System Sleep vs Runtime Power Management

Rafael J. Wysocki

Faculty of Physics U. Warsaw / SUSE Labs / Renesas

May 30, 2011

Outline

Definition Of Terms

It is common to say "power management" meaning "energy usage management". This convention is going to be followed here.

Definition Of Terms

It is common to say “power management” meaning ”energy usage management”. This convention is going to be followed here.

“Runtime power management” referred to in what follows doesn’t cover frequency/voltage scaling in CPUs and I/O devices that are not suspended.

Definition Of Terms

It is common to say "power management" meaning "energy usage management". This convention is going to be followed here.

"Runtime power management" referred to in what follows doesn't cover frequency/voltage scaling in CPUs and I/O devices that are not suspended.

Hibernation is not covered by what follows, so "system sleep" means "the state entered by suspending to RAM".

Definition Of Terms

It is common to say “power management” meaning ”energy usage management”. This convention is going to be followed here.

“Runtime power management” referred to in what follows doesn’t cover frequency/voltage scaling in CPUs and I/O devices that are not suspended.

Hibernation is not covered by what follows, so “system sleep” means “the state entered by suspending to RAM”.

“Remote wakeup” is a mechanism by which suspended devices signal that they should be resumed because of an external event possibly requiring attention.

Two Flavours of Power Management

Runtime power management (Runtime PM)

Turn off (stop clock or remove power) hardware components that aren't going to be used in the near future, transparently from the user space's viewpoint.

Two Flavours of Power Management

Runtime power management (Runtime PM)

Turn off (stop clock or remove power) hardware components that aren't going to be used in the near future, transparently from the user space's viewpoint.

System sleep

Knowing in advance that the whole system is not going to be used in the near future, turn off everything (possibly by force) except for the RAM chips.

Two Flavours of Power Management

Runtime power management (Runtime PM)

Turn off (stop clock or remove power) hardware components that aren't going to be used in the near future, transparently from the user space's viewpoint.

System sleep

Knowing in advance that the whole system is not going to be used in the near future, turn off everything (possibly by force) except for the RAM chips.

Claim

These two concepts are substantially different and generally cannot be used to implement each other.

System Sleep, Suspend and Resume

System sleep is a (low-power) state of the entire system.

System Sleep, Suspend and Resume

System sleep is a (low-power) state of the entire system.

System suspend and resume, respectively, are operations by which the system is put into that state and is brought back into the working state.

System Sleep, Suspend and Resume

System sleep is a (low-power) state of the entire system.

System suspend and resume, respectively, are operations by which the system is put into that state and is brought back into the working state.

Rules

1. System suspend can happen at any time.
2. It should put the system into the deepest (lowest-power) state possible
   - in which the contents of RAM is preserved and
   - from which the system can be woken up in (a) specific way(s).
3. System suspend and resume should be as fast as reasonably possible.

System Suspend and Processes

Since system suspend can happen at any time

User space has to be dealt with during system suspend.

System Suspend and Processes

Since system suspend can happen at any time

User space has to be dealt with during system suspend.

The freezer

Piece of code that puts tasks (all user space and some kernel threads) into a state in which they are known to do nothing.

System Suspend and Processes

Since system suspend can happen at any time

User space has to be dealt with during system suspend.

The freezer

Piece of code that puts tasks (all user space and some kernel threads) into a state in which they are known to do nothing.

Whatever runs next (e.g. device drivers' suspend callbacks) need not worry about user space.

System Suspend and Processes

Since system suspend can happen at any time

User space has to be dealt with during system suspend.

The freezer

Piece of code that puts tasks (all user space and some kernel threads) into a state in which they are known to do nothing.

Whatever runs next (e.g. device drivers' suspend callbacks) need not worry about user space.

However, since the freezer makes user space do nothing

User space is not available during system suspend and resume (e.g. do not request firmware or probe devices at those times).

Suspend Sequence

1. Call notifiers (while user space is still there).
2. Freeze tasks.
3. 1st phase of suspending devices (`.suspend() callbacks`).
4. Disable device interrupts.
5. 2nd phase of suspending devices (`.suspend_noirq() callbacks`).
6. Disable non-boot CPUs (using CPU hot-plug).
7. Turn interrupts off.
8. Execute system core callbacks.
9. Put the system to sleep.

Suspend Sequence

1. Call notifiers (while user space is still there).
2. Freeze tasks.
3. 1st phase of suspending devices (`.suspend() callbacks`).
4. Disable device interrupts.
5. 2nd phase of suspending devices (`.suspend_noirq() callbacks`).
6. Disable non-boot CPUs (using CPU hot-plug).
7. Turn interrupts off.
8. Execute system core callbacks.
9. Put the system to sleep.

