Mainline Kernel Support for Opportunistic Suspend

Rafael J. Wysocki

Renesas Electronics / SUSE Labs

August 19, 2012

Outline

First Submission (2009)

Wakelock – kernel object used to make the kernel's system suspend core code refuse to start or abort (if already started) a system transition to a sleep state (e. g. suspend-to-RAM).

First Submission (2009)

Wakelock – kernel object used to make the kernel's system suspend core code refuse to start or abort (if already started) a system transition to a sleep state (e. g. suspend-to-RAM).

What is that useful for?

Needed to implement a feature allowing the kernel to start system suspend transitions automatically under the "right" conditions (opportunistic suspend).

First Submission (2009)

Wakelock – kernel object used to make the kernel's system suspend core code refuse to start or abort (if already started) a system transition to a sleep state (e. g. suspend-to-RAM).

What is that useful for?

Needed to implement a feature allowing the kernel to start system suspend transitions automatically under the "right" conditions (opportunistic suspend).

Nobody was impressed

"This surely can be done in a different way" type of reaction.

- From the kernel (through the scheduler, perhaps?)
- From user space

Linux Foundation Collaboration Summit 2010

Meeting with Android kernel team

- Which of the Android-specific kernel features may be merged into the mainline kernel?
- How to change them so that they are more acceptable?

Linux Foundation Collaboration Summit 2010

Meeting with Android kernel team

- Which of the Android-specific kernel features may be merged into the mainline kernel?
- How to change them so that they are more acceptable?

"Wakelocks" was choosen as the first one to try

- Change the name (make it reflect what those things do).
- Clean up the code.
- Document it better.
- Tell people why it is important to you (motivation).
- Introduce the core functionality first, extensions later.

Suspend Blockers Patch Series (April 2010)

First reactions were not hostile

Requests for minor changes mostly.

Suspend Blockers Patch Series (April 2010)

First reactions were not hostile

Requests for minor changes mostly.

Fierce opposition from "Nokia camp" developers

- This is all wrong (e. g. too heavy-handed)!
- We should use a more fine grained approach (from the start).
- This is not a good use case for system suspend.
- Perhaps we don't need system suspend at all.

Suspend Blockers Patch Series (April 2010)

First reactions were not hostile

Requests for minor changes mostly.

Fierce opposition from "Nokia camp" developers

- This is all wrong (e. g. too heavy-handed)!
- We should use a more fine grained approach (from the start).
- This is not a good use case for system suspend.
- Perhaps we don't need system suspend at all.

The discussion started to draw attention

More and more people joined and started to throw non-technical agruments or arguments unrelated to the actual topic and insults.

Suspend Blockers Patch Series (April 2010)

First reactions were not hostile

Requests for minor changes mostly.

Fierce opposition from "Nokia camp" developers

- This is all wrong (e. g. too heavy-handed)!
- We should use a more fine grained approach (from the start).
- This is not a good use case for system suspend.
- Perhaps we don't need system suspend at all.

The discussion started to draw attention

More and more people joined and started to throw non-technical agruments or arguments unrelated to the actual topic and insults.

Finally, it became an all-out flame war and the technical point was lost.

Technical Argumentation Examples

System suspend may be pointless (cons)

In principle it should be possible to put the system into the same physical (low-power) state using runtime PM and CPUidle.

Technical Argumentation Examples

System suspend may be pointless (cons)

In principle it should be possible to put the system into the same physical (low-power) state using runtime PM and CPUidle.

That may not be the case, though

Consider a system with the currently used clock event device in a power domain along with some other I/O devices. That domain may be turned off during system suspend but not in the working state.

Technical Argumentation Examples

System suspend may be pointless (cons)

In principle it should be possible to put the system into the same physical (low-power) state using runtime PM and CPUidle.

That may not be the case, though

Consider a system with the currently used clock event device in a power domain along with some other I/O devices. That domain may be turned off during system suspend but not in the working state.

Freezing of user space may help to handle misbehaving apps (pros)

It ensures that applications don't use the CPU "in the background".

Technical Argumentation Examples

System suspend may be pointless (cons)

In principle it should be possible to put the system into the same physical (low-power) state using runtime PM and CPUidle.

