Runtime Power Management

Kevin Hilman

Deep Root Systems, LLC

khilman@deeprootsystems.com

Runtime PM - Intro

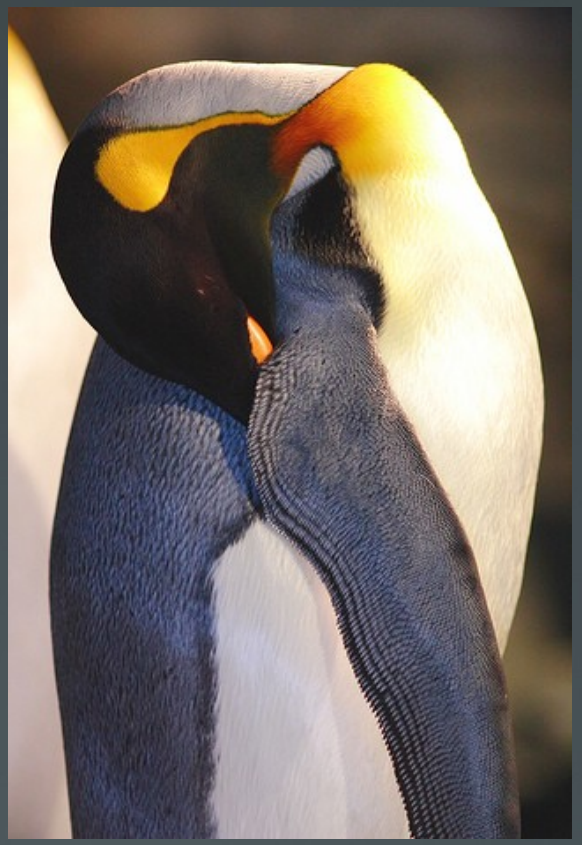
- New PM framework
- Independent PM of devices at runtime
- Idle devices can suspend

- Merged in 2.6.32
- Author: Rafael Wysocki

- But first...

System PM - Crash Course

- Traditional suspend/resume
- System-wide
- All devices together
- Initiated by userspace
- Any device can prevent system suspend



struct dev_pm_ops

- Exists in `struct device_driver`, `struct bus_type`, ...

```
struct dev_pm_ops {
        int (*prepare)(struct device *dev);
        void (*complete)(struct device *dev);
        int (*suspend)(struct device *dev);
        int (*resume)(struct device *dev);
         ...
        int (*suspend_noirq)(struct device *dev);
        int (*resume_noirq)(struct device *dev);
          ...
};
```

- All hooks are optional

`echo mem > /sys/power/state`

- `suspend_ops->begin()`
- `suspend_ops->prepare()`
- `suspend->ops->enter()`
- `suspend_ops->wake()`
- `suspend_ops->finish()`
- `suspend_ops->end()`

(each step iterated for **every** device – bus, type, class)

- Prepare: `->pm->prepare()`
- Early suspend: `->pm->suspend()`
- Late suspend: `->pm->suspend_noirq()`
- Early resume: `->pm->resume_noirq()`
- Late resume: `->pm->resume()`
- Complete: `->pm->complete()`

Runtime PM

- Device-local suspend/resume
- Single device at a time
- Controlled by driver

- devices are independent
- one device cannot prevent other devices from PM

- Dependencies? Stay tuned...

struct dev_pm_ops

- New members for runtime PM

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

Main Runtime PM API

- `pm_runtime_suspend(dev), pm_schedule_suspend(dev, delay)`
  - device *can* suspend
  - subsys: `->runtime_suspend()`
    - Driver: `->runtime_suspend()`
- `pm_runtime_resume(dev), pm_request_resume(dev)`
  - subsys: `->runtime_resume()`
    - Driver: `->runtime_resume()`
- `pm_runtime_idle(dev), pm_request_idle(dev)`
  - subsys: `->runtime_idle()`
    - Driver: `->runtime_idle()`

Runtime PM API: `_get(), _put()`

- Tell PM core whether device is in use
- I need the device
  - `pm_runtime_get(), _sync(), _noresume()`
  - Increment use count, `pm_runtime_resume()`
- I'm done
  - `pm_runtime_put(), _sync, _noidle()`
  - Decrement use count, `pm_runtime_idle()`
- Similar to clock framework usage
  - `clk_enable(), clk_disable()`

Drivers: simple API usage

- Probe
  - `pm_runtime_enable()`
  - probe/configure hardware
  - `pm_runtime_suspend()`
- Activity
  - `pm_runtime_get()`
  - Do work
  - `pm_runtime_put()`
- Done
  - `pm_runtime_suspend()`

Driver callbacks

- All are optional
- `->runtime_suspend()`
  - Save context
  - Power down HW
- `->runtime_resume()`
  - Power up HW
  - Restore context
- `->runtime_idle()`

Runtime PM for `platform_bus`

- Reminder: PM core → bus → driver
- `platform_bus`, common for SoC devices
- `platform_bus` PM functions are `weak`
- Bus code could handle common tasks instead of drivers
  - Clock mgmt
  - Common HW functions
  - Common wakeup handling
    - Check `device_may_wakeup()`

Customizing your platform

- Attatch platform-specific PM data to platform_device
- arch-specific, per-device data

```
struct platform_device {
        …
        struct pdev_archdata archdata;
};
```

- What to put in archdata struct?
  - whatever you need...
  - Device clock
  - HW identifier for platform PM code
  - ...

Bus example: clock management

- Simple example: manage device clocks in common bus layer
- Driver does `_get()` and `_put()` instead of clock management

```
int platform_pm_runtime_suspend(struct device *dev)
{
        struct platform_device *pdev = to_platform_device(dev);
        struct pdev_archdata *ad = &pdev->archdata;
        struct clk *clk = ad->clk;

        dev->driver->pm->runtime_suspend(dev);

        clk_disable(clk)
};
```

OMAP: bus suspend

- Overeride `platform_bus` methods
- Call driver method
  - `->runtime_suspend()`
  - driver should save context
- Disable device HW
  - clock(s), clockdomain
  - powerdomain
- Manage wakeup latencies

Latency Constraints

- Current latency framework is system-wide
  - CPUidle, PM QoS: `CPU_DMA_LATENCY`
- Also need device-specific sleep & wakeup latencies
- Power state of device depends desired latency
- Small wakeup latency?
  - Don't fully disable HW
  - Don't use deep power state

Future Discussion

- Latency/constraints in LDM
  - Device-specific latencies
  - Associted with device, bus...
- Device: sleep/wakeup latencies
- Bus: throughput

Summary

- More granular dynamic PM
- Suitied to modern SoCs w/flexible HW PM features
- Can simplify drivers
  - Move common driver code to bus-level
  - Handle runtime and system PM with same hooks
    - `UNIVERSAL_DEV_PM_OPS()`
- Need common latency/constraint work

Special Thanks

- `Documentation/power/runtime_pm.txt`
- Magnus Damm
  - Initial runtime PM proposals, patches
  - platform_bus, SH mobile
- Paul Walmsley
  - OMAP PM core dev
- Texas Instruments
  - funding OMAP kernel work

- Image credits, links
  - http://scienceblogs.com/tetrapodzoology/2008/09/natural_history_of_sleep.php
  - http://izismile.com/2009/01/13/funny_sleeping_animals_17_photos.htm
  - http://www.bear.org/