Tinalinux

功耗管理开发指南

1.0

2019.02.01

文档履历

版本号	日期	制/修订人	内容描述
1.0	2019.02.01	AWA1381	创建

目录

1. 概述

文档主要描述 Tina 功耗管理，包含 Standby 管理、standby 调试、nativepower 应用、powerkey 应用等内容。

1.1 编写目的

简要介绍 Tina 功耗管理机制。

1.2 适用范围

适用于 Allwinner Tina SDK。

1.3 适用人员

所有 Tina 平台的客户和想了解和学习 Tina 功耗管理的相关人员。

1.4 相关术语

术语	解释
Freeze	可称为 Suspend to idle，表示进程全部冻结。
Suspend	可称为 Suspend to ram，在 freeze 的基础上，让 RAM 进入自刷新模式。
Normal standby	Allwinner 内部术语：DRAM 进入自刷新，CPU0 进入 WFI。
Super standby	Allwinner 内部术语：DRAM 进入自刷新，所有 CPU（不包括 CPUS）电全部断电。
Arm Trusted Firmware	ARMv8-A 安全世界软件的一种实现，包含标准接口：PSCI、TBRR、SMCCC 等。
CPUS	小 cpu,R16/R18/R30/R40/MR133 等支持。
AXP	电源管理模块。

2. 功耗管理

2.1 背景概述

我们常说的功耗也就是功率：P=CFV²(C: 常系数，F：频率，V：电压)

而功耗管理通俗的讲就是省电，两个出发点：

能不用就不用一间歇供电和断电静态功耗。

必须用时少用一降低供电电压和频率动态功耗。

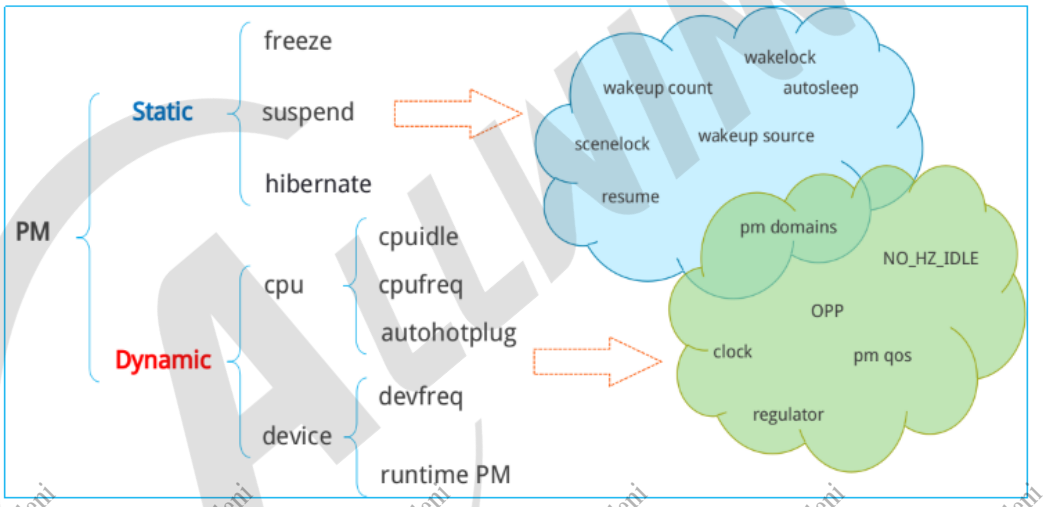

图 1: 功耗管理分类

本文档只简要介绍静态功耗，也就是常说的休眠唤醒（standby）。

2.2 standby 配置

在命令行中进入 Tina 根目录，执行命令进入配置主界面：

```
source build/envsetup.sh (见详注1)
lunch (见详注2)
make kernel_menuconfig (见详注3)
详注：
1 加载环境变量及tina提供的命令
2 输入编号，选择方案
3 进入内核配置主界面(对一个shell而言，前两个命令只需要执行一次)
```

配置路径：

```
Power management options --->
    [*] Suspend to RAM and standby
    [ ] Skip kernel's sys_sync() on suspend to RAM/standby
    [ ] Opportunistic sleep
    [ ] User space wakeup sources interface
    -*- Device power management core functionality
    [*] Power Management Debug Support
    [*] Extra PM attributes in sysfs for low-level debugging/testing
    [ ] Test suspend/resume and wakealarm during bootup
    [ ] Device suspend/resume watchdog
    < > Advanced Power Management Emulation
    [ ] Enable workqueue power-efficient mode by default
    [*] For storage less than 32M, enable this when using ota
```

若是 linux-4.9 内核，还需如下配置：

```
CPU Power Management --->
    CPU Idle --->
        [*] CPU idle PM support
        [ ] Ladder governor (for periodic timer tick)
        -*- Menu governor (for tickless system)
            ARM CPU Idle Drivers --->
                [*] Generic ARM/ARM64 CPU idle Driver
                [ ] CPU Idle Driver for Calxeda processors
                [ ] Cpu Idle Config for sunxi soc
```