That may not be the case, though

Consider a system with the currently used clock event device in a power domain along with some other I/O devices. That domain may be turned off during system suspend but not in the working state.

Freezing of user space may help to handle misbehaving apps (pros)

It ensures that applications don't use the CPU "in the background".

That may not be the case too

What about applications using suspend blockers?

PM Mini-Summit in 2010 (Before Kernel Summit)

Not a very successful event

- Few people in attendance.
- Nobody from the Android camp.
- Nokia developers present.

PM Mini-Summit in 2010 (Before Kernel Summit)

Not a very successful event

- Few people in attendance.
- Nobody from the Android camp.
- Nokia developers present.

Why was opportunistic suspend regarded as a mistake?

- It was perceived as a (poor man's) replacement of runtime PM.
- It would require changes throughout the whole kernel up to user space to ensure that automatic suspend is blocked in the entire wakeup events processing paths.

PM Mini-Summit in 2010 (Before Kernel Summit)

Not a very successful event

- Few people in attendance.
- Nobody from the Android camp.
- Nokia developers present.

Why was opportunistic suspend regarded as a mistake?

- It was perceived as a (poor man's) replacement of runtime PM.
- It would require changes throughout the whole kernel up to user space to ensure that automatic suspend is blocked in the entire wakeup events processing paths.

What about misbehaving applications?

It should be the user/app store responsibility to catch them.

In The Meantime . . .

Android was growing in the market

- More and more users.
- More and more devices.
- More and more vendors interested in it.

In The Meantime . . .

Android was growing in the market

- More and more users.
- More and more devices.
- More and more vendors interested in it.

And it was using . . . wakelocks

As in the first submission from 2009.

In The Meantime . . .

Android was growing in the market

- More and more users.
- More and more devices.
- More and more vendors interested in it.

And it was using . . . wakelocks

As in the first submission from 2009.

This started to be a real problem for the mainline kernel

- Device drivers written for Android were supposed to use wakelocks.
- They formed a growing pile of kernel code that couldn't be merged into the mainline (in the form it actually was used in).

New Approach: Wakeup Event Counters (June 2010)

Why don't we address the most obvious problem alone first?

There was a known race condition between the handling of wakeup events and the system suspend process: wakeup events could be lost if they occured during system suspend even though the associated physical signals would wake up the system from sleep.

New Approach: Wakeup Event Counters (June 2010)

Why don't we address the most obvious problem alone first?

There was a known race condition between the handling of wakeup events and the system suspend process: wakeup events could be lost if they occured during system suspend even though the associated physical signals would wake up the system from sleep.

The idea was to use counters

- A counter of wakeup events in progress.
- A running counter of processed wakeup events.
- Per-device counters of wakeup events associated with the give device.

New Approach: Wakeup Event Counters (June 2010)

Why don't we address the most obvious problem alone first?

There was a known race condition between the handling of wakeup events and the system suspend process: wakeup events could be lost if they occured during system suspend even though the associated physical signals would wake up the system from sleep.

The idea was to use counters

- A counter of wakeup events in progress.
- A running counter of processed wakeup events.
- Per-device counters of wakeup events associated with the give device.

The `/sys/power/wakeup_count` interface

Could be used to trigger wakeup events count checking during system suspend.

pm_stay_awake() and pm_relax()

pm_stay_awake()

- Increments the counter of wakeup events in progress and the counter of events associated with its argument (a device).
- Roughly corresponds to the "locking" of a wakelock.

pm_stay_awake() and pm_relax()

pm_stay_awake()

- Increments the counter of wakeup events in progress and the counter of events associated with its argument (a device).
- Roughly corresponds to the "locking" of a wakelock.

pm_relax()

- Decrements the counter of wakeup events in progress and increments the running counter of all wakeup events.
- Roughly corresponds to the "unlocking" of a wakelock.

pm_stay_awake() and pm_relax()

pm_stay_awake()

- Increments the counter of wakeup events in progress and the counter of events associated with its argument (a device).
- Roughly corresponds to the "locking" of a wakelock.

pm_relax()

- Decrements the counter of wakeup events in progress and increments the running counter of all wakeup events.
- Roughly corresponds to the "unlocking" of a wakelock.

pm_wakeup_event()

Roughly pm_stay_awake() with a timer set up to do the equivalent of pm_relax() in the future.

