Power Management In The Linux* Kernel
Current Status And Future

Rafael J. Wysocki

Intel Open Source Technology Center

September 18, 2013
Outline

1. Introduction
   - The Goal
   - Power Management Variants

2. System-Wide Power Management
   - How It Works
   - The Future

3. Runtime Power Management
   - CPU Power Management
   - I/O Device Runtime PM

4. Resources
What We Are After

Goal (Holy Grail of Computer PM)
Use only as much energy as needed to achieve sufficient performance.

What software can do
(1) Figure out how much performance/capacity is needed and how much latency is acceptable, (2) deliver what’s necessary and turn off everything else (if possible).

What about the Linux kernel?
The Kernel’s Role

Provide the ability to turn things off

- The “mechanics” (carrying out transitions).
- Signaling events (e.g. wakeup).
- Cost information.

Estimate what’s needed

- Follow the actions of user space.
- Use information available internally (e.g. from the CPU scheduler).
- Follow trends.
System-Wide PM and Runtime PM

System-wide power management
- Global energy-saving states.
- Whole system PM transitions.
- User space decides which energy-saving state to go to and when.

Runtime (working state) power management
- User space processes run.
- Individual CPU or I/O device energy saving utilized.
System-Wide PM Overview
System Suspend/Resume Control Flow

Suspend

1. Call Notifiers
2. Freeze Tasks
3. Device Suspend
   - .prepare()
   - .suspend()
   - .suspend_late()
   - .suspend_noirq()
4. Nonboot CPU Offline
5. System Core Offline
6. Turn Off Power

Resume

1. Turn On Power
2. System Core Online
3. Nonboot CPU Online
4. Device Resume
   - .resume_noirq()
   - .resume_early()
   - .resume()
   - .complete()
5. Thaw Tasks
6. Call Notifiers
System Hibernation Control Flow

- Call Notifiers
- Freeze Tasks
- Freeze Transition
  - Device Freeze
    - .prepare()
    - .freeze()
    - .freeze_late()
    - .freeze_noirq()
  - Nonboot CPU Offline
  - System Core Offline
- Create Image
- Thaw Transition
  - System Core Online
  - Nonboot CPU Online
  - Device Thaw
    - .thaw_noirq()
    - .thaw_early()
    - .thaw()
    - .complete()
- Save Image
- Power Off Transition
  - Device Power Off
    - .prepare()
    - .poweroff()
    - .poweroff_late()
    - .poweroff_noirq()
  - Nonboot CPU Offline
  - System Core Offline
- Turn Off Power

Rafael J. Wysocki (Intel OTC)  PM in the Linux Kernel  September 18, 2013  8 / 23
System Restore Control Flow

Call Notifiers

Freeze Tasks

Freeze Transition

Device Freeze

Device Restore

Nonboot CPU Offline

System Core Offline

Nonboot CPU Online

System Core Online

Restore Transition

Thaw Tasks

Call Notifiers

JUMP

Boot Kernel

Image Kernel
System-Wide PM Summary

Global energy-saving states, “frozen” user space
Freeze, Standby, Suspend, Hibernate.

Controlled by user space

1. User space selects the target state.
2. User space decides when to start transitions.
   - Direct command (sysfs write).
   - “Wakelocks” interface.

Used on many systems

Desktop distributions, Android (system suspend + “wakelocks”).
The Future Of System-Wide PM

- Freeze
- Suspend
- Standby
- Hibernate

- Working State
- Suspending Devices
- Freezing of Tasks

Sleep States:
- Core Offline
- CPUs Offline
- Platform Offline
The Future Of System-Wide PM – Summary

Hibernation’s future uncertain
- Replaced by technologies like Intel Rapid Start.
- Not implemented on ARM.

Future platforms may not support “platform offline”
- Suspend and standby may not make sense (no sleep states).
- CPUs offline may not be necessary.
- Freeze may be used to leverage the existing infrastructure.

Freeze may not be useful if runtime PM is very efficient
CPU idle: CPU C-states framework

Idle CPU cores go into low-power states automatically (the low-power state to go to depends on when the CPU core is going to be needed again and on wakeup latency requirements).

CPU performance scaling

Automatic switching performance levels of CPU cores depending on load.

- “Traditional” `cpufreq` (on x86 it uses ACPI P-states).
- `intel_pstate` (governor and scaling driver combined).
CPU Power Management Problem

CPU performance scaling and CPUIdle are separate
- **CPUIdle**: Select the most suitable (low-power) state.
- **CPUfreq**: Balance CPU load and performance level.

However, they are not independent
- CPUIdle depends on the CPU scheduler.
- CPU performance scaling affects the CPU scheduler.

Question: What is more efficient?
- Doing work in bursts and going idle between them.
- Doing work at a suitable performance level without going idle.
The Future Of CPU Power Management

Multiple CPU packages add complexity
Many different scheduling strategies are potentially viable.

PM-Aware Scheduling Conjecture
Energy efficiency may be improved *without hurting performance* by making the CPU scheduler take active role in CPU power management.

Predictions (usual disclaimers apply)
- PM-aware scheduling will be investigated and possibly implemented.
- CPU performance scaling and CPUIdle will be combined.
Working-State (Runtime) PM Frameworks – I/O Devices

Likely to become more and more important over time
System-wide PM may be less useful on future systems.

Device Runtime PM API
Unified API allowing device low-power states to be used consistently involving device drivers, subsystems and the platform.

devfreq (I/O Device Frequency Scaling)
Automatic voltage and frequency scaling for I/O devices depending on performance requirements (uses the Operation Performance Points API).
Example: Platform Device Runtime PM (ACPI)

Runtime Suspend

PM Core

ACPI PM Domain

Driver .runtime_suspend()

acpi_dev_runtime_suspend()

Runtime Resume

Driver .runtime_resume()

acpi_dev_runtime_resume()

ACPI PM Domain

PM Core
Runtime PM On Future Platforms

Hardware design trends

- Increasing integration of components.
- Increasing level of support for aggressive energy saving.

System-on-a-Chip (SoC) configurations

- CPU packages containing I/O devices (e.g. Intel Haswell ULT).
- I/O device states affect package C-states.
- PM features of different components are interdependent.

Challenge: To define clean interfaces for user space.
The Linux kernel supports power management in a number of ways. Both system-wide and working state (runtime) PM are supported. Support for system-wide PM in device drivers is generally better. I/O device runtime PM support improving, but there are issues. CPU PM is well supported, more integration possible. The (long-term) future of system-wide PM is uncertain. Hardware design trends increase PM complexity. Interdependencies between PM features are challenging.

Questions?
References

J. Corbet, *The cpuidle subsystem* (http://lwn.net/Articles/384146/).

Documentation And Source Code

- Documentation/cpu-freq/*
- Documentation/cpuidle/*
- Documentation/power/devices.txt
- Documentation/power/pci.txt
- Documentation/power/runtime_pm.txt
- include/linux/cpufreq.h
- include/linux/cpuidle.h
- include/linux/device.h
- include/linux/pm.h
- include/linux/pm_runtime.h
- include/linux/suspend.h
- drivers/acpi/processor_idle.c
- drivers/base/power/*
- drivers/cpufreq/*
- drivers/cpuidle/*
- kernel/power/*
Intel is a trademark of Intel Corporation in the U. S. and other countries. *Other names and brands may be claimed as the property of others. Copyright © 2013 Intel Corporation, All rights reserved.
Thanks!

Thank you for attention!