Disabling non-boot CPUs after suspending I/O devices is an ACPI requirement.

Resume Sequence

1. (Wakeup signal.)
2. Run boot CPU's wakeup code.
3. Execute system core callbacks.
4. Turn interrupts on.
5. Enable non-boot CPUs (using CPU hot-plug).
6. 1st phase of resuming devices (`.resume_noirq() callbacks`).
7. Enable device interrupts.
8. 2nd phase of suspending devices (`.resume() callbacks`).
9. Thaw tasks.
10. Call notifiers (when user space is back).

Why It Is Not Suitable For Runtime PM

First, because the suspend sequence may take arbitrary time to complete (in fact, the kernel will give up after about 20 s).

Why It Is Not Suitable For Runtime PM

First, because the suspend sequence may take arbitrary time to complete (in fact, the kernel will give up after about 20 s).

"But my hardware can be suspended really quickly ..."

Why It Is Not Suitable For Runtime PM

First, because the suspend sequence may take arbitrary time to complete (in fact, the kernel will give up after about 20 *s*).

"But my hardware can be suspended really quickly ..."

Well, the problem is not the hardware, it is the freezing of tasks.

Why It Is Not Suitable For Runtime PM

First, because the suspend sequence may take arbitrary time to complete (in fact, the kernel will give up after about 20 s).

"But my hardware can be suspended really quickly ..."

Well, the problem is not the hardware, it is the freezing of tasks.

"Hmm, perhaps I don't need to freeze tasks ..."

Why It Is Not Suitable For Runtime PM

First, because the suspend sequence may take arbitrary time to complete (in fact, the kernel will give up after about 20 s).

"But my hardware can be suspended really quickly ..."

Well, the problem is not the hardware, it is the freezing of tasks.

"Hmm, perhaps I don't need to freeze tasks ..."

No, there are too many ways in which user space may interact with drivers for the drivers to intercept all of them without concurrency issues (e.g. deadlocks).

Problem Of Detecting Idleness

System suspend operates on the whole system

Who knows when the whole system is not going to be used?

Problem Of Detecting Idleness

System suspend operates on the whole system

Who knows when the whole system is not going to be used?

Only the user knows that

- The kernel (or any individual part of it) has no idea.
- User space may only use very rough heuristics (e.g. the GUI hasn't been used for that much time, so presumably the system won't be used for a while).

Problem Of Detecting Idleness

System suspend operates on the whole system

Who knows when the whole system is not going to be used?

Only the user knows that

- The kernel (or any individual part of it) has no idea.
- User space may only use very rough heuristics (e.g. the GUI hasn't been used for that much time, so presumably the system won't be used for a while).

What does "idle" mean in the first place?

Only the user can tell what "hasn't been used" means with respect to the whole system.

Problem Of Detecting Idleness

System suspend operates on the whole system

Who knows when the whole system is not going to be used?

Only the user knows that

- The kernel (or any individual part of it) has no idea.
- User space may only use very rough heuristics (e.g. the GUI hasn't been used for that much time, so presumably the system won't be used for a while).

What does "idle" mean in the first place?

Only the user can tell what "hasn't been used" means with respect to the whole system.

Power break even for the whole system is difficult to estimate.

Problem Of Waking Up From “Opportunistic” Sleep

On some platforms some devices cannot wake up the system from sleep.

Problem Of Waking Up From "Opportunistic" Sleep

On some platforms some devices cannot wake up the system from sleep.

In particular, some timer/clock devices may not be able to do that.

Problem Of Waking Up From “Opportunistic” Sleep

On some platforms some devices cannot wake up the system from sleep.

In particular, some timer/clock devices may not be able to do that.

For runtime PM to work, all those timer/clock devices and all devices with remote wakeup capabilities should wake up.

Problem Of Waking Up From “Opportunistic” Sleep

On some platforms some devices cannot wake up the system from sleep.

In particular, some timer/clock devices may not be able to do that.

For runtime PM to work, all those timer/clock devices and all devices with remote wakeup capabilities should wake up.

User space decides which of them will wake up from system sleep.

Problem Of Waking Up From "Opportunistic" Sleep

On some platforms some devices cannot wake up the system from sleep.

In particular, some timer/clock devices may not be able to do that.

For runtime PM to work, all those timer/clock devices and all devices with remote wakeup capabilities should wake up.

User space decides which of them will wake up from system sleep.

Difference between "opportunistic" and user-initiated suspend is hard to tell.

CPUidle

Put idle CPUs into low-power states (no code execution)

- CPU scheduler knows when a CPU is idle.
- Next usage information from clock events.
- Maximum acceptable latency from PM QoS.
- CPU low-power states (C-states) characteristics are known.
- Governors decide what state to go for.

