

        f9）如果很幸运，以上操作都成功了，那么，切换吧。不过，别高兴太早，还得调用pm\_wakeup\_pending检查一下，这段时间内，是否有唤醒事件发生，如果有就要终止suspend。

        f10）如果一切顺利，调用suspend\_ops的enter回调，进行状态切换。这时，系统应该已经suspend了……

        f11）suspend过程中，唤醒事件发生，系统唤醒，该函数接着执行resume动作，并最终返回。resume动作基本上是suspend的反动作，就不再继续分析了。

        f12）或者，由于意外，suspend终止，该函数也会返回。

g）suspend\_enter返回，如果返回原因不是发生错误，且不是wakeup事件。则调用suspend\_ops的suspend\_again回调，检查是否需要再次suspend。再什么情况下要再次suspend呢？需要看具体的平台了，谁知道呢。

h）继续resume操作，resume device、start ftrace、resume console、suspend\_ops->end等等。

i）该函数返回后，表示系统已经resume。

**4.5 suspend\_finish**

比较简单：

1. **static** **void** suspend\_finish(**void**)
2. {
3. suspend\_thaw\_processes();
4. pm\_notifier\_call\_chain(PM\_POST\_SUSPEND);
5. pm\_restore\_console();
6. }

a）恢复所有的用户空间进程和内核线程。

b）发送suspend结束的通知。

c）将console切换回原来的。

**5. 重要知识点回顾**

**5.1 VT switch**

通常情况下，系统控制台模块（drivers\tty\vt\）会在suspend的过程中，重新分配一个console，并将控制台切换到该console上。然后在resume时，切换回旧的console。这就是VT switch功能。VT switch是很耗时的，因此内核提供了一些机制，控制是否使用这个功能：

1）提供一个接口函数pm\_set\_vt\_switch（drivers\tty\vt\vt\_ioctl.c），方便其它内核模块从整体上关闭或者开启VT switch功能。

2）VT switch全局开关处于开启状态时，满足如下的一种条件（可参考kernel\power\console.c相关的描述），即会使能VT switch

        a）有console driver调用pm\_vt\_switch\_required接口，显式的要求使能VT switch。PM core的console模块会把这些信息记录在一个名称为pm\_vt\_switch\_list的链表中。

       b）系统禁止在suspend的过程中suspend console（由kernel/printk.c中的console\_suspend\_enabled变量控制）。很有可能需要使用console查看suspend过程，此时为了使console不混乱，有必要进行VT switch。

       c）没有任何console driver关心是否需要VT switch，换句话说没有任何driver调用pm\_vt\_switch\_required接口要求使能或禁止VT switch功能。此时会按照旧的习惯，进行VT switch。

因此，suspend过程对console的处理分为4步：

prepare console：负责在需要VT swich时，将当前console切换到SUSPEND console。

1. int pm\_prepare\_console(**void**)
2. {
3. **if** (!pm\_vt\_switch())
4. **return** 0;
6. orig\_fgconsole = vt\_move\_to\_console(SUSPEND\_CONSOLE, 1);
7. **if** (orig\_fgconsole < 0)
8. **return** 1;
10. orig\_kmsg = vt\_kmsg\_redirect(SUSPEND\_CONSOLE);
11. **return** 0;
12. }

suspend console：挂起console，由kernel/printk.c实现，主要是hold住console用的互斥锁，使他人无法使用console。

resume console：对console解锁。

restore console：将console恢复为初始的console。

1. **void** pm\_restore\_console(**void**)
2. {
3. **if** (!pm\_vt\_switch())
4. **return**;
6. **if** (orig\_fgconsole >= 0) {
7. vt\_move\_to\_console(orig\_fgconsole, 0);
8. vt\_kmsg\_redirect(orig\_kmsg);
9. }
10. }

也许，您会问，why VT switch？先留着这个疑问吧，等到分析控制台时再回答。

**5.2 freezing of task**

进程的freezing功能，是suspend、hibernate等电源管理功能的组成部分，在新版本内核中，它被独立出来，作为一个独立的电源管理状态（freeze）。该功能的目的，是在电源管理的状态切换过程中，确保所有用户空间进程和部分内核线程处于一个稳定的状态。有关该功能的具体描述，请参考wowotech后续的文章。

**5.3 PM notifier**

PM notifier是基于内核blocking notifier功能实现的。blocking notifier提供了一种kernel内部的消息通知机制，消息接受者通过notifier注册的方式，注册一个回调函数，关注消息发送者发出的notifier。当消息产生时，消息产生者通过调用回调函数的形式，通知消息接受者。这种调用，是可以被阻塞的，因此称作blocking notifier。

那suspend功能为什么使用notifier呢？原因可能有多种，这里我举一个例子，这是我们日常开发中可能会遇到的。

由之前的描述可知，suspend过程中，suspend device发生在进程被freeze之后，resume device发生在进程被恢复之前。那么：

1）如果有些设备就需要在freeze进程之前suspend怎么办？

2）如果有些设备的resume动作需要较多延时，或者要等待什么事情发生，那么如果它的resume动作发生在进程恢复之前，岂不是要阻止所有进程的恢复？更甚者，如果该设备要等待某个进程的数据才能resume，怎么办？

再来看suspend\_prepare和suspend\_finish中的处理：

1. **static** int suspend\_prepare(suspend\_state\_t state) {
2. …
3. error = pm\_notifier\_call\_chain(PM\_SUSPEND\_PREPARE);
4. **if** (error)
5. **goto** Finish;
7. error = suspend\_freeze\_processes();
8. …
9. }
11. **static** **void** suspend\_finish(**void**)
12. {
13. suspend\_thaw\_processes();
14. pm\_notifier\_call\_chain(PM\_POST\_SUSPEND);
15. pm\_restore\_console();
16. }

