Device PM QoS and Generic PM Domains

Rafael J. Wysocki

Intel Open Source Technology Center

November 7, 2012

Outline

INTEL OPEN SOURCE TECHNOLOGY CENTER

Holy Grail of Power Management

Provide the desired (or required) functionality and performance using only as much energy as absolutely necessary.

Holy Grail of Power Management

Provide the desired (or required) functionality and performance using only as much energy as absolutely necessary.

This means

All hardware components in use / not in use in the lowest power states sufficiently functional / allowed, respectively.

Holy Grail of Power Management

Provide the desired (or required) functionality and performance using only as much energy as absolutely necessary.

This means

All hardware components in use / not in use in the lowest power states sufficiently functional / allowed, respectively.

Note

We may not be allowed to go too far with power management for regulatory, real-time response, etc., reasons.

Capabilities vs Expectations

Hardware capabilities	**Expectations/Requirements**
Can it work with reduced performance and/or functionality using less power?	Sustain specific throughput and/or efficiency, provide specific functionality.
What can be done with it when not in use?	Ensure specific time of reaction to certain events.

Capabilities vs Expectations

Hardware capabilities	Expectations/Requirements
Can it work with reduced performance and/or functionality using less power?	Sustain specific throughput and/or efficiency, provide specific functionality.
What can be done with it when not in use?	Ensure specific time of reaction to certain events.

Information about hardware capabilities comes from its specifications (at least in theory) and may be treated as static data.

Capabilities vs Expectations

Hardware capabilities	Expectations/Requirements
Can it work with reduced performance and/or functionality using less power?	Sustain specific throughput and/or efficiency, provide specific functionality.
What can be done with it when not in use?	Ensure specific time of reaction to certain events.

Information about hardware capabilities comes from its specifications (at least in theory) and may be treated as static data.

User expectations may change while the system is running.

Hardware Metastates

Active : Can process (user) data, may be idle.

Suspended : Cannot process (user) data.

Hardware Metastates

Active : Can process (user) data, may be idle.

Suspended : Cannot process (user) data.

They are well-defined for all hardware components, regardless of the type (including CPU cores and memory).

Hardware Metastates

Active : Can process (user) data, may be idle.

Suspended : Cannot process (user) data.

They are well-defined for all hardware components, regardless of the type (including CPU cores and memory).

The Linux* kernel's runtime PM framework tracks transitions between the two metastates (for I/O devices).

Hardware Metastates

Active : Can process (user) data, may be idle.

Suspended : Cannot process (user) data.

They are well-defined for all hardware components, regardless of the type (including CPU cores and memory).

The Linux* kernel's runtime PM framework tracks transitions between the two metastates (for I/O devices).

Each metastate can cover multiple physical "power" states.

Low-Power States

States of a hardware component in the suspended metastate

Characterized by:

- Energy usage (power draw).
- Exit latency.
- Break even time.

Low-Power States

States of a hardware component in the suspended metastate

Characterized by:

- Energy usage (power draw).
- Exit latency.
- Break even time.

For CPU cores (and sockets) they are referred to as C-states (idle states).

Low-Power States

States of a hardware component in the suspended metastate

Characterized by:

- Energy usage (power draw).
- Exit latency.
- Break even time.

For CPU cores (and sockets) they are referred to as C-states (idle states).

ACPI and PCI use 4 low-power states, $D1$, $D2$, $D3_{hot}$, $D3_{cold}$ (no power). The active metastate is represented by the standard full-power state $D0$.

Performance States

States of a hardware component in the active metastate

Characterized by:

- Clock rate(s) (frequency).
- Voltage.

Performance States

States of a hardware component in the active metastate

Characterized by:

- Clock rate(s) (frequency).
- Voltage.

For CPU cores (and sockets) they are referred to as P-states.

Performance States

States of a hardware component in the active metastate

Characterized by:

- Clock rate(s) (frequency).
- Voltage.

For CPU cores (and sockets) they are referred to as P-states.