CPUidle

Put idle CPUs into low-power states (no code execution)

- CPU scheduler knows when a CPU is idle.
- Next usage information from clock events.
- Maximum acceptable latency from PM QoS.
- CPU low-power states (C-states) characteristics are known.
- Governors decide what state to go for.

Details are still being worked on (package C-states, “turbo” modes etc.).

CPUidle

Put idle CPUs into low-power states (no code execution)

- CPU scheduler knows when a CPU is idle.
- Next usage information from clock events.
- Maximum acceptable latency from PM QoS.
- CPU low-power states (C-states) characteristics are known.
- Governors decide what state to go for.

Details are still being worked on (package C-states, “turbo” modes etc.).

CPU scheduler may take the “power topology” information into account (work in progress).

I/O Runtime PM

Framework for device runtime PM

1. Subsystems and drivers provide callbacks
   - `.runtime_suspend(dev)`
   - `.runtime_resume(dev)`
   - `.runtime_idle(dev)`
2. Subsystems and drivers handle remote wakeup.
3. The core handles concurrency (locking etc.).
4. The core takes care of device dependencies (parents vs children).
5. The core provides reference counting facilities (detection of idleness).

I/O Runtime PM

Framework for device runtime PM

1. Subsystems and drivers provide callbacks
   - `.runtime_suspend(dev)`
   - `.runtime_resume(dev)`
   - `.runtime_idle(dev)`
2. Subsystems and drivers handle remote wakeup.
3. The core handles concurrency (locking etc.).
4. The core takes care of device dependencies (parents vs children).
5. The core provides reference counting facilities (detection of idleness).

Initial idea was that subsystems and drivers will choose the (low-power) states to put devices into, but that doesn't work well with power domains.

What A Power Domain Is

Technically

Power domain is a set of devices sharing power resources (e.g. clocks, power planes).

What A Power Domain Is

Technically

Power domain is a set of devices sharing power resources (e.g. clocks, power planes).

From the kernel's perspective

Power domain is a set of devices whose power management uses the same set of callbacks with common PM data at the subsystem level.

What A Power Domain Is

Technically

Power domain is a set of devices sharing power resources (e.g. clocks, power planes).

From the kernel's perspective

Power domain is a set of devices whose power management uses the same set of callbacks with common PM data at the subsystem level.

Representation via `struct dev_power_domain` and derived structures.

What A Power Domain Is

Technically

Power domain is a set of devices sharing power resources (e.g. clocks, power planes).

From the kernel's perspective

Power domain is a set of devices whose power management uses the same set of callbacks with common PM data at the subsystem level.

Representation via `struct dev_power_domain` and derived structures.

If a power domain object exists for a device, its PM callbacks take precedence over bus type (or device class, or type) callbacks (new in the current Linus' tree).

Runtime PM Of Power Domains

Observations

1. All devices in a power domain have to be idle so that a shared power resource can be turned off (e.g. clock stopped or power removed).
2. Power is necessary for remote wakeup to work.
3. Latency to turn a power domain on generally depends on all devices in it.

Runtime PM Of Power Domains

Observations

1. All devices in a power domain have to be idle so that a shared power resource can be turned off (e.g. clock stopped or power removed).
2. Power is necessary for remote wakeup to work.
3. Latency to turn a power domain on generally depends on all devices in it.

Thus the PM core should provide means by which:

1. The status of devices in a power domain may be monitored.
2. Decisions to turn power domains off may be made on the basis of (known) device latencies and predicted next usage time (and PM QoS).

I/O Runtime PM Reference Counting Problem

In general, there is no guarantee that all device runtime PM usage counters will be 0 before (or even during) system suspend.

I/O Runtime PM Reference Counting Problem

In general, there is no guarantee that all device runtime PM usage counters will be 0 before (or even during) system suspend.

For this reason, the I/O runtime PM framework cannot be used directly for suspending devices during system suspend.

I/O Runtime PM Reference Counting Problem

In general, there is no guarantee that all device runtime PM usage counters will be 0 before (or even during) system suspend.

For this reason, the I/O runtime PM framework cannot be used directly for suspending devices during system suspend.

Nevertheless, it generally is possible to use the same PM callback routines for both runtime PM and system suspend/resume at the driver level (not necessarily at the subsystem level).

I/O Runtime PM Reference Counting Problem

In general, there is no guarantee that all device runtime PM usage counters will be 0 before (or even during) system suspend.

For this reason, the I/O runtime PM framework cannot be used directly for suspending devices during system suspend.

Nevertheless, it generally is possible to use the same PM callback routines for both runtime PM and system suspend/resume at the driver level (not necessarily at the subsystem level).