原来PM notifier是在设备模型的框架外，开了一个后门，那些比较特殊的driver，可以绕过设备模型，直接接收PM发送的suspend信息，以便执行自身的suspend动作。特别是resume时，可以在其它进程都正好工作的时候，只让suspend进程等待driver的resume。

感兴趣的读者，可以围观一下下面这个活生生的例子（顺便提一下，好的设计是不应该有例外的）：

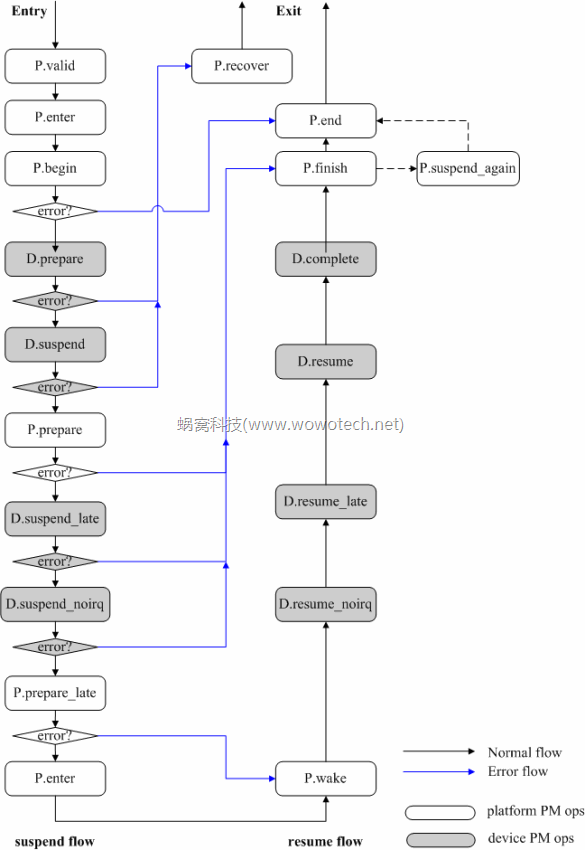
drivers\video\omap2\dss\core.c

**5.4 device PM ops 和platform PM ops的调用时机**

对Linux驱动工程师来说，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。但现实太复杂了，以至于kernel提供的这两个数据结构也很复杂，再回忆一下，如下：

1. **struct** dev\_pm\_ops {
2. int (\*prepare)(**struct** device \*dev);
3. **void** (\*complete)(**struct** device \*dev);
4. int (\*suspend)(**struct** device \*dev);
5. int (\*resume)(**struct** device \*dev);
6. int (\*freeze)(**struct** device \*dev);
7. int (\*thaw)(**struct** device \*dev);
8. int (\*poweroff)(**struct** device \*dev);
9. int (\*restore)(**struct** device \*dev);
10. int (\*suspend\_late)(**struct** device \*dev);
11. int (\*resume\_early)(**struct** device \*dev);
12. int (\*freeze\_late)(**struct** device \*dev);
13. int (\*thaw\_early)(**struct** device \*dev);
14. int (\*poweroff\_late)(**struct** device \*dev);
15. int (\*restore\_early)(**struct** device \*dev);
16. int (\*suspend\_noirq)(**struct** device \*dev);
17. int (\*resume\_noirq)(**struct** device \*dev);
18. int (\*freeze\_noirq)(**struct** device \*dev);
19. int (\*thaw\_noirq)(**struct** device \*dev);
20. int (\*poweroff\_noirq)(**struct** device \*dev);
21. int (\*restore\_noirq)(**struct** device \*dev);
22. int (\*runtime\_suspend)(**struct** device \*dev);
23. int (\*runtime\_resume)(**struct** device \*dev);
24. int (\*runtime\_idle)(**struct** device \*dev);
25. };
27. **struct** platform\_suspend\_ops {
28. int (\*valid)(suspend\_state\_t state);
29. int (\*begin)(suspend\_state\_t state);
30. int (\*prepare)(**void**);
31. int (\*prepare\_late)(**void**);
32. int (\*enter)(suspend\_state\_t state);
33. **void** (\*wake)(**void**);
34. **void** (\*finish)(**void**);
35. **bool** (\*suspend\_again)(**void**);
36. **void** (\*end)(**void**);
37. **void** (\*recover)(**void**);
38. };

虽然内核的注释已经相当详细了，但我们一定会犯晕，到底该实现哪些回调？这些回调的应用场景又是什么？蜗蜗以为，要熟练使用这些回调，唯一的方法就是多coding、多理解。除此之外，我们可以总结一下在电源状态切换时，这些回调的调用时机，从侧面帮助理解。如下（只介绍和suspend功能有关的，struct dev\_pm\_ops简称D，struct platform\_suspend\_ops简称P）：

[](http://www.wowotech.net/content/uploadfile/201408/63ee3ab38963489e40787f14378fb39620140822135344.gif)

**5.5 suspend过程的同步和PM wakeup**

最重要的事情，如果suspend的过程中，有唤醒事件产生怎么办？正常的流程，应该终止suspend，返回并处理事件。但由于suspend过程的特殊性，进程被freeze、关中断等等，导致事情并没有那么简单，以至于在很久的一段时间内，kernel都不能很好的处理。这也称作suspend过程的同步问题。

在美好的旧时光里，suspend大多用于热关机，因此同步问题的影响并不突出(因为操作并不频繁)。但来到新时代之后，事情变了，Android竟然用suspend作日常的待机（操作就相当频繁了），这时问题就大了。那怎么解决呢？得靠system wakeup framework，也就是suspend过程中所调用的pm\_wakeup\_pending接口所在的模块。我会在下一篇文章中继续该模块的分析，这里就不再继续了。

*原创文章，转发请注明出处。蜗窝科技，*[*www.wowotech.net*](http://www.wowotech.net/linux_kenrel/suspend_and_resume.html)*。*

