System Suspend Code Flows

Author: Rafael J. Wysocki <rafael.j.wysocki@intel.com>

At least one global system-wide transition needs to be carried out for the system to get from the working state into one of the supported :doc:`sleep states <sleep-states>`. Hibernation requires more than one transition to occur for this purpose, but the other sleep states, commonly referred to as *system-wide suspend* (or simply *system suspend*) states, need only one.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\pm\(linux-master) (Documentation) (admin-guide) (pm) suspend-flows.rst`, line 12); *backlink*

Unknown interpreted text role "doc".

For those sleep states, the transition from the working state of the system into the target sleep state is referred to as *system suspend* too (in the majority of cases, whether this means a transition or a sleep state of the system should be clear from the context) and the transition back from the sleep state into the working state is referred to as *system resume*.

The kernel code flows associated with the suspend and resume transitions for different sleep states of the system are quite similar, but there are some significant differences between the :ref:`suspend-to-idle <s2idle>` code flows and the code flows related to the :ref:`suspend-to-RAM <s2ram>` and :ref:`standby <standby>` sleep states.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\pm\(linux-master) (Documentation) (admin-guide) (pm) suspend-flows.rst`, line 25); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\pm\(linux-master) (Documentation) (admin-guide) (pm) suspend-flows.rst`, line 25); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\pm\(linux-master) (Documentation) (admin-guide) (pm) suspend-flows.rst`, line 25); *backlink*

Unknown interpreted text role "ref".

The :ref:`suspend-to-RAM <s2ram>` and :ref:`standby <standby>` sleep states cannot be implemented without platform support and the difference between them boils down to the platform-specific actions carried out by the suspend and resume hooks that need to be provided by the platform driver to make them available. Apart from that, the suspend and resume code flows for these sleep states are mostly identical, so they both together will be referred to as *platform-dependent suspend* states in what follows.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\pm\(linux-master) (Documentation) (admin-guide) (pm) suspend-flows.rst`, line 31); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\pm\(linux-master) (Documentation) (admin-guide) (pm) suspend-flows.rst`, line 31); *backlink*

Unknown interpreted text role "ref".

Suspend-to-idle Suspend Code Flow

The following steps are taken in order to transition the system from the working state to the :ref:`suspend-to-idle <s2idle>` sleep state:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-`

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\pm\(linux-master)(Documentation)(admin-guide)(pm)suspend-flows.rst, line 45); *backlink*

Unknown interpreted text role "ref".

1. Invoking system-wide suspend notifiers.

   Kernel subsystems can register callbacks to be invoked when the suspend transition is about to occur and when the resume transition has finished.

   That allows them to prepare for the change of the system state and to clean up after getting back to the working state.

2. Freezing tasks.

   Tasks are frozen primarily in order to avoid unchecked hardware accesses from user space through MMIO regions or I/O registers exposed directly to it and to prevent user space from entering the kernel while the next step of the transition is in progress (which might have been problematic for various reasons).

   All user space tasks are intercepted as though they were sent a signal and put into uninterruptible sleep until the end of the subsequent system resume transition.

   The kernel threads that choose to be frozen during system suspend for specific reasons are frozen subsequently, but they are not intercepted. Instead, they are expected to periodically check whether or not they need to be frozen and to put themselves into uninterruptible sleep if so. [Note, however, that kernel threads can use locking and other concurrency controls available in kernel space to synchronize themselves with system suspend and resume, which can be much more precise than the freezing, so the latter is not a recommended option for kernel threads.]

3. Suspending devices and reconfiguring IRQs.

   Devices are suspended in four phases called *prepare*, *suspend*, *late suspend* and *noirq suspend* (see :ref:`driverapi_pm_devices` for more information on what exactly happens in each phase).

   System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\pm\(linux-master)(Documentation)(admin-guide)(pm)suspend-flows.rst, line 79); *backlink*

   Unknown interpreted text role "ref".

   Every device is visited in each phase, but typically it is not physically accessed in more than two of them.

   The runtime PM API is disabled for every device during the *late* suspend phase and high-level ("action") interrupt handlers are prevented from being invoked before the *noirq* suspend phase.

   Interrupts are still handled after that, but they are only acknowledged to interrupt controllers without performing any device-specific actions that would be triggered in the working state of the system (those actions are deferred till the subsequent system resume transition as described below).

   IRQs associated with system wakeup devices are "armed" so that the resume transition of the system is started when one of them signals an event.

4. Freezing the scheduler tick and suspending timekeeping.

   When all devices have been suspended, CPUs enter the idle loop and are put into the deepest available idle state. While doing that, each of them "freezes" its own scheduler tick so that the timer events associated with the tick do not occur until the CPU is woken up by another interrupt source.

   The last CPU to enter the idle state also stops the timekeeping which (among other things) prevents high resolution timers from triggering going forward until the first CPU that is woken up restarts the timekeeping. That allows the CPUs to stay in the deep idle state relatively long in one go.

