PM Quality Of Service Interface

This interface provides a kernel and user mode interface for registering performance expectations by drivers, subsystems and user space applications on one of the parameters.

Two different PM QoS frameworks are available:

- CPU latency QoS.
- The per-device PM QoS framework provides the API to manage the per-device latency constraints and PM QoS flags.

The latency unit used in the PM QoS framework is the microsecond (usec).

1. PM QoS framework

A global list of CPU latency QoS requests is maintained along with an aggregated (effective) target value. The aggregated target value is updated with changes to the request list or elements of the list. For CPU latency QoS, the aggregated target value is simply the min of the request values held in the list elements.

Note: the aggregated target value is implemented as an atomic variable so that reading the aggregated value does not require any locking mechanism.

From kernel space the use of this interface is simple:

void cpu_latency_qos_add_request(handle, target_value):
: Will insert an element into the CPU latency QoS list with the target value. Upon change to this list the new target is recomputed and any registered notifiers are called only if the target value is now different. Clients of PM QoS need to save the returned handle for future use in other PM QoS API functions.

void cpu_latency_qos_update_request(handle, new_target_value):
: Will update the list element pointed to by the handle with the new target value and recompute the new aggregated target, calling the notification tree if the target is changed.

void cpu_latency_qos_remove_request(handle):
: Will remove the element. After removal it will update the aggregate target and call the notification tree if the target was changed as a result of removing the request.

int cpu_latency_qos_limit():
: Returns the aggregated value for the CPU latency QoS.

int cpu_latency_qos_request_active(handle):
: Returns if the request is still active, i.e. it has not been removed from the CPU latency QoS list.

int cpu_latency_qos_add_notifier(notifier):
: Adds a notification callback function to the CPU latency QoS. The callback is called when the aggregated value for the CPU latency QoS is changed.

int cpu_latency_qos_remove_notifier(notifier):
: Removes the notification callback function from the CPU latency QoS.

From user space:

The infrastructure exposes one device node, /dev/cpu_dma_latency, for the CPU latency QoS.

Only processes can register a PM QoS request. To provide for automatic cleanup of a process, the interface requires the process to register its parameter requests as follows.

To register the default PM QoS target for the CPU latency QoS, the process must open /dev/cpu_dma_latency.

As long as the device node is held open that process has a registered request on the parameter.

To change the requested target value, the process needs to write an s32 value to the open device node. Alternatively, it can write a hex string for the value using the 10 char long format e.g. "0x12345678". This translates to a cpu_latency_qos_update_request() call.

To remove the user mode request for a target value simply close the device node.

2. PM QoS per-device latency and flags framework

For each device, there are three lists of PM QoS requests. Two of them are maintained along with the aggregated targets of resume latency and active state latency tolerance (in microseconds) and the third one is for PM QoS flags. Values are updated in response to changes of the request list.

The target values of resume latency and active state latency tolerance are simply the minimum of the request values held in the parameter list elements. The PM QoS flags aggregate value is a gather (bitwise OR) of all list elements' values. One device PM QoS flag is defined currently: PM_QOS_FLAG_NO_POWER_OFF.

Note: The aggregated target values are implemented in such a way that reading the aggregated value does not require any locking mechanism.

From kernel mode the use of this interface is the following:

int dev_pm_qos_add_request(device, handle, type, value):

Will insert an element into the list for that identified device with the target value. Upon change to this list the new target is recomputed and any registered notifiers are called only if the target value is now different. Clients of dev_pm_qos need to save the handle for future use in other dev_pm_qos API functions.

int dev_pm_qos_update_request(handle, new_value):

Will update the list element pointed to by the handle with the new target value and recompute the new aggregated target, calling the notification trees if the target is changed.

int dev_pm_qos_remove_request(handle):

Will remove the element. After removal it will update the aggregate target and call the notification trees if the target was changed as a result of removing the request.

s32 dev_pm_qos_read_value(device, type):

Returns the aggregated value for a given device's constraints list.

