

# Re: Power Management framework proposal

---

*Source:* <http://linux.derkeiler.com/Mailing-Lists/Kernel/2007-07/msg09580.html>

---

- *From:* [david@xxxxxxx](mailto:david@xxxxxxx)
  - *Date:* Sun, 22 Jul 2007 20:51:48 -0700 (PDT)
- 

On Sun, 22 Jul 2007, Arjan van de Ven wrote:

On Sun, 2007-07-22 at 11:56 -0700, david@xxxxxxx wrote:

I have a concern with this approach though. It seems to assume that there is one global thing somewhere that sets the system state; in my experience that is the wrong approach; in fact there is a very definite evidence that there are many decisions on power that are to be made local at a high frequency. An example of this is the processor speed; the ondemand governer does exactly this for the cpus that can switch speeds fast; it's just impossible to beat such a local, fast decision with anything on a global scale.

the intent was not to have one global call that sets the mode on all devices, but rather have one call for each device/subsystem, just the same call in each case.

there's also nothing that says that there can only be one thing setting the mode (although that does mean a fourth call 'report\_current\_mode()' or similar is needed). and if you choose to have two pieces of software managing the same device things could get 'interesting'.

as for the speed that such decisions need to be made.

this API is not saying anything about the speed of the decisions. it's also not saying anything about if the decision making is being done by kernelspace or userspace. it's just providing a common way for whatever software is doing the decision making to find out it's options and set the modes.

## Re: Power Management framework proposal

but it makes for a layer between the device and the setting of the modes.. which sort of would defeat the option of having things truly local.

Settings don't mean much in general (in specific cases, maybe), it's the requirements that matter. The *\*intent\** matters. Linus forced this into cpufreq way back, and while I and perhaps others thought he was just being silly, 6 years later it turns out he was absolutely right.

and the more I am seeing of cpufreq the more it looks like what I'm proposing, so I'm glad to see that it's a good model :-)

Maybe something else

A power policy management framework doesn't need a unified framework (I know this for a fact, I'm hoping to release the code within a few weeks). A unified interface doesn't even help one single bit: the semantics of each part is *\*extremely\** different even if you make it look the same; the sameness is only cosmetic.

The consequences of managing a disk vs managing a cpu vs managing the LCD brightness via the X server are all very different. The tradeoffs you need to make are all very different. The things you want to control are all very different. Trying to force a standard interface makes the interface for a specific subsystem go away from the *\*actual\** best interface for that subsystem, for no gain since the thing that manages the policy needs to have different parts for each *\*anyway\**.

Ok, I can see that if things really are different then it's worth doing different things to control them.

however, let me go back to my original post on the subject here

right now drivers are supposed to have (forgive me if I get the function names wrong)

```
initialize()
shutdown()
suspend()
suspend_late()
resume()
resume_early()
```

with suspend taking one of several parameters

```
PM_EVENT_SUSPEND
PM_EVENT_FREEZE
PM_EVENT_PRETHAW
```

and the notes say that what is supposed to happen is fairly undefined because different things can have vastly

## Re: Power Management framework proposal

different capabilities. so to really control the device you need other, per driver interfaces as well.

this API is driven by the activities that the suspend process is currently designed to use, and each routine assumes given existing state, if you call it when in any other state the results are undefined.

any match to the actual capabilities of the hardware is purely coincidental. to have any ability to control the mode of anything at runtime requires that the code doing so must have specific knowledge of the driver in question.

compare this underdefined mess to the sanity that cpufreq gives you for controlling different vendors CPUs with their different capabilities.

with cpufreq you somewhere have a table that goes something along the lines of

```
freq voltage
2.0GHz 3.0v
1.5GHz 3.0v
1.0GHz 1.5v
500MHz 0.8v
```

and a function that lets you select the freq you want

if cpufreq were to switch over the the API I'm suggesting the table would change to

```
mode capacity power
0 0 0
1 100 100
2 75 100 (or possibly 95, there is some benefit to a slower clock at the same voltage)
3 50 25
4 25 7
```

so it would be a relatively minor change, probably causing more disruption then benefit to change in and of itself.

also, other then efficiency arguments, there's nothing that says the modes must be integers not strings. instead of 0-4 above you could use the entries from under freq in the first table.