标签: [Linux](http://www.wowotech.net/tag/Linux) [内核](http://www.wowotech.net/tag/%E5%86%85%E6%A0%B8) [suspend](http://www.wowotech.net/tag/suspend) [电源管理](http://www.wowotech.net/tag/%E7%94%B5%E6%BA%90%E7%AE%A1%E7%90%86) [resume](http://www.wowotech.net/tag/resume)

[](http://www.wowotech.net/support_us.html)

« [linux kernel的中断子系统之（三）：IRQ number和中断描述符](http://www.wowotech.net/irq_subsystem/interrupt_descriptor.html) | [Linux kernel的中断子系统之（二）：IRQ Domain介绍](http://www.wowotech.net/irq_subsystem/irq-domain.html)»

**评论：**

**heng**  
2016-08-18 15:16

谢谢wowo的分析，我想请问wowo：   
我在rk3288设置了USB唤醒功能，然后蓝牙USB接收到按键就会唤醒，这个功能我已经实现，但是其中有疑问的地方就是：到底是先蓝牙收到，然后触发USB的中断，然后唤醒，还是先触发USB中断，然后唤醒，然后才到蓝牙？

[回复](http://www.wowotech.net/linux_kenrel/suspend_and_resume.html#comment-4418)

**[wowo](http://www.wowotech.net/" \t "_blank)**  
2016-08-18 18:22

@heng：你的场景应该是这样的吧：   
蓝牙键盘\_\_\_\_\_/  ~~  \\_\_\_\_USB蓝牙适配器<-------USB------>RK3288   
抛开键盘不说，你的环境里面其实有两个大脑：适配器和RK3288.   
因此你所说的唤醒，其实是RK3288的唤醒，也就是说，RK3288在睡觉的时候，蓝牙适配器其实是醒着的。   
  
所以你觉得应该是怎么样的顺序呢？

[回复](http://www.wowotech.net/linux_kenrel/suspend_and_resume.html#comment-4419)

**heng**  
2016-08-19 08:32

@wowo：哦～这下清楚了～就是先到蓝牙的。原来蓝牙适配器是醒着的，我一直以为蓝牙也休眠了呢。谢谢wowo。那这个是不是就像android的联网，手机虽然休眠了，但是联网功能还正常在，然后消息来了就会唤醒？

[回复](http://www.wowotech.net/linux_kenrel/suspend_and_resume.html#comment-4420)

**[wowo](http://www.wowotech.net/" \t "_blank)**  
2016-08-19 09:23

@heng：是的。当然，蓝牙模块也有一个完整的系统，它内部也有可能有睡觉的东西，但这是微观层面的事情了，从3288的角度看，可以忽略了。

[回复](http://www.wowotech.net/linux_kenrel/suspend_and_resume.html#comment-4421)

**heng**  
2016-08-19 11:01

@wowo：明白了！谢谢wowo

[回复](http://www.wowotech.net/linux_kenrel/suspend_and_resume.html#comment-4423)

**junjun**  
2016-08-11 18:38

linux kernel documen文档devices.txt里有说"The suspend\_noirq phase occurs after IRQ handlers have been disabled, "， 没有明白这里的IRQ handler怎么对应到我的platform\_device上，我现在platform\_driver的suspned\_noirq会被调用到，但并没有相关的IRQ啊，当然也没有IRQ handler disable。求博主解惑啊，暂时还没有能力研究PM内部的代码。

[回复](http://www.wowotech.net/linux_kenrel/suspend_and_resume.html#comment-4389)

**[wowo](http://www.wowotech.net/" \t "_blank)**  
2016-08-11 19:18

@junjun：它的意思是，当调到各个driver的suspend\_noirq时，系统的irq已经被disable了，你可以放心的做你想做的事情。   
当然，什么事情需要在中断被disable的时候做，只有这个driver自己知道了。

[回复](http://www.wowotech.net/linux_kenrel/suspend_and_resume.html#comment-4390)

**lamaboy**  
2016-08-15 22:01

@wowo：你好，， wowo  ，看 你上面的讨论，， 我正在看最新的源码  4.5 ，发现是 suspend\_noirq  ，调用之后才禁止irq，，这样，， 不是就把逻辑改变了吗，还是我没有注意到呀，，   
代码如下；   
  
  
static int suspend\_enter(suspend\_state\_t state, bool \*wakeup)   
{   
    int error;   
  
    error = platform\_suspend\_prepare(state);   
    if (error)   
        goto Platform\_finish;   
  
    error = dpm\_suspend\_late(PMSG\_SUSPEND);   
    if (error) {   
        printk(KERN\_ERR "PM: late suspend of devices failed\n");   
        goto Platform\_finish;   
    }   
    error = platform\_suspend\_prepare\_late(state);   
    if (error)   
        goto Devices\_early\_resume;   
  
    error = dpm\_suspend\_noirq(PMSG\_SUSPEND);   
    if (error) {   
        printk(KERN\_ERR "PM: noirq suspend of devices failed\n");   
        goto Platform\_early\_resume;   
    }   
    error = platform\_suspend\_prepare\_noirq(state);   
    if (error)   
        goto Platform\_wake;   
  
    if (suspend\_test(TEST\_PLATFORM))   
        goto Platform\_wake;   
  
