CANONICAL

Introduction to Runtime Power Management for IO devices

Ming Lei <ming.lei@canoical.com>

Kernel engineer, HWE team
Canonical Ltd.

08 October 2011

Outline

- Introduction to Runtime PM
- Runtime PM Framework
- Runtime PM implementation examples
  - USB subsystem/OMAP3 or 4 platform
- What is the next
- References

Introductions: Terms

- PM(power management): energy usage management
- System sleep: the whole system is put into sleep or hibernation
- RPM: Runtime Power Management
- Runtime PM: does not cover CPU, only for I/O devices in the presentation

Introduction: Ideas behind Runtime PM

- Reduce the amount of energy used in whole system
- Limit the total power draw
- Not all devices are functional at most of time
- I/O devices which are in 'idle' state might be suspended
- Suspended I/O devices can be woken up by driver or itself (generally speaking, both are triggered by user)if they are to be used
- Similar in concepts to CPUIdle

Introduction: Runtime PM overview

- Transparent to user
- Fine granularity PM, per device, not per whole system like system sleep
- Userspace application is not required to be frozen as system sleep
- For one device, runtime pm happens when the device is in 'idle' state and aren't going to be used in the near future, still very different with system sleep

Introduction: Runtime PM overview

- What is the 'idle'?

  - Generally speaking, decided by device driver or its subsystem

  - For example:

    Network interface: driver thinks it is idle after cable is unplugged

    USB subsystem: no data transfer for some time

    I2C: idle if no transfer is ongoing

    ….

  - similar with CPUIdle, scheduler will check if cpu can be put in idle state

- BIOS independent(for x86)

Introduction: Runtime PM overview

- Wakeup setting

  - remote wake-up may be enabled for runtime suspend but disallowed for system sleep

  - compared with system sleep, shorter wakeup latency is involved in runtime pm since system sleep always enters the lowest power consumption state but runtime pm doesn't

Introduction: Why Do We Need a Framework for Device Runtime PM?

- System sleep is not enough to decrease runtime energy consumption
- Devices may depend on another device
- Be helpful to figure out 'idle' condition
- Not doable to do I/O runtime PM in CPUIdle

  - devices may be idle but cpu is not idle

  - task schedule may be caused during device suspend

  - some devices' suspend is moved from cpuidle platform driver to runtime PM (such as, uart suspend on omap3/4)

  - long latency is involved when returning from cpuidle

-
-

Introduction: Which subsystems/drivers has implemented runtime PM up to 3.1-rc4

- Usb subsystem

  HUB, usb mass storage, UVC, HID, CDC, serial, usb net, …
- PCI subsystem

  e1000e, rtl8169, ehci-pci, uhci-pci, ...
- SCSI subsystem

  sd
- I2C subsystem
- MMC subsystem
- Serial devices
- Misc device(gpio, key,.....)
- ARCHs(RPM via dev_pm_domain)

  arch/sh/kernel/cpu/shmobile/pm_runtime.c

  arch/arm/mach-omap1/pm_bus.c

  arch/arm/mach-shmobile/pm_runtime.c

  arch/arm/plat-omap/omap_device.c

Runtime PM Framework

Runtime PM framework

- Device “States"

  Runtime PM framework uses abstract states of devices

  - ACTIVE: Device can do I/O (presumably in the full-power state).

  - SUSPENDED: Device cannot do I/O (presumably in a low-power state).

  - SUSPENDING: Device state is changing from ACTIVE to SUSPENDED.

  - RESUMING: Device state is changing from SUSPENDED to ACTIVE.

Runtime PM framework

- callbacks to be implemented by subsystem or driver

  include/linux/pm.h

```
struct dev_pm_ops {
    ...
    int (*runtime_suspend)(struct device *dev);
    int (*runtime_resume)(struct device *dev);
    int (*runtime_idle)(struct device *dev);
};
```

- RESUMINGING → ACTIVE: ->runtime_resume succeed
- RESUMINGING → SUSPENDED: ->runtime_resume fails
- SUSPENDING → SUSPENDED: ->runtime_suspend succeed
- SUSPENDING → ACTIVE: ->runtime_suspend fails

Runtime PM framework

- ->runtime_idle
  - check if the device is idle and can be suspended
  - if yes, call appropriate runtime PM API to [schedule/queue] auto suspend/suspend for the device

