

## Re: [PATCH -mm] kexec jump -v9

---

*Source:* <http://linux.derkeiler.com/Mailing-Lists/Kernel/2008-05/msg07252.html>

---

- *From:* [ebiederm@xxxxxxxxxxxxx](mailto:ebiederm@xxxxxxxxxxxxx) (Eric W. Biederman)
  - *Date:* Thu, 15 May 2008 15:03:17 -0700
- 

"Rafael J. Wysocki" <[rjw@xxxxxxx](mailto:rjw@xxxxxxx)> writes:

Well, it looks like we do similar things concurrently. Please have a look here: [http://kerneltrap.org/Linux/Separating\\_Suspend\\_and\\_Hibernation](http://kerneltrap.org/Linux/Separating_Suspend_and_Hibernation)

Yes. Part of the reason I wanted to separate these two conversations I knew something was going on.

Similar patches are in the Greg's tree already.

Taking a look.

I just can't get past the fact in that the only reason hibernation can not use the widely implemented and tested probe/remove is because of filesystems on block devices, and that you are proposing to add 4 methods for each and every driver to handle that case, when they don't need ANYTHING!

I wonder how hard teaching the upper layers to deal with hotplug/remove is?

The more I look at this the more I get the impression that hibernation and suspend should be solved in separate patches. I'm not at all convinced that what is good for the goose is good for the gander for things like your prepare method.

Hibernation seems to be an extreme case of hotplug.

Suspend seems to be just an extreme case of putting unused devices in low power state.

....

I don't like the fact that these methods are power management specific.

How should this impact the greater kernel ecosystem.

```
+ * The externally visible transitions are handled with the help of the following
+ * callbacks included in this structure:
+ *
+ * @prepare: Prepare the device for the upcoming transition, but do NOT change
+ * its hardware state. Prevent new children of the device from being
+ * registered after @prepare() returns (the driver's subsystem and
+ * generally the rest of the kernel is supposed to prevent new calls to the
+ * probe method from being made too once @prepare() has succeeded). If
+ * @prepare() detects a situation it cannot handle (e.g. registration of a
+ * child already in progress), it may return -EAGAIN, so that the PM core
+ * can execute it once again (e.g. after the new child has been registered)
+ * to recover from the race condition. This method is executed for all
+ * kinds of suspend transitions and is followed by one of the suspend
+ * callbacks: @suspend(), @freeze(), or @poweroff().
+ * The PM core executes @prepare() for all devices before starting to
+ * execute suspend callbacks for any of them, so drivers may assume all of
+ * the other devices to be present and functional while @prepare() is being
+ * executed. In particular, it is safe to make GFP_KERNEL memory
+ * allocations from within @prepare(), although they are likely to fail in
+ * case of hibernation, if a substantial amount of memory is requested.
+ * However, drivers may NOT assume anything about the availability of the
+ * user space at that time and it is not correct to request firmware from
+ * within @prepare() (it's too late to do that).
+ *
+ * @complete: Undo the changes made by @prepare(). This method is executed for
+ * all kinds of resume transitions, following one of the resume callbacks:
+ * @resume(), @thaw(), @restore(). Also called if the state transition
+ * fails before the driver's suspend callback (@suspend(), @freeze(),
+ * @poweroff()) can be executed (e.g. if the suspend callback fails for one
+ * of the other devices that the PM core has unsuccessfully attempted to
+ * suspend earlier).
+ * The PM core executes @complete() after it has executed the appropriate
+ * resume callback for all devices.
```

The names above are terrible. Perhaps: @pause/@unpause.

@pause Stop all device driver user space facing activities, and prepare for a possible power state transition.

Essentially these should be very much like bringing an ethernet interface down. The device is still there but we can't do anything with it. The only difference is that this may not be user visible.

```
+ * @suspend: Executed before putting the system into a sleep state in which the
+ * contents of main memory are preserved. Quiesce the device, put it into
+ * a low power state appropriate for the upcoming system state (such as
+ * PCI_D3hot), and enable wakeup events as appropriate.
+ *
+ * @resume: Executed after waking the system up from a sleep state in which the
```

Re: [PATCH -mm] kexec jump -v9

- + \* contents of main memory were preserved. Put the device into the
- + \* appropriate state, according to the information saved in memory by the
- + \* preceding @suspend(). The driver starts working again, responding to
- + \* hardware events and software requests. The hardware may have gone
- + \* through a power-off reset, or it may have maintained state from the
- + \* previous suspend() which the driver may rely on while resuming. On most
- + \* platforms, there are no restrictions on availability of resources like
- + \* clocks during @resume().

Unless I have misread something. These are exactly the same as @poweroff and @restore.

@suspend place the device in a low power state.  
Enable wakeup events.

Can we use this for cases when we need low power but haven't stopped the cpu? I think so.

- + \* @freeze: Hibernation-specific, executed before creating a hibernation image.
- + \* Quiesce operations so that a consistent image can be created, but do NOT
- + \* otherwise put the device into a low power device state and do NOT emit
- + \* system wakeup events. Save in main memory the device settings to be
- + \* used by @restore() during the subsequent resume from hibernation or by
- + \* the subsequent @thaw(), if the creation of the image or the restoration
- + \* of main memory contents from it fails.
- + \*
- + \* @thaw: Hibernation-specific, executed after creating a hibernation image OR
- + \* if the creation of the image fails. Also executed after a failing
- + \* attempt to restore the contents of main memory from such an image.
- + \* Undo the changes made by the preceding @freeze(), so the device can be
- + \* operated in the same way as immediately before the call to @freeze().

Just @detach/@reattach.

@detach Detach the driver from the hardware, while keeping the driver instance for the hardware alive.

Essentially this is what the shutdown method is today.  
Except for being ready for a reattach.

@reattach  
See if the hardware for the driver is present and reclaim it and bring it up to speed for processing requests.

- + \* @poweroff: Hibernation-specific, executed after saving a hibernation image.
- + \* Quiesce the device, put it into a low power state appropriate for the
- + \* upcoming system state (such as PCI\_D3hot), and enable wakeup events as
- + \* appropriate.
- + \*
- + \* @restore: Hibernation-specific, executed after restoring the contents of main

Re: [PATCH -mm] kexec jump -v9

Re: [PATCH -mm] kexec jump -v9

- + \* memory from a hibernation image. Driver starts working again,
- + \* responding to hardware events and software requests. Drivers may NOT
- + \* make ANY assumptions about the hardware state right prior to @restore().
- + \* On most platforms, there are no restrictions on availability of
- + \* resources like clocks during @restore().
- + \*

If we have events we care about we just need to do:  
reattach(); suspend(); It is all the same from the point of view of  
the device. Not the system but the device.

Eric

--

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

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

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