   From this point on, the CPUs can only be woken up by non-timer hardware interrupts. If that happens, they go back to the idle state unless the interrupt that woke up one of them comes from an IRQ that has been armed for system wakeup, in which case the system resume transition is started.

Suspend-to-idle Resume Code Flow

The following steps are taken in order to transition the system from the :ref:`suspend-to-idle <s2idle>` sleep state into the working state:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\pm\(linux-master)(Documentation)(admin-guide)(pm)suspend-flows.rst, line 123); *backlink*

Unknown interpreted text role "ref".

1. Resuming timekeeping and unfreezing the scheduler tick.

   When one of the CPUs is woken up (by a non-timer hardware interrupt), it leaves the idle state entered in the last step of the preceding suspend transition, restarts the timekeeping (unless it has been restarted already by another CPU that woke up earlier) and the scheduler tick on that CPU is unfrozen.

   If the interrupt that has woken up the CPU was armed for system wakeup, the system resume transition begins.

2. Resuming devices and restoring the working-state configuration of IRQs.

   Devices are resumed in four phases called *noirq resume*, *early resume*, *resume* and *complete* (see :ref:`driverapi_pm_devices` for more information on what exactly happens in each phase).

   System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\pm\(linux-master) (Documentation) (admin-guide) (pm) suspend-flows.rst`, **line 139**); ***backlink***

   Unknown interpreted text role "ref".

   Every device is visited in each phase, but typically it is not physically accessed in more than two of them.

   The working-state configuration of IRQs is restored after the *noirq* resume phase and the runtime PM API is re-enabled for every device whose driver supports it during the *early* resume phase.

3. Thawing tasks.

   Tasks frozen in step 2 of the preceding suspend transition are "thawed", which means that they are woken up from the uninterruptible sleep that they went into at that time and user space tasks are allowed to exit the kernel.

4. Invoking system-wide resume notifiers.

   This is analogous to step 1 of the suspend transition and the same set of callbacks is invoked at this point, but a different "notification type" parameter value is passed to them.

Platform-dependent Suspend Code Flow

The following steps are taken in order to transition the system from the working state to platform-dependent suspend state:

1. Invoking system-wide suspend notifiers.

   This step is the same as step 1 of the suspend-to-idle suspend transition described above.

2. Freezing tasks.

   This step is the same as step 2 of the suspend-to-idle suspend transition described above.

3. Suspending devices and reconfiguring IRQs.

   This step is analogous to step 3 of the suspend-to-idle suspend transition described above, but the arming of IRQs for system wakeup generally does not have any effect on the platform.

   There are platforms that can go into a very deep low-power state internally when all CPUs in them are in sufficiently deep idle states and all I/O devices have been put into low-power states. On those platforms, suspend-to-idle can reduce system power very effectively.

   On the other platforms, however, low-level components (like interrupt controllers) need to be turned off in a platform-specific way (implemented in the hooks provided by the platform driver) to achieve comparable power reduction.

   That usually prevents in-band hardware interrupts from waking up the system, which must be done in a special platform-dependent way. Then, the configuration of system wakeup sources usually starts when system wakeup devices are suspended and is finalized by the platform suspend hooks later on.

4. Disabling non-boot CPUs.

   On some platforms the suspend hooks mentioned above must run in a one-CPU configuration of the system (in particular, the hardware cannot be accessed by any code running in parallel with the platform suspend hooks that may, and often do, trap into the platform firmware in order to finalize the suspend transition).

   For this reason, the CPU offline/online (CPU hotplug) framework is used to take all of the CPUs in the system, except for one (the boot CPU), offline (typically, the CPUs that have been taken offline go into deep idle states).

   This means that all tasks are migrated away from those CPUs and all IRQs are rerouted to the only CPU that remains online.

5. Suspending core system components.

This prepares the core system components for (possibly) losing power going forward and suspends the timekeeping.

6. Platform-specific power removal.

   This is expected to remove power from all of the system components except for the memory controller and RAM (in order to preserve the contents of the latter) and some devices designated for system wakeup.

   In many cases control is passed to the platform firmware which is expected to finalize the suspend transition as needed.

Platform-dependent Resume Code Flow

The following steps are taken in order to transition the system from a platform-dependent suspend state into the working state:

1. Platform-specific system wakeup.

   The platform is woken up by a signal from one of the designated system wakeup devices (which need not be an in-band hardware interrupt) and control is passed back to the kernel (the working configuration of the platform may need to be restored by the platform firmware before the kernel gets control again).

2. Resuming core system components.

   The suspend-time configuration of the core system components is restored and the timekeeping is resumed.

3. Re-enabling non-boot CPUs.

   The CPUs disabled in step 4 of the preceding suspend transition are taken back online and their suspend-time configuration is restored.

4. Resuming devices and restoring the working-state configuration of IRQs.

   This step is the same as step 2 of the suspend-to-idle suspend transition described above.

5. Thawing tasks.

   This step is the same as step 3 of the suspend-to-idle suspend transition described above.

6. Invoking system-wide resume notifiers.

   This step is the same as step 4 of the suspend-to-idle suspend transition described above.