I don't know how cpufreq handles a cpu with logic blocks that can be turned off individually but with the type of API I'm talking about you could easily have

```
mode capacity power
0 0 0
1 100 100 (full clock, both blocks on)
2 50 60 (full clock, one block off)
3 50 25 (half clock, half voltage, both blocks on)
4 25 15 (half clock, half voltage, one blocks off)
5 25 7 (quarter clock, quarter voltage, both blocks on)
6 12 4 (quarter clock, quarter voltage, one blocks off)
7 0 1 (clock stopped, but chip still energized, faster to wake up from then mode 0)
```

## Re: Power Management framework proposal

with the benefits of mode 2 vs 3, 4 vs 5, and 7 vs 0 showing up in the transition cost matrix where it would show that it's faster to go up to the high-capacity modes from the first of each set then from the second, even though there are power saving advantages to the second in each set.

but the idea of adding the cpu control to this API was an afterthought, the biggest thing was to get something better than the current mess for other devices, and the fact that cpufreq was initially seen as a waste of time, but now you are seeing it's value could be an argument to do a similar transition for the power modes of other devices as well.

Now I realize that the needs for "hard small embedded" are different from "PC like", and to be honest, I don't think it's entirely possible to unify them; I don't think it's even worthwhile to pursue that (look at where those attempts have gotten us so far)... but I suspect even in the small embedded space a standard, forced and thus unnatural interface isn't what is needed.

I am thinking that a standard way to define the available modes of operation of a piece of hardware is an advantage for all scales. even if the generic API doesn't quite cover every possible mode (if you have enough knobs to twist the combinational explosion of the possible modes may mean that you don't actually implement all of them) making it possible for software to discover and set the modes for different devices without having to know specifics of the drivers would be a good thing.

you mention LCD backlights as an example of something non-standard enough to create a new interface for. I think it would fit the API I'm proposing quite nicely

example 1: a laptop screen

mode	capacity	power	description
0	0	0	off
1	100	100	full brightness
2	70	60	half power to the backlight
3	50	35	quarter power to the backlight
4	30	25	eighth power to the backlight
5	5	10	backlight off.

example 2: a front-panel display on a server (no variable backlight control)

mode	capacity	power	description
0	0	0	off
1	100	100	backlight on
2	50	10	backlight off

unless the device had a light sensor with it I wouldn't expect these settings to be changed automatically, but this API would make it trivial for userspace tools to be able to control the brightness of any display with no driver-specific code, they would just look for display type objects, read the capabilities, and change the modes as the user requests.

currently it would probably take two different software packages to control the backlights on these two devices, one that understands the video display driver (and would probably be pretty specific to that driver)

## Re: Power Management framework proposal

and a second one that would understand the front-panel display driver.

with the current situation it's practically impossible to create a tool that allows you to set the power saving modes for everything in a system. that tool would need to know the ins and outs of every driver, and keep up to date on driver changes.

and the flip side of this is that it's also very hard to get the power saving features of a new device handled in an appropriate manner, you not only need to write the capabilities into the driver, you have to write a utility to control those capabilities, and then try and get similar software included in all the sstem utilities that you would want to use to control those capabilities

with the approach I'm proposing creating such a tool would be fairly simple, it would walk the sysfs tree to see what hardware is there, read what modes it can be set in (including flags that tell you that things below it need to be in modes with specific capabilities if appropriate) and let you change them.

if you don't want to make the shift with cpufreq, that's fine. it sounds like you are at least 90% of the way there anyway, it's not that big a deal, but do you think that there's value in replacing the current ad-hoc approach with something more structured (even if it's not this proposal)?

David Lang

—

To unsubscribe from this list: send the line "unsubscribe linux-kernel" in the body of a message to majordomo@xxxxxxxxxxxxxxxx

More majordomo info at <http://vger.kernel.org/majordomo-info.html>

Please read the FAQ at <http://www.tux.org/lkml/>