System Sleep vs Runtime Power Management

Rafael J. Wysocki

Faculty of Physics U. Warsaw / SUSE Labs

August 17, 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 Flavors of Power Management

Runtime power management (Runtime PM)

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

Two Flavors of Power Management

Runtime power management (Runtime PM)

Turn off (stop clocks or remove power) hardware components that are idle at the moment and 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 Flavors of Power Management

Runtime power management (Runtime PM)

Turn off (stop clocks or remove power) hardware components that are idle at the moment and 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.

Principle

If both of them are going to be used on a given platform, subsystems and drivers need to support both explicitly.

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 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 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. Turn off the CPU (possibly arm wakeup signals).

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. Turn off the CPU (possibly arm wakeup signals).

Disabling non-boot CPUs after suspending I/O devices is necessary on ACPI-based systems.

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 ..."

Yes, you do. 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, races).

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 "is not 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 "is not 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 devices will wake up from system sleep and that may interfere (destructively) with “opportunistic” system sleep.

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.

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).
6. The core provides helpers (e.g. `pm_runtime_suspend()`).

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).
6. The core provides helpers (e.g. `pm_runtime_suspend()`).

Subsystem callbacks may be overriden by power management domain callbacks (representation via `struct dev_pm_domain`).

Runtime PM Helpers And System Suspend

OK, so why isn't it a good idea to call `pm_runtime_suspend()` in a system suspend callback?

Runtime PM Helpers And System Suspend

OK, so why isn't it a good idea to call `pm_runtime_suspend()` in a system suspend callback?

First of all, because there is no guarantee that all device runtime PM usage counters will be 0 before (or even during) system suspend (the are controlled by user space).

Runtime PM Helpers And System Suspend

OK, so why isn't it a good idea to call `pm_runtime_suspend()` in a system suspend callback?

First of all, because there is no guarantee that all device runtime PM usage counters will be 0 before (or even during) system suspend (the are controlled by user space).

System suspend is supposed to work when `CONFIG_PM_RUNTIME` is unset.

Runtime PM Helpers And System Suspend

OK, so why isn't it a good idea to call `pm_runtime_suspend()` in a system suspend callback?

First of all, because there is no guarantee that all device runtime PM usage counters will be 0 before (or even during) system suspend (the are controlled by user space).

System suspend is supposed to work when `CONFIG_PM_RUNTIME` is unset.

System suspend's handling of wakeup devices is generally different from the runtime PM's one.

Runtime PM Helpers And System Suspend

OK, so why isn't it a good idea to call `pm_runtime_suspend()` in a system suspend callback?

First of all, because there is no guarantee that all device runtime PM usage counters will be 0 before (or even during) system suspend (the are controlled by user space).

System suspend is supposed to work when `CONFIG_PM_RUNTIME` is unset.

System suspend's handling of wakeup devices is generally different from the runtime PM's one.

There are other reasons (see the changelog of commit "*PM: Limit race conditions between runtime PM and system sleep*").

Runtime PM vs System Suspend Rule Of Thumb

Runtime PM

The `.runtime_suspend()` callback operates on a device that's already quiescent.

Runtime PM vs System Suspend Rule Of Thumb

Runtime PM

The `.runtime_suspend()` callback operates on a device that's already quiescent.

System suspend

- The `.suspend()` callback's role is to quiesce the device.
- The `.suspend_noirq()` callback operates on a device quiesced by `.suspend()`.

Runtime PM vs System Suspend Rule Of Thumb

Runtime PM

The `.runtime_suspend()` callback operates on a device that's already quiescent.

System suspend

- The `.suspend()` callback's role is to quiesce the device.
- The `.suspend_noirq()` callback operates on a device quiesced by `.suspend()`.

Conclusion

Often `.runtime_suspend()` and `.suspend_noirq()` can point to the same routine, while `.suspend()` is specific to system suspend.

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

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

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

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.

The system sleep framework is not suitable for implementing runtime PM and the runtime PM framework is not suitable for implementing system suspend callbacks.

References

- R. J. Wysocki, *Runtime Power Management in the PCI Subsystem* (http://www.linuxplumbersconf.org/2010/ocw/system/presentations/279/original/PCI_runtime_PM.pdf).
- R. J. Wysocki, *Runtime Power Management vs System Sleep* (http://events.linuxfoundation.org/slides/2011/linuxcon-japan/lcj2011_wysocki.pdf).
- R. J. Wysocki, *Runtime Power Management Framework for I/O Devices in the Linux Kernel* (http://events.linuxfoundation.org/slides/2010/linuxcon2010_wysocki.pdf).
- R. J. Wysocki, *Runtime Power Management Framework for I/O Devices in the Linux Kernel* (http://events.linuxfoundation.org/slides/2011/linuxcon-japan/lcj2011_wysocki2.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_domain.h
- include/linux/pm_runtime.h
- include/linux/pm_wakeup.h
- include/linux/suspend.h
- drivers/base/power/
- drivers/cpuidle/
- kernel/power/