- ->runtime_suspend
  - configure remote wakeup signal if it is allowed and capable
  - execute platform specific things(clock, power,...) to suspend the device

- ->runtime_resume
  - execute platform specific things(clock, power,...) to wakeup the device

Runtime PM framework

- Where to implement callbacks and its priority

```
XX: suspend/resume/idle
if (dev->pm_domain)
        callback = dev->pm_domain->ops.runtime_XX;
 else if (dev->type && dev->type->pm)
        callback = dev->type->pm->runtime_XX;
 else if (dev->class && dev->class->pm)
        callback = dev->class->pm->runtime_XX;
 else if (dev->bus && dev->bus->pm)
        callback = dev->bus->pm->runtime_XX;
 else
        callback = NULL;
```

Runtime PM framework: Callback

- irq_safe is set if the callbacks can be run in interrupt off contexts
- irq_safe is not set

  dev->power.lock is released and irq is enabled before callbacks are called
- irq_safe is set

  dev->power.lock is released but irq is not enabled before callbacks are called
- Release of power lock is the root of all kinds of races

Runtime PM framework: API

- Suspend related API
  - pm_runtime_suspend
  - pm_schedule_suspend
  - pm_runtime_autosuspend
  - pm_request_autosuspend
- Resume related API
  - pm_runtime_resume
  - pm_request_resume
- Idle related API
  - pm_runtime_idle
  - pm_request_idle

Runtime PM framework: API

- Reference Counting
  - Device with references held cannot be suspended/Idled
  - Device can be resumed no matter if its reference is zero or not
- Get reference
  - pm_runtime_get
  - pm_runtime_get_sync
  - pm_runtime_get_noresume
- Put reference
  - pm_runtime_put
  - pm_runtime_put_sync
  - pm_runtime_put_autosuspend
  - pm_runtime_put_sync_suspend
  - pm_runtime_put_sync_autosuspend
  - pm_runtime_put_noidle

Runtime PM framework: API

- Misc API
  - pm_runtime_enable/pm_runtime_disable
  - pm_runtime_allow/pm_runtime_forbid
  - pm_runtime_irq_safe
  - pm_suspend_ignore_children
  - pm_runtime_barrier
  - device_run_wake/device_set_run_wake
  - pm_runtime_use_autosuspend
  - pm_runtime_set_autosuspend_delay
  - pm_runtime_autosuspend_expiration
  - pm_runtime_mark_last_busy

RPM framework: Idle notification

- Put reference and check if the usage count is zero

  pm_runtime_put

  pm_runtime_put_sync
- After ->runtime_resume successfully

  queue a idle notification to the device
- After ->runtime_suspend successfully

  queue a idle notification to its parent
- Before allowing autosuspend

RPM framework: Auto Suspend

- First introduced in usb subsystem
- User can set 'autosuspend delay' or use the default value
- pm_runtime_mark_last_busy marks the last time of
  the device's busy state
- Once auto suspend is scheduled or executed, runtime PM will
  - check if the difference between current time
    and last busy is beyond 'autosuspend delay' of the device.
  - If yes, ->runtime_suspend will be called for the device

RPM framework: Interaction between RPM and system sleep

- Device is suspended before system sleep
  - may runtime resume first in .suspend for different remote wakeup settings
  - During system resume, the simplest approach is to bring all devices back to full power, even if they had been suspended before the system suspend began
- During system suspend
  - pm_runtime_get_noresume(dev)
  - pm_runtime_barrier(dev)
  - abort suspend if wakeup events are found
  - call ->suspend(dev)
  - pm_runtime_disable(dev)
- During system resume
  - pm_runtime_enable(dev)
  - call ->resume(dev)
  - pm_runtime_put_sync(dev)

RPM framework: Wakeup signal

- Depend on the platform and bus type
  - Special signals from low-power states (device signal causes another device to generate an interrupt).
    PCI Power Management Event (PME) signals.
    USB “remote wakeup".
  - Interrupts from low-power states (wakeup interrupts).
- What is needed?
  - Power Domain/Subsystem and/or driver callbacks need to set up devices to generate these signals.
  - The resulting interrupts need to be handled (devices should be put into the ACTIVE state as a result).