配置成功后可以在 Tina 端看到如下节点：

```
root@TinaLinux:/# cd sys/power/
root@TinaLinux:/sys/power# ls
autosleep          pm_print_times     state              wake_unlock
pm_async           pm_test            sunxi              wakeup_count
pm_freeze_timeout  pm_wakeup_irq      wake_lock
root@TinaLinux:/sys/power# ls sunxi/
dram_crc_paras     scene_lock         suspend_delay_ms   wakeup_src
parse_bitmap_en    scene_state        suspend_freq
parse_status_code  scene_unlock       sys_pwr_dm_mask
pm_debug_mask      scenelock_debug    time_to_wakeup_ms
```

图 2: 休眠唤醒调试节点

注意：不同平台可能存在差异。

2.3 standby 功能

2.3.1 关键代码和主要流程

关键代码路径：

linux-4.9/kernel/power/main.c
linux-4.9/kernel/power/suspend.c
linux-4.9/drivers/soc/sunxi/pm.c

主要流程：

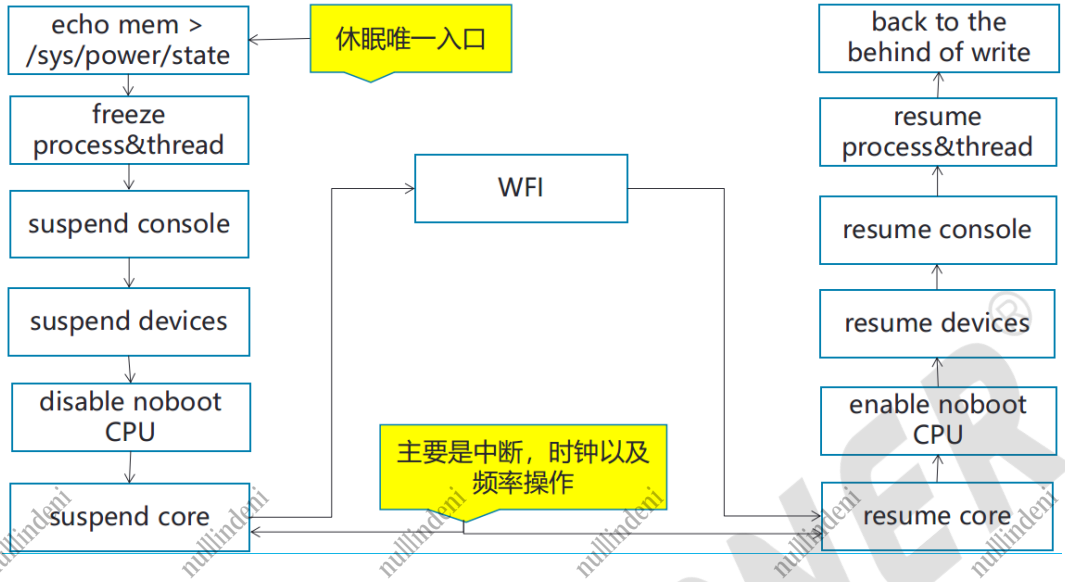

图 3: standby 主要流程

2.3.2 系统休眠

休眠操作就一条命令：

```
echo mem > /sys/power/state;
```

系统就会进入休眠。现象：控制台打印 ``PM: Entering mem sleep"，然后控制台关闭。如下图：

```
root@TinaLinux:/# echo mem > sys/power/state
[  977.790332] PM: suspend entry 1970-01-01 00:16:17.802426141 UTC
[  977.796919] [pm]valid
[  977.799573] PM: Syncing filesystems ... done.
[  977.807765] PM: Preparing system for mem sleep
[  977.857977] Freezing user space processes ... (elapsed 0.001 seconds) done.
[  977.867035] Freezing remaining freezable tasks ... (elapsed 0.001 seconds) done.
[  977.876715] PM: Entering mem sleep
[  977.880567] [pm]3 state begin
[  977.883867] Suspending console(s) (use no_console_suspend to debug)
```