It is, in fact, recommended that drivers do that if they can.

Remote Wakeup Problem

Runtime PM requires that remote wakeup be set up, if supported, for all devices being suspended (needed for transparency from the user space's perspective).

Remote Wakeup Problem

Runtime PM requires that remote wakeup be set up, if supported, for all devices being suspended (needed for transparency from the user space's perspective).

In the system sleep case that depends on information provided by user space via *sysfs*.

Remote Wakeup Problem

Runtime PM requires that remote wakeup be set up, if supported, for all devices being suspended (needed for transparency from the user space's perspective).

In the system sleep case that depends on information provided by user space via *sysfs*.

Therefore subsystem-level PM callbacks need to work differently during system suspend/resume and during the analogous runtime PM operations.

Remote Wakeup Problem

Runtime PM requires that remote wakeup be set up, if supported, for all devices being suspended (needed for transparency from the user space's perspective).

In the system sleep case that depends on information provided by user space via *sysfs*.

Therefore subsystem-level PM callbacks need to work differently during system suspend/resume and during the analogous runtime PM operations.

This applies to power domain PM callbacks too.

System Sleep And CPUidle

CPUidle is not sufficient for “quiescing” non-boot CPUs during system suspend because of clock events (e.g. timers).

System Sleep And CPUidle

CPUidle is not sufficient for "quiescing" non-boot CPUs during system suspend because of clock events (e.g. timers).

CPU scheduler has to be told not to schedule anything on those CPUs before they go offline (anything else would be racy).

System Sleep And CPUidle

CPUidle is not sufficient for "quiescing" non-boot CPUs during system suspend because of clock events (e.g. timers).

CPU scheduler has to be told not to schedule anything on those CPUs before they go offline (anything else would be racy).

System core callbacks turn the infrastructure that CPUidle relies on (i.e. clock event devices, interrupt controllers) off.

System Sleep And CPUidle

CPUidle is not sufficient for “quiescing” non-boot CPUs during system suspend because of clock events (e.g. timers).

CPU scheduler has to be told not to schedule anything on those CPUs before they go offline (anything else would be racy).

System core callbacks turn the infrastructure that CPUidle relies on (i.e. clock event devices, interrupt controllers) off.

System suspend may use the same hardware mechanism that CPUidle uses to put CPUs into low-power states, but it does that in a different context.

System Sleep And CPUidle

CPUidle is not sufficient for "quiescing" non-boot CPUs during system suspend because of clock events (e.g. timers).

CPU scheduler has to be told not to schedule anything on those CPUs before they go offline (anything else would be racy).

System core callbacks turn the infrastructure that CPUidle relies on (i.e. clock event devices, interrupt controllers) off.

System suspend may use the same hardware mechanism that CPUidle uses to put CPUs into low-power states, but it does that in a different context.

System sleep means "stay in the low power state indefinitely", so the lowest-power state available can always be used.

System Sleep And Runtime PM Are Different Things

System sleep and runtime PM are related to each other, but they are not the same thing.

System Sleep And Runtime PM Are Different Things

System sleep and runtime PM are related to each other, but they are not the same thing.

In some situations they may bring the system to the same physical state, but they do that in different ways.

System Sleep And Runtime PM Are Different Things

System sleep and runtime PM are related to each other, but they are not the same thing.

In some situations they may bring the system to the same physical state, but they do that in different ways.

It generally is not a good idea to replace one with the other (that will lead to problems in the long run).

References

- P. Bellasi, *Linux Power Management Architecture* (http://www.google.pl/url?sa=t&source=web&cd=8&ved=0CFYQFjAH&url=http%3A%2F%2Fhome.dei.polimi.it%2Fbellasi%2Flib%2Fexe%2Ffetch.php%3Fmedia%3Dteaching%3A2010%3Alinuxpowermanagement.pdf&rct=j&q=Linux%20Power%20Management%20Architecture&ei=g2niTdCOConEvQO7r-zxBg&usg=AFQjCNFa5Ed-fOTDaXGcBX766oBOKuouYw&sig2=6Oe3dkZrUQRscdV6suNBVA&cad=rja).
- R. J. Wysocki, *Runtime Power Management in the PCI Subsystem* (http://www.linuxplumbersconf.org/2010/ocw/system/presentations/279/original/PCI_runtime_PM.pdf).

Documentation And Source Code

- Documentation/power/devices.txt
- Documentation/power/runtime_pm.txt
- Documentation/power/pci.txt
- include/linux/cpuidle.h
- include/linux/device.h
- include/linux/pm.h
- include/linux/pm_runtime.h
- include/linux/pm_wakeup.h
- include/linux/suspend.h
- drivers/base/power/
- drivers/cpuidle/
- kernel/power/