Wakeup Source Objects (September 2010)

`struct wakeup_source`

- Representing entities that can generate wakeup events within the kernel (not only devices).
- Collecting wakeup statistics.

Wakeup Source Objects (September 2010)

`struct wakeup_source`

- Representing entities that can generate wakeup events within the kernel (not only devices).
- Collecting wakeup statistics.

Scalability improvements

- Global spinlock dropped.
- One atomic variable used to store both the global counters.
- List of wakeup source objects protected by RCU.
- Separate spinlock for each wakeup source.

Wakeup Source Objects (September 2010)

`struct wakeup_source`

- Representing entities that can generate wakeup events within the kernel (not only devices).
- Collecting wakeup statistics.

Scalability improvements

- Global spinlock dropped.
- One atomic variable used to store both the global counters.
- List of wakeup source objects protected by RCU.
- Separate spinlock for each wakeup source.

Create wakeup source objects for wakeup devices automatically

When they are enabled to wake up the system from sleep.

"Raw" Wakeup Source API

__pm_stay_awake(), __pm_relax(), __pm_wakeup_event()

Analogous to pm_stay_awake(), pm_relax() and pm_wakeup_event(), respectively, but operate on wakeup source objects directly (instead of devices).

"Raw" Wakeup Source API

__pm_stay_awake(), __pm_relax(), __pm_wakeup_event()

Analogous to pm_stay_awake(), pm_relax() and pm_wakeup_event(), respectively, but operate on wakeup source objects directly (instead of devices).

wakeup_source_create(), wakeup_source_add()

Create a wakeup source object and add it to the kernel's list of wakeup sources.

"Raw" Wakeup Source API

__pm_stay_awake(), __pm_relax(), __pm_wakeup_event()

Analogous to pm_stay_awake(), pm_relax() and pm_wakeup_event(), respectively, but operate on wakeup source objects directly (instead of devices).

wakeup_source_create(), wakeup_source_add()

Create a wakeup source object and add it to the kernel's list of wakeup sources.

wakeup_source_remove(), wakeup_source_destroy()

Delete a wakeup source object from the kernel's list of wakeup sources and destroy it.

"Raw" Wakeup Source API

__pm_stay_awake(), __pm_relax(), __pm_wakeup_event()

Analogous to pm_stay_awake(), pm_relax() and pm_wakeup_event(), respectively, but operate on wakeup source objects directly (instead of devices).

wakeup_source_create(), wakeup_source_add()

Create a wakeup source object and add it to the kernel's list of wakeup sources.

wakeup_source_remove(), wakeup_source_destroy()

Delete a wakeup source object from the kernel's list of wakeup sources and destroy it.

Analogous to the (kernel) wakelocks API.

Linux Foundation Collaboration Summit 2011

Linux Foundation Collaboration Summit 2011

My Personal "OK moment"

Two teenagers on a Muni bus in San Francisco talking about mobile phones and one of them using the words "iPhone" and "Android" in one sentence.

Linux Foundation Collaboration Summit 2011

My Personal "OK moment"

Two teenagers on a Muni bus in San Francisco talking about mobile phones and one of them using the words "iPhone" and "Android" in one sentence.

At the conference

There were people asking about the status of wakelocks and whether or not there was any plan to merge that feature.

Linux Foundation Collaboration Summit 2011

My Personal "OK moment"

Two teenagers on a Muni bus in San Francisco talking about mobile phones and one of them using the words "iPhone" and "Android" in one sentence.

At the conference

There were people asking about the status of wakelocks and whether or not there was any plan to merge that feature.

Then I started to think about implementing kernel-based opportunistic suspend on top of wakeup sources.

Kernel Summit 2011

"Patch review" discussion

That became a discussion about merging out-of-the-tree features. During that discussion Linus stated clearly that in his opinion we should consider merging "suspend blockers".