图 4: 休眠时的部分打印

2.3.3 系统唤醒

通过触发中断的方式，唤醒系统。常用唤醒源有：电源按键唤醒、其他按键唤醒、Wifi 唤醒、蓝牙唤醒、Usb 唤醒、Sd 卡唤醒、定时唤醒、自配置 GPIO。

注意：不同平台的唤醒源支持情况不同。

目前 Tina 系统普遍支持的有按键、wifi、自配置 GPIO。

现象：控制台打开，打印 ``PM: Finishing wakeup."。如下图：

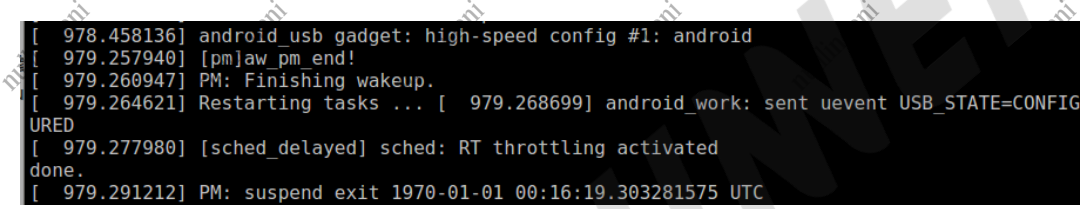

图 5: 唤醒时的部分打印

唤醒源的配置

1. 配置步骤

几乎所有唤醒源都是通过直接和 CPU 相连的管脚发送中断触发的，所以可以理解为 GPIO 中断(Timer 例外)。配置唤醒源就是配置某个管脚的中断功能，

主要分为几步：

I. 指定引脚；

II. 配置该引脚为中断模式；

III. 设置中断的触发模式；

IV. 设置中断标志位，使能该引脚中断口；

V. 使能该引脚对应中断号的中断；

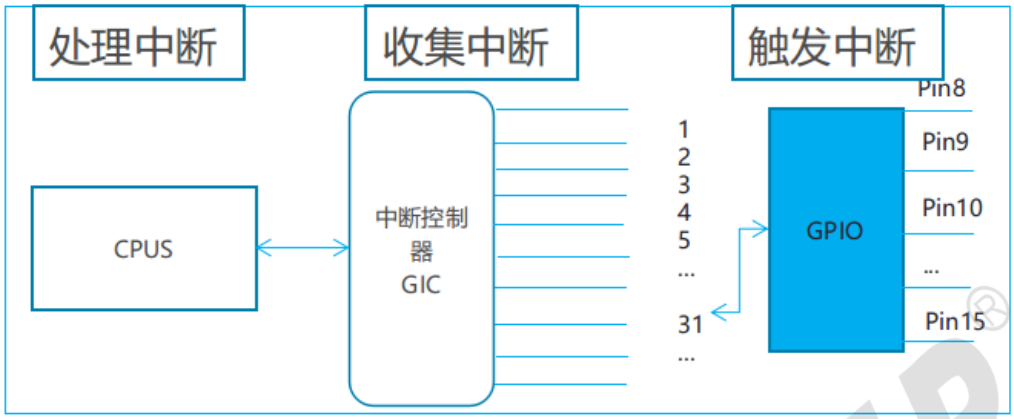

图 6: 中断处理流程

2. 主要接口

```
request_irq(irq_num, irq_handler, irq_flag,device, device_id)
enable_irq_wake(irq_num)
```

注：如 LRADC，TPADC，Timer，只需要驱动调用这些接口就可以了。但如果是 GPIO，还得区分具体是那个 GPIO 口。不同内核版本接口可能存在差异。

3. 例子

如 linux-4.9,wifi 配置的 gpio

```
sunxi_wlan_probe
{
    data->gpio_wlan_hostwake = of_get_named_gpio_flags(np,
    "wlan_hostwake",0, (enum of_gpio_flags *)&config);
    ret = gpio_direction_input(data->gpio_wlan_hostwake);
    ret = device_init_wakeup(dev, 1);
    ret = dev_pm_set_wake_irq(dev, gpio_to_irq(data->gpio_wlan_hostwake));
    ....
    }
```

dts | sys_config 配置

```
wlan_hostwake = port:PE05<6><default><default><default> //wifi
keys { //gpio模拟按键
compatible = "gpio-按键唤醒keys";
status = "okay";

wakeup {
    wakeup-source;
    gpios = <&pio PH9 6 2 2 1>;
    label = "wakeup";
    linux,code = <KEY_WAKEUP>;
    };
    };
```

常用唤醒操作

1. 按键唤醒

```
echo mem > sys/power/state; //进入休眠
```

按下任意 KEYADC 按键唤醒系统。

2. 定时唤醒

```
echo 1000 > /sys/power/sunxi_debug/time_to_wakeup_ms;  //设置定时约5s
echo mem > sys/power/state; //进入休眠
```

系统将在约 5s 后自动唤醒。

3.wifi 唤醒

准备两块 R328 开发板 A,B（其中可以一块是路由器），确认板子能正常休眠唤醒。

```
wifi_connect_ap_test AWTest password //A,B两块板子连接同一个wifi。
ifconfig //查看A,B两块板子的IP地址。
ping IP //保证两块板子在正常工作时能相互ping通。
echo mem > /sys/power/state; //A板进入休眠。
ping IPA //B板执行，尝试唤醒A板。
```

2.4 standby 调试

2.4.1 调试节点

1./sys/power/state

```
cat /sys/power/state --->freeze standby mem
```

首先第一个调试节点就是/sys/power/state，我们可以 cat 发现有三个值可以写入，目前 standby 和 mem 是等价的。写入 mem 字符串测试是否能够正常进入休眠，然后通过按键或是其他唤醒源唤醒测试能否正常唤醒。