    /\*   
     \* PM\_SUSPEND\_FREEZE equals   
     \* frozen processes + suspended devices + idle processors.   
     \* Thus we should invoke freeze\_enter() soon after   
     \* all the devices are suspended.   
     \*/   
    if (state == PM\_SUSPEND\_FREEZE) {   
        trace\_suspend\_resume(TPS("machine\_suspend"), state, true);   
        freeze\_enter();   
        trace\_suspend\_resume(TPS("machine\_suspend"), state, false);   
        goto Platform\_wake;   
    }   
  
    error = disable\_nonboot\_cpus();   
    if (error || suspend\_test(TEST\_CPUS))   
        goto Enable\_cpus;   
  
    arch\_suspend\_disable\_irqs();   
    BUG\_ON(!irqs\_disabled());   
  
    error = syscore\_suspend();   
    if (!error) {   
        \*wakeup = pm\_wakeup\_pending();   
        if (!(suspend\_test(TEST\_CORE) || \*wakeup)) {   
            trace\_suspend\_resume(TPS("machine\_suspend"),   
                state, true);   
            error = suspend\_ops->enter(state);   
            trace\_suspend\_resume(TPS("machine\_suspend"),   
                state, false);   
            events\_check\_enabled = false;   
        } else if (\*wakeup) {   
            error = -EBUSY;   
        }   
        syscore\_resume();   
    }   
  
    arch\_suspend\_enable\_irqs();   
    BUG\_ON(irqs\_disabled());   
  
Enable\_cpus:   
    enable\_nonboot\_cpus();   
  
Platform\_wake:   
    platform\_resume\_noirq(state);   
    dpm\_resume\_noirq(PMSG\_RESUME);   
  
Platform\_early\_resume:   
platform\_resume\_early(state);   
  
Devices\_early\_resume:   
    dpm\_resume\_early(PMSG\_RESUME);   
  
Platform\_finish:   
    platform\_resume\_finish(state);   
    return error;   
}

[回复](http://www.wowotech.net/linux_kenrel/suspend_and_resume.html#comment-4405)

**[wowo](http://www.wowotech.net/" \t "_blank)**  
2016-08-15 22:19

@lamaboy：抱歉，你是对的（顺便想楼上的那么老兄道歉），我表达的太随意。   
这里的suspend\_noirq，其实是针对当前被suspend的这个设备（而不是系统），因此这里的noirq，是指设备已经没有产生irq的能力了（因为已经先调过suspend了）。

[回复](http://www.wowotech.net/linux_kenrel/suspend_and_resume.html#comment-4406)

**bat0301**  
2016-07-25 20:28

窝窝好，请教一个问题：   
  charger中断线程里面是通过i2c读寄存器，但是现在发现系统在charger中断线程执行的时候，系统正在进入休眠，中断被关了，导致读i2c失败（i2c通过中断收发）。   
  为什么中断线程刚进去，系统居然睡眠了呢？另外我试着在中断线程里面使用wake\_lock进行唤醒系统，最终却没有阻值系统的睡眠。我查了下，events\_check\_enabled居然一直是flase。   
  我想知道两个问题：1.为何中断线程执行的时候系统会睡眠？2.为何wake\_lock不起作用。

[回复](http://www.wowotech.net/linux_kenrel/suspend_and_resume.html#comment-4315)

**[wowo](http://www.wowotech.net/" \t "_blank)**  
2016-07-26 08:59

@bat0301：events\_check\_enabled为false是对的，具体可参考http://www.wowotech.net/pm\_subsystem/wakeup\_count.html。   
中断线程是在进程空间执行，可以睡眠。   
wake\_lock不起作用？你是怎么用的呢？

[回复](http://www.wowotech.net/linux_kenrel/suspend_and_resume.html#comment-4317)

**bat0301**  
2016-07-26 14:25

@wowo：谢谢窝窝的耐心解答。我在线程里面执行wake\_lock函数。我的理解是执行完后，系统不应该继续睡眠，实际却像没发生一样继续睡眠了。是不是wake\_lock的使用有问题？   
  
关于events\_check\_enabled为flase我再研究研究代码。

[回复](http://www.wowotech.net/linux_kenrel/suspend_and_resume.html#comment-4318)

**bat0301**  
2016-07-26 14:48

@bat0301：在suspend\_enter接口中，suspend前的最后一刻，会调用pm\_wakeup\_pending接口，而pm\_wakeup\_pending函数里面，如果events\_check\_enabled为flase，则直接返回0。如果有唤醒事件发生（返回非0），如果有就要终止suspend，也就是执行syscore\_resume。   
按道理，我在我的线程里面使用wake\_lock，就是产生了一个唤醒的事件，但是打印看到events\_check\_enabled全程都是0，那就不难解释为什么睡眠没有被打断了，因为睡眠过程中，pm\_wakeup\_pending一直返回0，所以根本检测不到来了唤醒事件吧？   
不知道我是wake\_lock用法不对导致的还是其他原因？

[回复](http://www.wowotech.net/linux_kenrel/suspend_and_resume.html#comment-4319)

**[wowo](http://www.wowotech.net/" \t "_blank)**  
2016-07-26 18:49

@bat0301：返回0，不会进入这个if语句：   
                \*wakeup = pm\_wakeup\_pending();                                     
                if (!(suspend\_test(TEST\_CORE) || \*wakeup)) {                       
                        error = suspend\_ops->enter(state);                         
                        events\_check\_enabled = false;                             
                }   
系统才会终止suspend。

[回复](http://www.wowotech.net/linux_kenrel/suspend_and_resume.html#comment-4322)

**js\_wawayu**  
2016-07-15 16:58

请教一个问题，   
cmdline中boot\_cpus=0,1,2,3,4,5，   
但是在系统suspend的时候如下的log：   
Disabling non-boot CPUs ...   
]psci: Retrying again to check for CPU kill   
]psci: CPU1 killed.   
psci: CPU2 killed.   
psci: CPU3 killed.   
psci: CPU4 killed.   
psci: CPU7 killed.   
按照cmdline cpu1,2,3,4都属于boot cpu为什么在suspend时候disable non-boot cpu的时候把这几个cpu也关了？boot cpu是不是只能有一个cpu0?   
谢谢！

[回复](http://www.wowotech.net/linux_kenrel/suspend_and_resume.html#comment-4267)

**[wowo](http://www.wowotech.net/" \t "_blank)**  
2016-07-16 09:06

@js\_wawayu：我找了很久，没有发现boot\_cpus的cmdline，你用的是什么系统呢？

[回复](http://www.wowotech.net/linux_kenrel/suspend_and_resume.html#comment-4268)