Kernel Summit 2011

"Patch review" discussion

That became a discussion about merging out-of-the-tree features. During that discussion Linus stated clearly that in his opinion we should consider merging "suspend blockers".

I thought it was actually too late for that, but the idea of implementing kernel-based opportunistic suspend on top of wakeup sources suddenly appeared to me as something that might succeed.

Kernel Summit 2011

"Patch review" discussion

That became a discussion about merging out-of-the-tree features. During that discussion Linus stated clearly that in his opinion we should consider merging "suspend blockers".

I thought it was actually too late for that, but the idea of implementing kernel-based opportunistic suspend on top of wakeup sources suddenly appeared to me as something that might succeed.

I still was concerned about extra code that would have to be added to ensure the "protection" of wakeup events on their way up to user space (at least one wakeup source would have to be active all the time).

Autosleep Patch Set (February 2012)

`/sys/power/autosleep` (not present in current Android)

Make the kernel attempt to start a transition into a sleep state (e. g. system suspend) whenever there are no active wakeup sources.

Autosleep Patch Set (February 2012)

/sys/power/autosleep (not present in current Android)

Make the kernel attempt to start a transition into a sleep state (e. g. system suspend) whenever there are no active wakeup sources.

/sys/power/wake_lock

Create wakeup source objects (with names) and make them active.

Autosleep Patch Set (February 2012)

/sys/power/autosleep (not present in current Android)

Make the kernel attempt to start a transition into a sleep state (e. g. system suspend) whenever there are no active wakeup sources.

/sys/power/wake_lock

Create wakeup source objects (with names) and make them active.

/sys/power/wake_unlock

"Deactivate" wakeup source objects created via /sys/power/wake_lock.

Autosleep Patch Set (February 2012)

/sys/power/autosleep (not present in current Android)

Make the kernel attempt to start a transition into a sleep state (e. g. system suspend) whenever there are no active wakeup sources.

/sys/power/wake_lock

Create wakeup source objects (with names) and make them active.

/sys/power/wake_unlock

"Deactivate" wakeup source objects created via /sys/power/wake_lock.

Limited number and garbage collection

There is a build-time limit of the number of wakeup source objects that user space can create and a garbage collection mechanism for the unused ones (both were made optional at the request of Android developers).

Wakeup Event During Wait Issue

The problem appears when user space is waiting for an event that may be a wakeup one (the system should be able to go to sleep while waiting, but it should be woken up and prevented from going to sleep again once an event has been detected).

Wakeup Event During Wait Issue

The problem appears when user space is waiting for an event that may be a wakeup one (the system should be able to go to sleep while waiting, but it should be woken up and prevented from going to sleep again once an event has been detected).

Android handles that through `ioctl()` interfaces added to drivers like `evdev`, but that potentially requires adding many of them (not a popular idea).

Wakeup Event During Wait Issue

The problem appears when user space is waiting for an event that may be a wakeup one (the system should be able to go to sleep while waiting, but it should be woken up and prevented from going to sleep again once an event has been detected).

Android handles that through `ioctl()` interfaces added to drivers like `evdev`, but that potentially requires adding many of them (not a popular idea).

During the discussion Matt Helsley suggested that it might be handled through `epoll()` if one additional flag was added to it. That idea was then implemented by Arve Hjønnevåg from the Android kernel team.

EPOLLWAKEUP And CAP_BLOCK_SUSPEND

EPOLLWAKEUP

New `epoll()` flag such that if an `epoll()` event with that flag set is ready, a wakeup source will be automatically activated by the kernel. It will prevent the system from going to sleep until the event is consumed (removed from the queue) by user space.

EPOLLWAKEUP And CAP_BLOCK_SUSPEND

EPOLLWAKEUP

New `epoll()` flag such that if an `epoll()` event with that flag set is ready, a wakeup source will be automatically activated by the kernel. It will prevent the system from going to sleep until the event is consumed (removed from the queue) by user space.

The `EPOLLWAKEUP` flag is ignored uless the process using `epoll()` has the `CAP_BLOCK_SUSPEND` capability (name suggested by Michael Kerrisk). That capability is also necessary for using the `/sys/power/wake_lock` and `/sys/power/wake_unlock` interfaces.