2./sys/power/wake_lock

```
cat /sys/power/wake_lock --->NativePower.Display.lock
```

有些平台有唤醒锁的机制（有这个锁就不能进入休眠），例如 R16 平台：如果/sys/power/wake_lock 节点下有锁存在需要释放后才能休眠。

3./sys/power/wake_unlock

```
echo NativePower.Display.lock > /sys/power/wake_unlock 用来释放唤醒锁。
```

与/sys/power/wake_lock 节点成对出现，用来释放唤醒锁。

4./sys/power/pm_test

```
cat /sys/power/pm_test --->[none] core processors platform devices freezer
如：echo devices > /sys/power/pm_test
    echo mem > /sys/power/state
```

该节点专门用来做不同程度的休眠唤醒测试的，休眠 5 S 后自动唤醒。

5./sys/module/printk/parameters/console_suspend

```
echo N > /sys/module/printk/parameters/console_suspend
```

这个节点默认值为 Y，表明延迟 suspende 控制台，此后所即使进入休眠，也会将打印信息输出到控制态，这样方便调试。

6./sys/module/kernel/parameters/initcall_debug

```
echo Y > /sys/module/kernel/parameters/initcall_debug
```

该节点默认值为 N，会打印各个 device 的名称，调用的开始和结束时刻；linux-3.10 及之后的版本, 打印的是 syscore 对应的设备。这样可以精确到具体是那个设备休眠唤醒出了问题。

7./proc/sys/kernel/printk

```
echo 8 > /proc/sys/kernel/printk
```

该节点用来修改打印等级，数值越大，等级越高。一般休眠唤醒测试时执行

8./sys/power/wakeup_src

```
cat /sys/power/wakeup_src
**********************************************************************************************
dynamic wakeup src config:
wakeup_src 0x800000
WAKEUP_SRC is as follow:
CPUS_WAKEUP_GPIO bit 0x800000
want to know gpio config & suspended status detail?
cat /sys/power/aw_pm/debug_mask for help.
wakeup_gpio_map 0x284
WAKEUP_GPIO,for cpus:pl,pm, and axp, is as follow:
        WAKEUP_GPIO_PL port 2.
    WAKEUP_GPIO_PL port 7.
    WAKEUP_GPIO_PL port 9.
    wakeup_gpio_group 0x0
    WAKEUP_GPIO,for cpux:pa,pb,pc,pd,.., is as follow:
    ==========================wakeup src setting usage help info========:
    echo wakeup_src_e para (1:enable)/(0:disable) > /sys/power/wakeup_src
    demo: echo 0x2000 0x200 1 > /sys/power/wakeup_src
    wakeup_src_e para info:
    WAKEUP_SRC is as follow:
```

```
    CPU0_WAKEUP_MSGBOX bit 0x1 CPU0_WAKEUP_KEY bit 0x2
    CPUS_WAKEUP_LOWBATT bit 0x1000 CPUS_WAKEUP_USB bit 0x2000
    CPUS_WAKEUP_AC bit 0x4000 CPUS_WAKEUP_ASCEND bit 0x8000
    CPUS_WAKEUP_DESCEND bit 0x10000 CPUS_WAKEUP_SHORT_KEY bit 0x20000
    CPUS_WAKEUP_LONG_KEY bit 0x40000 CPUS_WAKEUP_IR bit 0x80000
    CPUS_WAKEUP_ALM0 bit 0x100000 CPUS_WAKEUP_ALM1 bit 0x200000
    CPUS_WAKEUP_TIMEOUT bit 0x400000
    CPUS_WAKEUP_GPIO bit 0x800000
    want to know gpio config & suspended status detail?
    cat /sys/power/aw_pm/debug_mask for help.
    CPUS_WAKEUP_USBMOUSE bit 0x1000000 CPUS_WAKEUP_LRADC bit 0x2000000
    CPUS_WAKEUP_CODEC bit 0x8000000 CPUS_WAKEUP_BAT_TEMP bit 0x10000000
    CPUS_WAKEUP_FULLBATT bit 0x20000000 CPUS_WAKEUP_HMIC bit 0x40000000
    CPUS_WAKEUP_POWER_EXP bit 0x80000000
    gpio para info:
        SUNXI_BANK_SIZE bit 0x20
        SUNXI_PA_BASE bit 0x0
        SUNXI_PB_BASE bit 0x20
        SUNXI_PC_BASE bit 0x40
        SUNXI_PD_BASE bit 0x60
        SUNXI_PE_BASE bit 0x80
        SUNXI_PF_BASE bit 0xa0
        SUNXI_PG_BASE bit 0xc0
        SUNXI_PH_BASE bit 0xe0
        SUNXI_PI_BASE bit 0x100
        SUNXI_PJ_BASE bit 0x120
        SUNXI_PK_BASE bit 0x140
        SUNXI_PL_BASE bit 0x160
        SUNXI_PM_BASE bit 0x180
        SUNXI_PN_BASE bit 0x1a0
        SUNXI_PO_BASE bit 0x1c0
        AXP_PIN_BASE bit 0x400
*********************************************************************************************
```