**js\_wawayu**  
2016-07-16 22:55

@wowo：哦，我理解错了，应该是dts里定义的，反正boot\_cpus定义了这五个cpu。

[回复](http://www.wowotech.net/linux_kenrel/suspend_and_resume.html#comment-4269)

**[wowo](http://www.wowotech.net/" \t "_blank)**  
2016-07-17 17:22

@js\_wawayu：我觉得这里的“boot\_cpus”，并不是我们所理解的那个boot cpu。具体要看你的代码逻辑了，我这边看不到:-)

[回复](http://www.wowotech.net/linux_kenrel/suspend_and_resume.html#comment-4270)

**js\_wawayu**  
2016-07-18 11:06

@wowo：smp.c里有下面的代码：   
early\_param("boot\_cpus", boot\_cpus);   
kernel-parameter.txt里有如下的解释：   
early\_param("boot\_cpus", boot\_cpus);   
  
我用的是高通平台

[回复](http://www.wowotech.net/linux_kenrel/suspend_and_resume.html#comment-4272)

**js\_wawayu**  
2016-07-18 11:07

@js\_wawayu：写错了kernel-parameter.txt里的解释如下：   
boot\_cpus=    [SMP] Rather than attempting to online all possible CPUs at   
            boot time, only online the specified set of CPUs.

[回复](http://www.wowotech.net/linux_kenrel/suspend_and_resume.html#comment-4273)

**[wowo](http://www.wowotech.net/" \t "_blank)**  
2016-07-18 13:09

@js\_wawayu：看解释，就很清楚了，不是boot/secondary的意思，

**[bt\_baseband](http://no/" \t "_blank)**  
2016-04-26 15:20

I found a type error: "CPU hotplug、wakeup处理等----过----个知识点"

[回复](http://www.wowotech.net/linux_kenrel/suspend_and_resume.html#comment-3863)

**[wowo](http://www.wowotech.net/" \t "_blank)**  
2016-04-26 16:10

@bt\_baseband：多谢提醒，稍后改正:-)

[回复](http://www.wowotech.net/linux_kenrel/suspend_and_resume.html#comment-3864)

**[firo](http://firoyang.org/" \t "_blank)**  
2015-09-05 18:40

Hi, wowo   
我现在遇到的问题, 从现象看是 在test  suspend的时候, cpu被disable后不能enable.   
操作下:   
echo processors > /sys/power/pm\_test   
echo mem  > /sys/power/mem   
log:   
Disabling non-boot CPUs ...   
CPU1: shutdown   
CPU2: shutdown   
CPU3: shutdown   
suspend debug: Waiting for 5 seconds.   
Enabling non-boot CPUs ...   
SMP: CPU 0 Waking up CPU 1   
CPU1: failed to come online   
Error taking CPU1 up: -5   
SMP: CPU 0 Waking up CPU 2   
CPU2: failed to come online   
Error taking CPU2 up: -5   
SMP: CPU 0 Waking up CPU 3   
CPU3: failed to come online   
Error taking CPU3 up: -5   
PM: noirq resume of devices complete after 80.429 msecs   
PM: early resume of devices complete after 0.409 msecs   
我追踪看到一个奇诡的现象, 不能和理解,  我的基本信息3.14内核 arm32:   
cpu0 唤醒了cpu1   
cpu1开始执行secondary\_start\_kernel开始启动自己, 调用路径:   
secondary\_start\_kernel->cpu\_startup\_entry->cpu\_idle\_loop->while(1)   
  
在这个函数里面有个语句set\_cpu\_online(cpu, true);按理说, 1号cpu就是online的状态了.   
而在cpu\_idle\_loop的while(1)的循环里面这个   
                        if (cpu\_is\_offline(smp\_processor\_id()))   
                                arch\_cpu\_idle\_dead();   
这个if语句里的cpu\_is\_offline(smp\_processor\_id())尽然是true. 也就是说1号cpu是offline的!   
wowo你认为这个现象正常吗?我没找到相关修改cpu 状态的函数逻辑.我的问题可能和这个有关.

[回复](http://www.wowotech.net/linux_kenrel/suspend_and_resume.html#comment-2531)

**[firo](http://firoyang.org/" \t "_blank)**  
2015-09-05 18:45

@firo：还有Error taking CPU1 up: -5 这个错误输出是因为相应的secondary  cpu的状态是offline打出来的, 这个错误输出是cpu0打出来的, 具体函数在enable\_nonboot\_cpus.

[回复](http://www.wowotech.net/linux_kenrel/suspend_and_resume.html#comment-2532)

**wowo**  
2015-09-05 20:57

@firo：firo,您确认这句话成立吗：“cpu1开始执行secondary\_start\_kernel开始启动自己”？   
CPU3: failed to come online ，是\_\_cpu\_up打印的。   
Error taking CPU3 up: -5 ，是enable\_nonboot\_cpus打印的。   
没有证据显示secondary\_start\_kernel执行了。   
我怀疑是suspend的过程中，secondary CPUs的执行环境被破坏了，没有机会执行。

[回复](http://www.wowotech.net/linux_kenrel/suspend_and_resume.html#comment-2533)

**[firo](http://firoyang.org/" \t "_blank)**  
2015-09-05 21:19

@wowo：你的推理很可能是正确的!   
  
我之前在cpu\_die这个函数里面加了句dump\_stack,   
打出的bt栈底是secondary\_start\_kernel, 我以为secondary CPUs正常启动了.   
实际是disable\_nonboot\_cpu处罚了secondary CPUs在idle\_cpu\_loop里面的cpu\_die的执行, 从而输出了bt.   
  
看这样子还是secondary CPUs没有正确跳转到初始化的代码上!   
  