EPOLLWAKEUP And CAP_BLOCK_SUSPEND

EPOLLWAKEUP

New `epoll()` flag such that if an `epoll()` event with that flag set is ready, a wakeup source will be automatically activated by the kernel. It will prevent the system from going to sleep until the event is consumed (removed from the queue) by user space.

The `EPOLLWAKEUP` flag is ignored uless the process using `epoll()` has the `CAP_BLOCK_SUSPEND` capability (name suggested by Michael Kerrisk). That capability is also necessary for using the `/sys/power/wake_lock` and `/sys/power/wake_unlock` interfaces.

The `EPOLLWAKEUP` flag and the `CAP_BLOCK_SUSPEND` capability are not present in the current Android (for what I know).

Where Are We Now?

- The autosleep support as described was shipped in the 3.5 kernel.

Where Are We Now?

- The autosleep support as described was shipped in the 3.5 kernel.
- Theoretically, it should allow Android developers to switch over to wakeup sources, but that requires user space modifications (because of the `EPOLLWAKEUP` flag, `CAP_BLOCK_SUSPEND` and the `/sys/power/autosleep` interface).

Where Are We Now?

- The autosleep support as described was shipped in the 3.5 kernel.
- Theoretically, it should allow Android developers to switch over to wakeup sources, but that requires user space modifications (because of the `EPOLLWAKEUP` flag, `CAP_BLOCK_SUSPEND` and the `/sys/power/autosleep` interface).
- I have no idea if/when that is going to happen.

Where Are We Now?

- The autosleep support as described was shipped in the 3.5 kernel.
- Theoretically, it should allow Android developers to switch over to wakeup sources, but that requires user space modifications (because of the `EPOLLWAKEUP` flag, `CAP_BLOCK_SUSPEND` and the `/sys/power/autosleep` interface).
- I have no idea if/when that is going to happen.
- Still, given that the Android kernel developers took part in the development of the autosleep patch set, I'm optimistic.

Where Are We Now?

- The autosleep support as described was shipped in the 3.5 kernel.
- Theoretically, it should allow Android developers to switch over to wakeup sources, but that requires user space modifications (because of the `EPOLLWAKEUP` flag, `CAP_BLOCK_SUSPEND` and the `/sys/power/autosleep` interface).
- I have no idea if/when that is going to happen.
- Still, given that the Android kernel developers took part in the development of the autosleep patch set, I'm optimistic.

Questions?

References


Jonathan Corbet, *Wakelocks and the embedded problem* (http://lwn.net/Articles/318611/).


Jonathan Corbet, *From wakelocks to a real solution* (http://lwn.net/Articles/319860/).


Jonathan Corbet, *Suspend block* (http://lwn.net/Articles/385103/).


Jonathan Corbet, *Blocking suspend blockers* (http://lwn.net/Articles/388131/).


Jonathan Corbet, *Suspend blocker suspense* (http://lwn.net/Articles/389407/).


Jonathan Corbet, *What comes after suspend blockers* (http://lwn.net/Articles/390369/).


Jonathan Corbet, *This week's episode of "Desperate Androids"* (http://lwn.net/Articles/391245/).


Jonathan Corbet, *Another wakeup event mechanism* (http://lwn.net/Articles/393314/).


Rafael J. Wysocki, *An alternative to suspend blockers* (http://lwn.net/Articles/416690/).


Jonathan Corbet, *A new approach to opportunistic suspend* (http://lwn.net/Articles/460644/).


Jonathan Corbet, *Yet another opportunity for opportunistic suspend* (http://lwn.net/Articles/463517/).


Jonathan Corbet, *KS2011: Patch review* (http://lwn.net/Articles/464298/).


Jonathan Corbet, *Autosleep and wake locks* (http://lwn.net/Articles/479841/).

Source Code

- include/linux/pm_wakeup.h
- drivers/base/power/main.c
- drivers/base/power/sysfs.c
- drivers/base/power/wakeup.c
- kernel/power/*

Thanks!

Thank you for attention!