|  |
| --- |
| [Linux睡眠唤醒机制--Kernel态](http://blog.csdn.net/myarrow/article/details/8136109) |
|  |
| [Android/linux(earlysuspend、lateresume)睡眠唤醒机制简介](http://blog.sina.com.cn/s/blog_759dc36b0100stax.html) |

**在Linux中,休眠主要分三个主要的步骤:**   1) 冻结用户态进程和内核态任务  
   2) 调用注册的设备的suspend的回调函数, 顺序是按照注册顺序  
   3) 休眠核心设备和使CPU进入休眠态

主要流程如下:

   # echo mem > /sys/power/state  
   命令系统进入休眠. 也可以使用

   # cat /sys/power/state  
   来得到内核支持哪几种休眠方式.

用户对于/sys/power/state 的读写会调用到 kernel/kernel/power/main.c中的**state\_store()**， 用户可以写入 const char \* const pm\_states[] 中定义的字符串， 比如"mem", "standby"。

const char \*const pm\_states[PM\_SUSPEND\_MAX] = {  
#ifdef CONFIG\_EARLYSUSPEND  
 [PM\_SUSPEND\_ON]  = "on",  
#endif  
 [PM\_SUSPEND\_STANDBY] = "standby",  
 [PM\_SUSPEND\_MEM] = "mem",  
};

 然后state\_store()会调用

**request\_suspend\_state(state);或者****enter\_state()**<注：这是经典Linux调用流程， 在Android系统中，Kernel将调用request\_suspend\_state，而不是enter\_state>，

#ifdef CONFIG\_EARLYSUSPEND

if (state == PM\_SUSPEND\_ON || valid\_state(state)) {

error = 0;

**request\_suspend\_state(state);**

}

#else

error = **enter\_state(state);**

#endif

如果CONFIG\_EARLYSUSPEND宏开的话，kernel会先走earlysuspend，反之则直接走suspend；从这里开始就要分两个分支了，如果支持earlysuspend的话就进入 **request\_suspend\_state(state)**函数，如果不支持的话就进入标准Linux的**enter\_state(state)**函数。

**1:如果进入request\_suspend\_state(state);**

# [earlysuspend调用过程](http://blog.csdn.net/chen198746/article/details/8959434)

Early Suspend和Late Resume是 Android在标准Linux的基础上增加的一项特性。当用户空间的向内核请求进入suspend时，这时候会先进入early suspend状态，驱动程序可以注册early suspend的回调函数，当进入该状态时，内核会逐一地调用这些回调函数。例如显示屏的驱动程序通常会注册early suspend，在他的回调函数中，驱动程序会把屏幕和背光都关闭。在这种状态下，所有的后台进程都还在活动中，该播放歌曲的播放歌曲，该下载数据的依然 在下载，只是显示屏不良而已。进入early suspend状态以后，一旦所有的电源锁（wake lock）被释放，系统马上会进入真正的suspend流程，直到最后系统停止工作，等待外部事件的唤醒。

调用request\_suspend\_state()函数，不过在调用该函数之前会先valid\_state()一下，这给了平台相关的代码一个机会确认该平台是否支持所请求的电源状态。

还记得前面初始化时建立的工作队列suspend\_woek\_queue吗？根 据之前的电源状态和请求的状态， request\_suspend\_state()只是简单地向suspend\_work\_queue中加入early\_suspend\_work或者是 late\_resume\_work并调度他们执行。early\_suspend\_work的工作函数是early\_suspend()：

终于看到啦，early\_suspend()遍历early\_suspend\_handlers链表，从中取出各个驱动程序注册的early\_suspend结构，然后调用它的suspend回调函数。至此整个earlysuspend的流程完成。但是，这时整个系统只是处于所谓的idle状态，cpu还在工作，后台进程也在工作中，那什么时候系统会真正地进入睡眠状态？注意到最后一句关键的调用了没有：  
wake\_unlock(&main\_wake\_lock);

**2:如果进入enter\_state();**

它首先会检查一些状态参数，然后同步文件系统, enter\_state()中会调用**suspend\_prepare()**,当进入到suspend\_prepare()中以后， 它会给suspend分配一个虚拟终端来输出信息， 然后广播一个系统要进入suspend的Notify， 关闭掉用户态的helper进程， 然后一次调用**suspend\_freeze\_processes()**冻结所有的进程， 这里会保存所有进程当前的状态， 也许有一些进程会拒绝进入冻结状态， 当有这样的进程存在的时候， 会导致冻结失败,此函数就会放弃冻结进程，并且解冻刚才冻结的所有进程, suspend\_prepare()中会调用