该节点用来配置唤醒源，

```
echo wakeup_src para (1:enable)/(0:disable) > /sys/power/sunxi/wakeup_src
```

例如添加 PL7 为唤醒源:

```
echo 0x800000 0x167 1 > /sys/power/sunxi/wakeup_src
```

参数说明：0x800000 表示 CPUS_WAKEUP_GPIO 唤醒源 0x167 指定 PL7 1 表示 enable

9./sys/power/scene_lock

```
cat /sys/power/scene_lock
talking_standby usb_standby mp3_standby boot_fast super_standby normal_standby gpio_standby misc_standby
```

该节点用来存放场景锁，做一些场景测试。Scene_unlock 节点用来释放场景锁。

10./sys/power/pm_print_times

```
echo 1 > /sys/power/pm_print_times
```

该节点用来开启设备调用的开始和结束的打印。

开启打印如下：

```
[ 1168.881358] calling pio+ @ 149, parent: soc
[ 1168.886142] call pio+ returned 0 after 1 usecs
[ 1168.891106] calling soc+ @ 149, parent: none
[ 1168.895983] call soc+ returned 0 after 1 usecs
[ 1168.900923] calling clocks+ @ 149, parent: none
[ 1168.906083] call clocks+ returned 0 after 1 usecs
```

2.5 常见问题

2.5.1 设备级别

现象：执行 echo mem > /sys/power/state; 无论何种方式系统都无法唤醒。devices 级别测试失败。

定位问题：

根据 pm_test 节点先判断是在那个阶段失败。根据经验最容易出问题的是设备级别。

打开相关调试：

```
echo 1 > /sys/power/pm_print_times;
echo N > /sys/module/printk/parameters/console_suspend;
echo Y > /sys/module/kernel/parameters/initcall_debug;
echo 8 > /proc/sys/kernel/printk;
echo devices > /sys/power/pm_test;
echo mem > /sys/power/state;
```

若存在节点/sys/power/pm_print_times

执行 echo 1 > /sys/power/pm_print_times 就可以开启设备调用的打印。

若不存在节点/sys/power/pm_print_times 则如下操作：

在内核代码中做如下修改在 drivers/base/power/main.c 中，打开设备调用的打印。

```
176 // if (pm_print_times_enabled) {
177 pr_info("calling %s+ @ %i, parent: %s\n",
178 dev_name(dev), task_pid_nr(current),
179 dev->parent ? dev_name(dev->parent) : "none");
180 calltime = ktime_get();
181 // }
```

```
191 // if (pm_print_times_enabled) {
192 rettime = ktime_get();
193 delta = ktime_sub(rettime, calltime);
194 pr_info("call %s+ returned %d after %Ld usecs\n", dev_name(dev),
195 error, (unsigned long long)ktime_to_ns(delta) 的>> 10);
196 if(initcall_debug_delay_ms > 0){
197 printk("sleep %d ms for debug. \n", initcall_debug_delay_ms);
198 msleep(initcall_debug_delay_ms);
199 }
200 // }
```

如上进行注释。

若有 ``platform wakeup, standby wakesource is:0x0" 打印则是唤醒阶段失败，否则是休眠阶段失败。通过 log 对比 calling 和 call returned 是否是成对出现。找出缺少的就可以定位到具体那个设备，再到对应驱动去排查。

2.5.2 核心级别

现象：执行 echo mem > /sys/power/state; 无论何种方式系统都无法唤醒。devices 级别测试通过。

定位问题：根据 pm_test 节点先确定 devices 阶段正常。

然后重启执行

```
echo 1 > /sys/power/pm_print_times;
echo N > /sys/module/printk/parameters/console_suspend;
echo Y > /sys/module/kernel/parameters/initcall_debug;
echo 8 > /proc/sys/kernel/printk;
echo core > /sys/power/pm_test; //core级别
echo mem > /sys/power/state;
```

若失败可定位到/linux-4.9/kernel/power/suspend.c 文件。

若成功可定位到/linux-4.9/drivers/soc/sunxi/pm.c 文件。

（不同平台路径可能存在差异，如 R16 有 AXP 和 CPUS,linux-4.9 加入了 ATF，但都是 pm.c）

aw_pm_enter()-->standby(&standby_info);-->standby_main()(对应../pm/standby/standby.c 中的)

在 pm.c 中添加打印定位，pm.c 以及 standby.c 中都有 save_mem_status(RESUME0_START | 0X02);

可通过系统启动 log 的 RTC 打印值来定位休眠唤醒流程如：

```
fel flag = 0x00000000
rtc[0] value = 0x00000000
rtc[1] value = 0x00000000
rtc[2] value = 0x00000000
rtc[3] value = 0x00000000
rtc[4] value = 0x00000000
rtc[5] value = 0x00000000
rtc[6] value = 0x00000000
rtc[7] value = 0x00000000
```