我今天,  看代码发现arch/arm/mach-mybsp/platsmp.c的代码里面竟然没有正确的初始化smp\_ops.cpu\_kill这个成员.  我是从内核3.2直接把这个suspend和idle功能迁移到内核3.14上的. 内核3.2的代码里面没有smp\_operations这个结构(我没找到).cpu die, cpu kill这些功能在3.2的内核都是通过在mach-\*/下面定义platform\_cpu\_die/kill这样的全局函数, 从而在被smp子系统调用的.   
  
另外, 我看到http://events.linuxfoundation.org/sites/events/files/slides/clement-smp-bring-up-on-arm-soc.pdf 这个doc里面说cpu  kill 和 cpu die之间要同步. 我怀疑可能和这个有关, 不过感觉概率不大.   
  
至于你说的"secondary CPUs的执行环境被破坏了" 这种可能性很大.   
我追踪过CPU0 调用smp\_boot\_secondary平台相关的代码都被执行了.   
这些代码在3.2是好用的, 另外代码的逻辑非常简单:   
1. smp cross call 发IPI  wakeup唤醒 相应的secondary CPUs   
2. 设置secondary CPUs 的boot  address.   
3. 清除 secondary CPUs的reset 标志位.   
  
不知道哪里能破坏secondary CPUs的执行环境.wowo有没有可能的怀疑点?

[回复](http://www.wowotech.net/linux_kenrel/suspend_and_resume.html#comment-2534)

**[firo](http://firoyang.org/" \t "_blank)**  
2015-09-05 21:26

@firo：我很可能是因为disable nonboot cpus这个路径上包含了一些从3.2移过来的代码导致的.   
就多了这两句,   
#ifdef CONFIG\_SHEEVA\_DEEP\_IDLE   
        armadaxp\_fabric\_prepare\_deepIdle();   
        armadaxp\_fabric\_prepare\_hotplug();   
#endif   
这个CONFIG\_SHEEVA\_DEEP\_IDLE配置项被打开了.

[回复](http://www.wowotech.net/linux_kenrel/suspend_and_resume.html#comment-2535)

**[wowo](http://www.wowotech.net/" \t "_blank)**  
2015-09-06 09:53

@firo：可以从两个方面入手：secondary是否真的被唤醒？boot Address是否正确？Address里面的内容是否正确？   
因为boot CPU在醒着，这些东西还是很好查的。

[回复](http://www.wowotech.net/linux_kenrel/suspend_and_resume.html#comment-2536)

**[firo](http://firoyang.org/" \t "_blank)**  
2015-09-06 10:14

@wowo：Thanks!   
  
我试着排查一下.

[回复](http://www.wowotech.net/linux_kenrel/suspend_and_resume.html#comment-2537)

**[firo](http://firoyang.org/" \t "_blank)**  
2015-09-06 17:26

@wowo：Hi, wowo,   
阶段性的进展:   
调试发现在smp\_boot\_secondary这个函数的步骤3   
  
清除 secondary CPUs的reset 标志位.   
  
之前Secondary CPUs的reset标志位就是没置位的!   
我怀疑问题出在disable\_nonboot\_cpus 和 cpu\_die里面.   
追到了wfi, 也没看到相关的设置reset标志位的调用.   
  
另外, 查看硬件手册, 有上这样的描述:   
Place the CPU to Wait For Interrupt mode. When the CPU enters this state, the following steps   
occur:   
a) CPU activity is halted, outstanding instructions are completed, and the write buffer is drained   
out.   
b) The CPU clocks are halted.   
c) An internal reset is generated to the CPU.   
d) The PMU power down the CPU core.   
  
貌似是硬件, 来设置这个reset标志位.

[回复](http://www.wowotech.net/linux_kenrel/suspend_and_resume.html#comment-2539)

**[wowo](http://www.wowotech.net/" \t "_blank)**  
2015-09-06 18:36

@firo：这部分已经是arch-dependent的了，取决于您所使用的平台的特性。cpu die和cpu boot的实现，要相互配合。另外，可以检查一下，cpu die中是否有主动关clock、关power？

[回复](http://www.wowotech.net/linux_kenrel/suspend_and_resume.html#comment-2540)

**[firo](http://firoyang.org/" \t "_blank)**  
2015-09-06 18:45

@wowo：die的实现:   
1. 设置唤醒事件, 和在可能的情况下power down l2   
2. disable  cpu snoop, 这个貌似和cpu的coherency有关.   
3. 进入wfi

**[firo](http://firoyang.org/" \t "_blank)**  
2015-09-08 17:32

@wowo：原因找到了, 因为kernel menuconfig 的原因导致一段汇编代码没编译.   
哎....

**[firo](http://firoyang.org/" \t "_blank)**  
2015-08-30 13:03

代码分布   
2）Device PM   
加上   
include/linux/pm.h   
如何?   
  
另外, 这句   
a）调用suspend\_ops的enter回调（有的话），通知平台代码，以便让其作相应的准备（需要的话）   
和代码语义不符.   
if (need\_suspend\_ops(state) && (!suspend\_ops || !suspend\_ops->enter))   
                return -EPERM;   
我理解这里只是做一个检测, 没有调用.

[回复](http://www.wowotech.net/linux_kenrel/suspend_and_resume.html#comment-2509)

**wowo**  
2015-08-30 19:07

@firo：多谢指正，确实如此，等等修改。谢谢

[回复](http://www.wowotech.net/linux_kenrel/suspend_and_resume.html#comment-2510)

**[firo](http://firoyang.org/" \t "_blank)**  
2015-08-30 00:52

非常赞! 思路清晰, 语言通俗, 内容全面!!!   
脑浆炸裂一般爽啊!   
从此楼主博客是我家!!!