Runtime PM implementation examples
USB subsystem / OMAP3 or 4

RPM implementation in usb subsystem

- Usb subsystem implementation(drivers/usb/core/driver.c)

  ->runtime_suspend

```
if (autosuspend_check(udev) != 0)
        return -EAGAIN;
status = usb_suspend_both(udev, PMSG_AUTO_SUSPEND);
```

  ->runtime_resume

```
status = usb_resume_both(udev, PMSG_AUTO_RESUME);
```

  ->runtime_idle

```
if (autosuspend_check(udev) == 0)
        pm_runtime_autosuspend(dev);
```

RPM implementation in usb subsystem

- Usb interface driver
  - simple usage(UVC class, usb-skeleton, ...)
    resume during ->open
    suspend during ->release
  - complicated examples(cdc-acm, cdc_ether, HID, …)
    wakeup device only during urb transfer

RPM implementation in OMAP3/4 platform

- Platform bus
- Power domain(arch/arm/plat-omap/omap_device.c)

```
static struct dev_pm_domain omap_device_pm_domain = {
        .ops = {
                SET_RUNTIME_PM_OPS(_od_runtime_suspend, _od_runtime_resume,
                                   _od_runtime_idle)
                USE_PLATFORM_PM_SLEEP_OPS
                .suspend_noirq = _od_suspend_noirq,
                .resume_noirq = _od_resume_noirq,
        }
};
```

RPM implementation in OMAP3/4 platform

- omap_device_register

  od->pdev.dev.parent = &omap_device_parent;

  od->pdev.dev.pm_domain =&omap_device_pm_domain;

  return platform_device_register(&od->pdev);

- _od_runtime_idle(struct device *dev)

  ->pm_generic_runtime_idle(dev);

  ->dev->driver->pm->runtime_idle(dev) /*if defined*/

- _od_runtime_suspend(struct device *dev)

  ->pm_generic_runtime_suspend(dev);

  ->dev->driver->pm->runtime_suspend(dev) /*if defined*/

  ->omap_device_idle(pdev); /*configure idle mode, disable clocks,...*/

RPM implementation in OMAP3/4 platform

- _od_runtime_resume(struct device *dev)

```
->omap_device_enable(pdev);     /*configure idle mode, enable clocks,...*/
->pm_generic_runtime_resume(dev);
    ->->dev->driver->pm->runtime_resume(dev)        /*if defined*/
```

- omap_device(arch/arm/plat-omap/include/plat/omap_device.h)

```
struct omap_device {
    struct platform_device          pdev;
    struct omap_hwmod               **hwmods;                   /*IPs inside the device*/
    u8              hwmods_cnt;
    struct omap_device_pm_latency   *pm_lats;               /*multiple latency level*/
    u8              pm_lats_cnt;
    ……
}
```

RPM implementation in OMAP3/4 platform

- arch/arm/plat-omap/include/plat/omap_hwmod.h

```
-struct omap_hwmod {
        ……
        struct clk          _clk;                                   /*function clock*/
        struct clockdomain      *clkdm;                         /*clock domain of the function clock*/
        struct omap_hwmod_ocp_if        **masters;          /*interfaces that this hwmod can initiate on*/
        struct omap_hwmod_ocp_if        **slaves;           /*interfaces that this hwmod can respond on*/
        ……
    }
struct omap_hwmod_ocp_if {
        struct omap_hwmod       *master;                    /*initiator*/
        struct omap_hwmod       *slave;                     /*target*/
        struct clk          *_clk;                              /*interface clock*/
        ……
    }
```

RPM implementation in OMAP3/4 platform

- more fine granularity PM unit
  - hwmod is the basic PM unit instead of device
- different view about PM dependency compared with device model(parent vs. child)
  - more hwmods are involved for one device's PM
- Universal clock/clockdomain/.. hardware handling during suspend/resume
- Support multiple latency level

What is the next

What is the next

- Implement RPM in more subsystems or drivers
- PM Qos per device

  - patches from TI will enter 3.2

  - wakeup latency request can be put on per device

  - user space API is under discussion
- PM dependency out of device model
- No idle knowledge in some devices' drivers

  - user space hint is needed?

  - Or device specific implementation

References

References

- drivers/base/power/
- drivers/usb/core/
- arch/arm/plat-omap/omap_device.c
- arch/arm/mach-omap2/
- Include/linux/pm*.h
- Documents/power/runtime_pm.txt
- R. J. Wysocki, Runtime Power Management Framework for I/O Devices in the Linux Kernel (http://events.linuxfoundation.org/slides/2010/linuxcon2010_wysocki.pdf).

CANONICAL

Questions please
Thank you