3. nativepower 应用

实时监控电源状态，包括电池充放电、电源状态变化，利用消息通知机制，从底层上报给应用，保护系统用电安全以及监控电源状态变化。

3.1 Tina 配置

make menuconfig 进入 Tina 应用配置界面:

```
Allwinner --->
    <*> nativepower..... nativepower for allwinner--->
        [*] nativepower communicate other process by dbus
    <*> nativepower_utils..... nativepower utilites
```

3.2 代码路径

关键代码路径：

```
/tina/package/allwinner/nativepower/deamon/main.c
/tina/package/allwinner/nativepower/libpower
/tina/package/allwinner/nativepower/libsuspend
```

nativepower package tree:

```
/tina/ackage/allwinner/nativepower
        ---daemon          场景管理，服务端监控和处理客户端发来的dbus消息
    ---demo              使用说明
    ---files             不同功耗场景的配置
    ---include       具体场景的宏定义和函数定义
    ---libnativepower   应用层接口，具体场景函数的实现以及客户端的处理
    ---libpower         wake_lock的请求，释放，计数，获取等
    ---libsuspend 监测wake_lock的状态，使系统进入不同状态
    ---Makefile 编译规则
```

3.3 框架流程

daemon 的 main.cpp 的 main 函数里：

1. 先会申请一个 NATIVE_POWER_DISPLAY_LOCK 的锁，让系统不能进入休眠。

2. 后通过 autosuspend_enable 启动 libsuspend 中的线程去监测 wake_lock 的状态，当状态为无锁时，系统自动进入休眠。具体启动的线程与内核支持的 suspend 方式有关。有 earlysuspend_thread_func 和 suspend_thread_func 两种。

3. 接着会启动 ubus_server_thread_func , 去响应客户端的请求。

4. 完成这些初始化的动作之后，会进入 while 循环，循环监测自动休眠时间是否到了，到了的话就释放 NATIVE_POWER_DISPLAY_LOCK 锁，让系统进入休眠。

整体框架示意图如下：

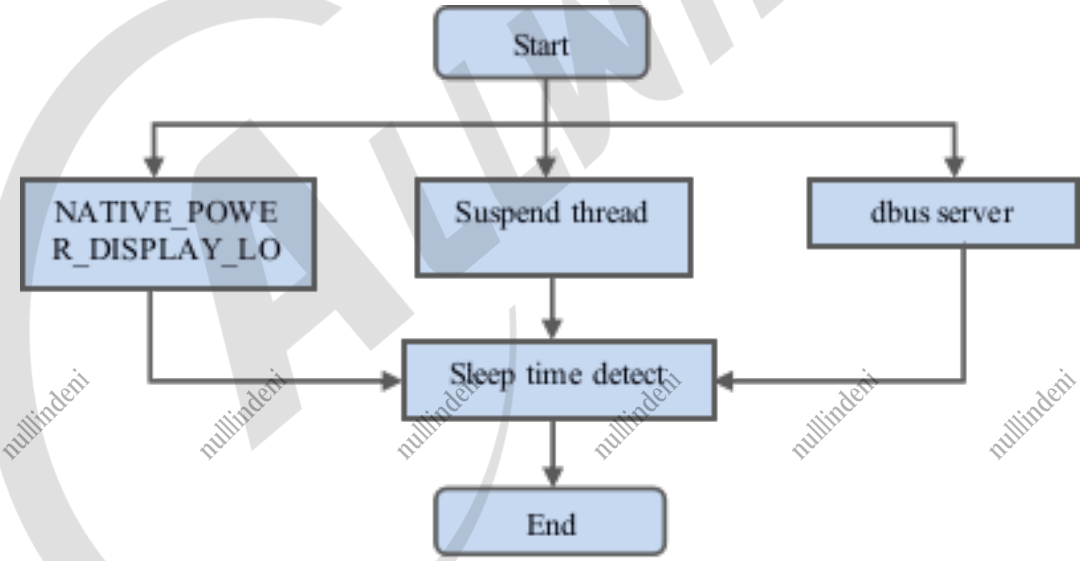

图 7: nativepower 整体框架

3.4 场景管理

系统启动后，会自动设置一个 boot complete 的场景。其他场景的设置，则由用户程序根据自己的需求去调用应用层接口，通过 bus 来设置。不同的场景，通过配置文件去配置。具体到守护进程内，

这是调用了 np_scene_change，然后根据场景名称，读取/etc/config/nativepower 下的对应配置，设置到/sys 文件系统的对应节点中。