[回复](http://www.wowotech.net/linux_kenrel/suspend_and_resume.html#comment-2505)

**wowo**  
2015-08-30 07:06

@firo：多谢夸奖~

[回复](http://www.wowotech.net/linux_kenrel/suspend_and_resume.html#comment-2508)

**[linuxer](http://www.wowotech.net/" \t "_blank)**  
2015-07-20 19:05

有一个小瑕疵，文档中描述“f）调用suspend\_test，判断是否为suspend to freeze，如果是，则已经完成，直接跳至h步骤。”，看起来这里表述可能有问题，对于suspend过程，如果系统要进入的电源管理状态是PM\_SUSPEND\_FREEZE的话，还是需要执行suspend\_devices\_and\_enter的（步骤g）。   
  
对于PM\_SUSPEND\_FREEZE，其动作包括三个部分：   
1、冻结进程   
2、suspended devices（电源管理状态的参数是PM\_SUSPEND\_FREEZE）   
3、cpu进入idle

[回复](http://www.wowotech.net/linux_kenrel/suspend_and_resume.html#comment-2252)

**[wowo](http://www.wowotech.net/" \t "_blank)**  
2015-07-20 19:58

@linuxer：大神怎么有闲情雅致看电源管理了？哈哈。是的，被suspend\_test误导了，还不够细致啊，等下改改。

[回复](http://www.wowotech.net/linux_kenrel/suspend_and_resume.html#comment-2253)

**[linuxer](http://www.wowotech.net/" \t "_blank)**  
2015-07-20 23:35

@wowo：项目中有一个issue和freezer相关，因此看的比较仔细，哈哈   
另外，我马上要写中断子系统中和电源管理子系统相关的内容，因此也在阅读电源管理的代码

[回复](http://www.wowotech.net/linux_kenrel/suspend_and_resume.html#comment-2256)

**[heziq](http://www.wowotech.net/" \t "_blank)**  
2015-07-14 18:45

@wowo   
@linuxer   
  
struct device\_driver {   
    const char        \*name;   
    struct bus\_type        \*bus;   
  
    struct module        \*owner;   
    const char        \*mod\_name;    /\* used for built-in modules \*/   
  
    bool suppress\_bind\_attrs;    /\* disables bind/unbind via sysfs \*/   
  
    const struct of\_device\_id    \*of\_match\_table;   
    const struct acpi\_device\_id    \*acpi\_match\_table;   
  
    int (\*probe) (struct device \*dev);   
    int (\*remove) (struct device \*dev);   
    void (\*shutdown) (struct device \*dev);   
    int (\*suspend) (struct device \*dev, pm\_message\_t state);   
    int (\*resume) (struct device \*dev);   
    const struct attribute\_group \*\*groups;   
  
    const struct dev\_pm\_ops \*pm;   
  
    struct driver\_private \*p;   
};   
在这里面有    int (\*suspend) (struct device \*dev, pm\_message\_t state);   
    int (\*resume) (struct device \*dev);   
但是我在内核中没有找到使用他的地方，一般情况下使用的是struct dev\_pm\_ops 中的suspend和resume。既然没有使用，为什么还留着他呢？   
  
在休眠过程中的调用   
    if (!callback && dev->driver && dev->driver->pm) {   
        info = "driver ";   
        callback = pm\_op(dev->driver->pm, state);   
    }   
很明显调用的是dev->driver->pm，而没有调用dev->driver->suspend.dev是struct dev结构体   
  
在这个Ac97\_bus.c 文件中好像用了一次dev->driver->suspend

[回复](http://www.wowotech.net/linux_kenrel/suspend_and_resume.html#comment-2212)

**[wowo](http://www.wowotech.net/" \t "_blank)**  
2015-07-14 21:00

@heziq：driver->suspend是旧接口，现在很少使用了。保留的目的是向前兼容。

[回复](http://www.wowotech.net/linux_kenrel/suspend_and_resume.html#comment-2213)

1 [2](http://www.wowotech.net/pm_subsystem/suspend_and_resume.html/comment-page-2#comments) [3](http://www.wowotech.net/pm_subsystem/suspend_and_resume.html/comment-page-3#comments)

**发表评论：**

窗体顶端

 昵称

 邮件地址 (选填)

 个人主页 (选填)



http://www.wowotech.net/include/lib/checkcode.php 

窗体底端

* **站内搜索**

窗体顶端



窗体底端

* **功能**

[留言板](http://www.wowotech.net/message_board.html)[评论列表](http://www.wowotech.net/?plugin=commentlist)