**suspend\_devices\_and\_enter()**来把所有的外设休眠， 在这个函数中， 如果平台注册了suspend\_ops(通常是在板级定义中定义和注册，在kernel/arch/arm/mach-xx/pm.c中调用suspend\_set\_ops)， 这里就会调用 suspend\_ops->begin()； 然后调用dpm\_suspend\_start，他们会依次调用驱动的suspend() 回调来休眠掉所有的设备。

     当所有的设备休眠以后， suspend\_ops->prepare()会被调用， 这个函数通常会作一些准备工作来让板机进入休眠。 接下来Linux，在多核的CPU中的非启动CPU会被关掉，通过注释看到是避免这些其他的CPU造成race condio，接下来的以后只有一个CPU在运行了, suspend\_devices\_and\_enter()中的**suspend\_enter()**会被调用， 这个函数会关闭arch irq， 调用 **device\_power\_down()**， 它会调用**suspend\_late()**函数， 这个函数是系统真正进入休眠最后调用的函数，通常会在这个函数中作最后的检查。 如果检查没问题， 接下来休眠所有的系统设备和总线，并且调用 suspend\_pos->enter() 来使CPU进入省电状态，这时就已经休眠了。代码的执行也就停在这里了。

**三、Linux Resume流程**

       如果在休眠中系统被中断或者其他事件唤醒，接下来的代码就会开始执行，这个唤醒的顺序是和休眠的循序相反的，所以系统设备和总线会首先唤醒，使能系统中 断，使能休眠时候停止掉的非启动CPU， 以及调用suspend\_ops->finish()， 而且在suspend\_devices\_and\_enter()函数中也会继续唤醒每个设备，使能虚拟终端， 最后调用 suspend\_ops->end()。

      在返回到enter\_state()函数中的，当 suspend\_devices\_and\_enter() 返回以后，外设已经唤醒了，但是进程和任务都还是冻结状态， 这里会调用suspend\_finish()来解冻这些进程和任务， 而且发出Notify来表示系统已经从suspend状态退出， 唤醒终端。

      到这里，所有的休眠和唤醒就已经完毕了，系统继续运行了。

以下列出suspend\_devices\_and\_enter()的实现

/\*\*

\* suspend\_devices\_and\_enter - suspend devices and enter the desired system

\* sleep state.

\* @state: state to enter

\*/

int suspend\_devices\_and\_enter(suspend\_state\_t state)

{

int error;

if (!suspend\_ops)

return -ENOSYS;

trace\_machine\_suspend(state);

// 如果平台注册了suspend\_ops(通常是在板级定义中定义和注册，

// 在kernel/arch/arm/mach-xx/pm.c中调用suspend\_set\_ops)，

// 这里就会调用 suspend\_ops->begin()；

if (suspend\_ops->begin) {

error = suspend\_ops->begin(state);

if (error)

goto Close;

}

suspend\_console();

suspend\_test\_start();

// 依次调用驱动的suspend() 回调来休眠掉所有的设备。

error = dpm\_suspend\_start(PMSG\_SUSPEND);

if (error) {

printk(KERN\_ERR "PM: Some devices failed to suspend\n");

goto Recover\_platform;

}

suspend\_test\_finish("suspend devices");

if (suspend\_test(TEST\_DEVICES))

goto Recover\_platform;

// 这个函数会关闭arch irq， 调用 device\_power\_down()， 它会调用suspend\_late()函数，

// 这个函数是系统真正进入休眠最后调用的函数，通常会在这个函数中作最后的检查。

// 如果检查没问题， 接下来休眠所有的系统设备和总线，并且调用 suspend\_pos->enter()

// 来使CPU进入省电状态，这时就已经休眠了。代码的执行也就停在这里了。

error = suspend\_enter(state);

Resume\_devices:

suspend\_test\_start();

dpm\_resume\_end(PMSG\_RESUME);

suspend\_test\_finish("resume devices");

resume\_console();

Close:

if (suspend\_ops->end)

suspend\_ops->end();

trace\_machine\_suspend(PWR\_EVENT\_EXIT);

return error;

Recover\_platform:

if (suspend\_ops->recover)

suspend\_ops->recover();

goto Resume\_devices;

}