具体流程示意图如下：

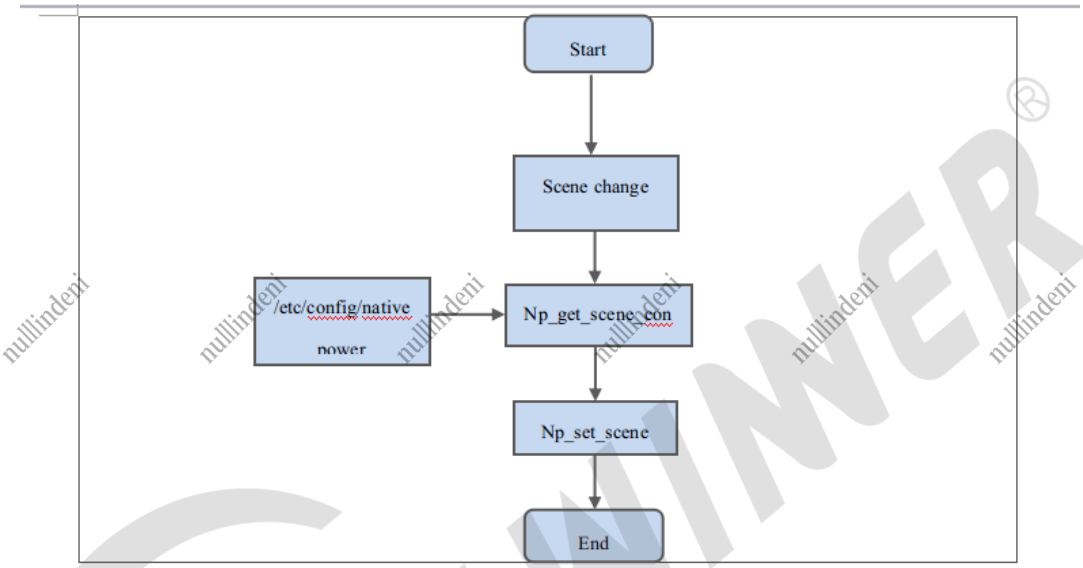

图 8: nativepower 处理流程

3.5 使用示例

C 语言调用

```
np_scene_change("boot_complete");
```

命令行调用

```
dbus-send --system --type=method_call --print-reply --dest=nativepower.dbus.server /nativepower/service/method nativepower.method.interface.method
    uint32:7 uint32:0
```

其中 int32:7 表示 id, 固定为 7，uint32:0 与以下枚举值对应。可由用户按需添加。

nativepower_client suspend 系统休眠

3.6 应用层接口

3.6.1 代码文件

关键代码路径：

```
package/allwinner/nativepower/include
package/allwinner/nativepower/libnativepower
```

3.6.2 总体框架

Libnativepower 封装了通过 dbus 发送请求的接口，这些请求会调用守护进程 nativepower_daemon 的功能函数。

3.6.3 主要功能函数

```
int PowerManagerSuspend(long event_uptime, SuspendReason reason); (详见1)
int PowerManagerShutDown(ShutdownReason reason);(详见2)
int PowerManagerReboot(RebootReason reason);(详见3)
int PowerManagerAcquireWakeLock(const char * id);(详见4)
int PowerManagerReleaseWakeLock(const char *id)功耗管理;(详见5)
int PowerManagerUserActivity(UserActivityReason reason);(详见6)
int PowerManagerSetAwakeTimeout(long timeout_s);管理(详见7)
int PowerManagerSetScene(NativePowerScene scene);(详见8)
```

```
详注：
1 Suspend该函数提供直接进入standby入口，用户调用后系统会直接进入standby。
2 Shutdown 该函数提供系统关机的入口。
3 Reboot 该函数提供重启系统的入口。
4 AcquireWakeLock 该函数提供用户层申请wake_lock的入口。
5 ReleaseWakeLock 该函数提供用户层释放wake_lock的入口。
6 UserActivity 该函数提供用户刷新自动待机时间的入口。
7 SetAwakeTimeout 该函数提供用户设置自动待机时间的入口。
```

8 SetScene 该函数提供用户设置不同功耗场景的入口。

3.6.4 配置文件

用户配置不同功耗场景的配置文件位于

```
package/allwinner/nativepower/files/nativepower
```

具体配置项如下：

```
config scene [场景名称]
option bootlock
option cpu_freq //配置cpu频率
option cpu_gov //配置cpu场景
option CPU0_WAKEUP_MSGBOXhot //配置cpu热插拔
option roomage //设置soc不同的场景
option gpu_freq //配置gpu频率
option dram_freq //配置dram频率
option dram_scene //配置dram场景
```

3.7 模块分析

3.7.1 daemon

1. 概述

开启如下几个线程：

```
tinasuspend_thread 自动进入suspend的操作。
thread_key_power 监控按键power key，并进行处理。
dbus_thread 对客户端发来的dbus消息进行监控和处理。
```

2. 主要函数功能

```
void wakeup_callback(bool sucess);          (详见1)
int set_sleep_state(unsigned int state);  (详见2)
static int open_input(int *fd_array, int *num);          (详见3)
static int open_input(int *fd_array, int *num);          (详见4)
static void *scanPowerKey(void *arg);  (详见5)
int np_input_init();  (详见6)
int np_scene_change(const char *scene_name);  (详见7)
```