enum pm_qos_flags_status dev_pm_qos_flags(device, mask)

Check PM QoS flags of the given device against the given mask of flags. The meaning of the return values is as follows:

PM_QOS_FLAGS_ALL:
All flags from the mask are set
PM_QOS_FLAGS_SOME:
Some flags from the mask are set
PM_QOS_FLAGS_NONE:
No flags from the mask are set
PM_QOS_FLAGS_UNDEFINED:
The device's PM QoS structure has not been initialized or the list of requests is empty.

int dev_pm_qos_add_ancestor_request(dev, handle, type, value)

Add a PM QoS request for the first direct ancestor of the given device whose power.ignore_children flag is unset (for DEV_PM_QOS_RESUME_LATENCY requests) or whose power.set_latency_tolerance callback pointer is not NULL (for DEV_PM_QOS_LATENCY_TOLERANCE requests).

int dev_pm_qos_expose_latency_limit(device, value)

Add a request to the device's PM QoS list of resume latency constraints and create a sysfs attribute pm_qos_resume_latency_us under the device's power directory allowing user space to manipulate that request.

void dev_pm_qos_hide_latency_limit(device)

Drop the request added by dev_pm_qos_expose_latency_limit() from the device's PM QoS list of resume latency constraints and remove sysfs attribute pm_qos_resume_latency_us from the device's power directory.

int dev_pm_qos_expose_flags(device, value)

Add a request to the device's PM QoS list of flags and create sysfs attribute pm_qos_no_power_off under the device's power directory allowing user space to change the value of the PM_QOS_FLAG_NO_POWER_OFF flag.

void dev_pm_qos_hide_flags(device)

Drop the request added by dev_pm_qos_expose_flags() from the device's PM QoS list of flags and remove sysfs attribute pm_qos_no_power_off from the device's power directory.

Notification mechanisms:

The per-device PM QoS framework has a per-device notification tree.

int dev_pm_qos_add_notifier(device, notifier, type):

Adds a notification callback function for the device for a particular request type.

The callback is called when the aggregated value of the device constraints list is changed.

int dev_pm_qos_remove_notifier(device, notifier, type):

Removes the notification callback function for the device.

Active state latency tolerance

This device PM QoS type is used to support systems in which hardware may switch to energy-saving operation modes on the fly. In those systems, if the operation mode chosen by the hardware attempts to save energy in an overly aggressive way, it may cause excess latencies to be visible to software, causing it to miss certain protocol requirements or target frame or sample rates etc.

If there is a latency tolerance control mechanism for a given device available to software, the .set_latency_tolerance callback in that device's dev_pm_info structure should be populated. The routine pointed to by it is should implement whatever is necessary to

transfer the effective requirement value to the hardware.

Whenever the effective latency tolerance changes for the device, its .set_latency_tolerance() callback will be executed and the effective value will be passed to it. If that value is negative, which means that the list of latency tolerance requirements for the device is empty, the callback is expected to switch the underlying hardware latency tolerance control mechanism to an autonomous mode if available. If that value is PM_QOS_LATENCY_ANY, in turn, and the hardware supports a special "no requirement" setting, the callback is expected to use it. That allows software to prevent the hardware from automatically updating the device's latency tolerance in response to its power state changes (e.g. during transitions from D3cold to D0), which generally may be done in the autonomous latency tolerance control mode.

If .set_latency_tolerance() is present for the device, sysfs attribute pm_qos_latency_tolerance_us will be present in the devivce's power directory. Then, user space can use that attribute to specify its latency tolerance requirement for the device, if any. Writing "any" to it means "no requirement, but do not let the hardware control latency tolerance" and writing "auto" to it allows the hardware to be switched to the autonomous mode if there are no other requirements from the kernel side in the device's list.

Kernel code can use the functions described above along with the DEV_PM_QOS_LATENCY_TOLERANCE device PM QoS type to add, remove and update latency tolerance requirements for devices.