* **随机文章**
  + [linux kernel的中断子系统之（九）：tasklet](http://www.wowotech.net/irq_subsystem/tasklet.html)
  + [从“码农”说起](http://www.wowotech.net/tech_discuss/111.html)
  + [显示技术介绍(1)\_概述](http://www.wowotech.net/display/display_tech_overview.html)
  + [显示技术介绍(3)\_CRT技术](http://www.wowotech.net/display/crt_intro.html)
  + [Linux 2.5.43版本的RCU实现（废弃）](http://www.wowotech.net/kernel_synchronization/Linux-2-5-43-RCU.html)
* **最新评论**
  + 123@   
    [赞](http://www.wowotech.net/linux_kenrel/kernel_config_overview.html#4433)
  + MOLLY   
    [10元小意思，买瓶饮料，慢慢coding](http://www.wowotech.net/support_us.html#4432)
  + wowo   
    [@jackzhu：你说的太好了，希望热爱技术的大家，能够齐心...](http://www.wowotech.net/about.html#4431)
  + jackzhu   
    [的确在研究LINUX的过程中，我也同样感觉到了问题的所在，实...](http://www.wowotech.net/about.html#4430)
  + Asker   
    [Hi Linuxer, 请教下, 您上边讲的"sleep系...](http://www.wowotech.net/timer_subsystem/timer_subsystem_userspace.html#4429)
* **文章分类**
  + [Linux内核分析(7)](http://www.wowotech.net/sort/linux_kenrel) [订阅该分类](http://www.wowotech.net/rss.php?sort=4)
    - [统一设备模型(14)](http://www.wowotech.net/sort/device_model) [订阅该分类](http://www.wowotech.net/rss.php?sort=12)
    - [电源管理子系统(40)](http://www.wowotech.net/sort/pm_subsystem) [订阅该分类](http://www.wowotech.net/rss.php?sort=13)
    - [中断子系统(14)](http://www.wowotech.net/sort/irq_subsystem) [订阅该分类](http://www.wowotech.net/rss.php?sort=14)
    - [进程管理(5)](http://www.wowotech.net/sort/process_management) [订阅该分类](http://www.wowotech.net/rss.php?sort=15)
    - [内核同步机制(17)](http://www.wowotech.net/sort/kernel_synchronization) [订阅该分类](http://www.wowotech.net/rss.php?sort=16)
    - [GPIO子系统(3)](http://www.wowotech.net/sort/gpio_subsystem) [订阅该分类](http://www.wowotech.net/rss.php?sort=17)
    - [时间子系统(13)](http://www.wowotech.net/sort/timer_subsystem) [订阅该分类](http://www.wowotech.net/rss.php?sort=18)
    - [通信类协议(4)](http://www.wowotech.net/sort/comm) [订阅该分类](http://www.wowotech.net/rss.php?sort=20)
    - [内存管理(6)](http://www.wowotech.net/sort/memory_management) [订阅该分类](http://www.wowotech.net/rss.php?sort=21)
    - [图形子系统(1)](http://www.wowotech.net/sort/graphic_subsystem) [订阅该分类](http://www.wowotech.net/rss.php?sort=23)
    - [文件系统(0)](http://www.wowotech.net/sort/filesystem) [订阅该分类](http://www.wowotech.net/rss.php?sort=26)
  + [u-boot分析(2)](http://www.wowotech.net/sort/u-boot) [订阅该分类](http://www.wowotech.net/rss.php?sort=25)
  + [Linux应用技巧(11)](http://www.wowotech.net/sort/linux_application) [订阅该分类](http://www.wowotech.net/rss.php?sort=3)
  + [软件开发(6)](http://www.wowotech.net/sort/soft) [订阅该分类](http://www.wowotech.net/rss.php?sort=1)
  + [基础技术(6)](http://www.wowotech.net/sort/basic_tech) [订阅该分类](http://www.wowotech.net/rss.php?sort=6)
    - [蓝牙(10)](http://www.wowotech.net/sort/bluetooth) [订阅该分类](http://www.wowotech.net/rss.php?sort=10)
    - [ARMv8A Arch(12)](http://www.wowotech.net/sort/armv8a_arch) [订阅该分类](http://www.wowotech.net/rss.php?sort=19)
    - [显示(3)](http://www.wowotech.net/sort/display) [订阅该分类](http://www.wowotech.net/rss.php?sort=22)
  + [基础学科(9)](http://www.wowotech.net/sort/basic_subject) [订阅该分类](http://www.wowotech.net/rss.php?sort=7)
  + [技术漫谈(10)](http://www.wowotech.net/sort/tech_discuss) [订阅该分类](http://www.wowotech.net/rss.php?sort=8)
  + [项目专区(0)](http://www.wowotech.net/sort/project) [订阅该分类](http://www.wowotech.net/rss.php?sort=9)
    - [X Project(10)](http://www.wowotech.net/sort/x_project) [订阅该分类](http://www.wowotech.net/rss.php?sort=24)
* **文章存档**
  + [2016年8月(4)](http://www.wowotech.net/record/201608)
  + [2016年7月(5)](http://www.wowotech.net/record/201607)
  + [2016年6月(10)](http://www.wowotech.net/record/201606)
  + [2016年5月(8)](http://www.wowotech.net/record/201605)
  + [2016年4月(7)](http://www.wowotech.net/record/201604)
  + [2016年3月(5)](http://www.wowotech.net/record/201603)
  + [2016年2月(5)](http://www.wowotech.net/record/201602)
  + [2016年1月(6)](http://www.wowotech.net/record/201601)
  + [2015年12月(6)](http://www.wowotech.net/record/201512)
  + [2015年11月(9)](http://www.wowotech.net/record/201511)
  + [2015年10月(9)](http://www.wowotech.net/record/201510)
  + [2015年9月(4)](http://www.wowotech.net/record/201509)
  + [2015年8月(3)](http://www.wowotech.net/record/201508)
  + [2015年7月(7)](http://www.wowotech.net/record/201507)
  + [2015年6月(3)](http://www.wowotech.net/record/201506)
  + [2015年5月(7)](http://www.wowotech.net/record/201505)
  + [2015年4月(9)](http://www.wowotech.net/record/201504)
  + [2015年3月(9)](http://www.wowotech.net/record/201503)
  + [2015年2月(6)](http://www.wowotech.net/record/201502)
  + [2015年1月(6)](http://www.wowotech.net/record/201501)
  + [2014年12月(17)](http://www.wowotech.net/record/201412)
  + [2014年11月(8)](http://www.wowotech.net/record/201411)
  + [2014年10月(9)](http://www.wowotech.net/record/201410)
  + [2014年9月(7)](http://www.wowotech.net/record/201409)
  + [2014年8月(12)](http://www.wowotech.net/record/201408)
  + [2014年7月(6)](http://www.wowotech.net/record/201407)
  + [2014年6月(6)](http://www.wowotech.net/record/201406)
  + [2014年5月(9)](http://www.wowotech.net/record/201405)
  + [2014年4月(9)](http://www.wowotech.net/record/201404)
  + [2014年3月(7)](http://www.wowotech.net/record/201403)
  + [2014年2月(3)](http://www.wowotech.net/record/201402)
  + [2014年1月(4)](http://www.wowotech.net/record/201401)

[订阅Rss](http://www.wowotech.net/rss.php)

Copyright @ 2013-2015 [蜗窝科技](http://www.wowotech.net/) All rights reserved. Powered by [emlog](http://www.emlog.net/)