详注：
1 设置has_sleep状态，申请wakelock－NativePower.Display.lock，设置mLastUserActivityTime。
2 获取sleep_lock互斥锁，将state值设给has_sleep。说明：0 表示 awake； 1 表示 sleep。
3 搜索并打开匹配/dev/input/event*的文件名，将fd存放到fd_array中，将num设置为fd个数。
4 从系统启动开始计时，获取当前的时间，返回ms数。
5 对power按键的监听。
6 创建执行scanPowerKey函数的线程。
7 调用np_get_scene_config依据给定的scene_name获取对应的场景信息scene。

3.7.2 demo

1. 概述

打印使用说明。用法为：nativepower_client sel [arg]

2. 主要函数功能

```
void usage(char *name);                              (详见1)
int main(int argc, char **argv);                 (详见2)
```

详注：
1 打印一些使用说明的提示。
2 解析第一个参数，依据此参数，分别进行处理：
- SEL_SUSPEND：PowerManagerSuspend
- SEL_SHUTDOWN：PowerManagerShutDown
- SEL_REBOOT：PowerManagerReboot
- SEL_ACQUIRE：PowerManagerAcquireWakeLock
- SEL_RELEASE：PowerManagerReleaseWakeLock
- SEL_USRACTIVITY：PowerManagerUserActivity
- SEL_SETWAKETIME：PowerManagerSetAwakeTimeout
- SEL_SETSCENE：PowerManagerSetScene

3.7.3 libnativepower

1. 概述

Client 向 Server 发送一些 dbus 消息。

2. 主要函数功能

```
int PowerManagerSuspend(long event_uptime, SuspendReason reason);     (详见1)
int PowerManagerShutDown(ShutdownReason reason)   ;                   (详见2)
int PowerManagerReboot(RebootReason reason) ;  (详见3)
int PowerManagerAcquireWakeLock(const char * id) ;  (详见4)
int PowerManagerReleaseWakeLock(const char *id) ;  (详见5)
int PowerManagerUserActivity(UserActivityReason reason) ;  (详见6)
int PowerManagerSetAwakeTimeout(long timeout_s) ;  (详见7)
DBusConnection *dbus_client_open() ;                  (详见8)
```

```
详注：
1 向server发送dbus消息，参数包含NATIVEPOWER_DEAMON_GOTOSLEEP，reason。
2 向server发送dbus消息，参数包含NATIVEPOWER_DEAMON_SHUTDOWN，reason。
3 向server发送dbus消息，参数包含NATIVEPOWER_DEAMON_REBOOT，reason。
4 向server发送dbus消息，参数包含NATIVEPOWER_DEAMON_ACQUIRE_WAKELOCK，wakelock。
5 向server发送dbus消息，参数包含NATIVEPOWER_DEAMON_RELEASE_WAKELOCK，wakelock。
6 向server发送dbus消息，参数包含NATIVEPOWER_DEAMON_USERACTIVITY，reason。
7 向server发送dbus消息，参数包括NATIVEPOWER_DEAMON_SET_AWAKE_TIMEOUT。
8 调用dbus_bus_get连接到系统总线。
```

3.7.4 libsuspend

1. 概述

检测 wake_lock 锁，使系统进入不同状态。

2. 主要函数功能

```
static int autosuspend_init(void);                          (详见1)
void set_wakeup_callback(void (*func)(bool success));  (详见2)
int wait_for_fb_wake(void);                                 (详见3)
```

1 autosuspend的初始化,调用autosuspend_tinasuspend_init或者autosuspend_earlysuspend_init。
2 设置全局wakeup_func的值
3 确保打开/sys/power/wait_for_fb_wake文件没有错误。

3.7.5 libpower

1. 概述

Wake_lock 锁的请求，释放，等待等处理。

2. 主要函数功能

```
int acquire_wake_lock(int lock, const char* id);          (详见1)
int release_wake_lock(const char* id)                        (详见2)
int get_wake_lock_count()                                       (详见3)
```

1 调用initialize_fds函数，新增了一个wake_lock。
2 调用initialize_fds函数，删除一个给定名字的wake_lock。
3 调用initialize_fds函数，获取g_fds[0]对应文件中的所有wake_lock，统计其个数。

4. Declaration

This document is the original work and copyrighted property of Allwinner Technology (“Allwinner”). Reproduction in whole or in part must obtain the written approval of Allwinner and give clear acknowledgement to the copyright owner. The information furnished by Allwinner is believed to be accurate and reliable. Allwinner reserves the right to make changes in circuit design and/or specifications at any time without notice. Allwinner does not assume any responsibility and liability for its use. Nor for any infringements of patents or other rights of the third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of Allwinner. This datasheet neither states nor implies warranty of any kind, including fitness for any particular application.tates nor implies warranty of any kind, including fitness for any particular application.