Dynamic Voltage and Frequency Scaling (DVFS)

Switching between different performance states.

Correlations Between Hardware Components

Both low-power and performance states of hardware components may be internal or dependent on the states of other hardware components.

Correlations Between Hardware Components

Both low-power and performance states of hardware components may be internal or dependent on the states of other hardware components.

Power domain

Set of hardware components whose power settings are mutually correlated.

Correlations Between Hardware Components

Both low-power and performance states of hardware components may be internal or dependent on the states of other hardware components.

Power domain

Set of hardware components whose power settings are mutually correlated.

Clock/Voltage domain

Clock rates/voltages of hardware components in the domain cannot be changed independently.

Correlations Between Hardware Components

Both low-power and performance states of hardware components may be internal or dependent on the states of other hardware components.

Power domain

Set of hardware components whose power settings are mutually correlated.

Clock/Voltage domain

Clock rates/voltages of hardware components in the domain cannot be changed independently.

Quite often there is a shared "power switch" (or a number of them).

How to Choose Low-Power and Performance States?

How to Choose Low-Power and Performance States?

Extreme strategies

1. Go for the maximum performance – usually not sustainable in the long run (expensive, not environmentally friendly etc.).
2. Go for the maximum energy savings – may not meet requirements and/or expectations.

How to Choose Low-Power and Performance States?

Extreme strategies

1. Go for the maximum performance – usually not sustainable in the long run (expensive, not environmentally friendly etc.).
2. Go for the maximum energy savings – may not meet requirements and/or expectations.

Quite obviously, a balanced strategy is needed.

How to Choose Low-Power and Performance States?

Extreme strategies

1. Go for the maximum performance – usually not sustainable in the long run (expensive, not environmentally friendly etc.).
2. Go for the maximum energy savings – may not meet requirements and/or expectations.

Quite obviously, a balanced strategy is needed.

However, for every balanced strategy to work, there must be a way to specify the requirements and/or expectations.

What is PM QoS?

Power Management Quality of Service

Set of interfaces allowing requirements related to power management to be passed to the relevant kernel subsystems.

What is PM QoS?

Power Management Quality of Service

Set of interfaces allowing requirements related to power management to be passed to the relevant kernel subsystems.

Constraints are represented by lists of values from different sources (including user space).

What is PM QoS?

Power Management Quality of Service

Set of interfaces allowing requirements related to power management to be passed to the relevant kernel subsystems.

Constraints are represented by lists of values from different sources (including user space).

For each constraint those values are combined to produce an effective requirement (it may be the minimum, the sum, the bitwise OR of them etc.).

Global PM QoS

PM QoS classes

`CPU_DMA_LATENCY`, `NETWORK_LATENCY`, `NETWORK_THROUGHPUT`.

Global PM QoS

PM QoS classes

`CPU_DMA_LATENCY`, `NETWORK_LATENCY`, `NETWORK_THROUGHPUT`.

- For each class there is a special device file in `/dev` allowing user space to specify its requirement in that class.
- Kernel subsystems specify their values by calling

```
void pm_qos_add_request(struct pm_qos_request *req,
                        int pm_qos_class,
                        s32 value);
```

- They may be modified or withdrawn with the help of, respectively, `pm_qos_update_request()` and `pm_qos_remove_request()`.

Global PM QoS Continued

Requests with automatic expiration timeout

```
void pm_qos_update_request_timeout(struct pm_qos_request *req,
                                   s32 new_value,
                                   unsigned long timeout_us);
```

Global PM QoS Continued

Requests with automatic expiration timeout

```
void pm_qos_update_request_timeout(struct pm_qos_request *req,
                                   s32 new_value,
                                   unsigned long timeout_us);
```

Checking effective requirements

```
int pm_qos_request(int pm_qos_class);
```

Global PM QoS Continued

Requests with automatic expiration timeout

```
void pm_qos_update_request_timeout(struct pm_qos_request *req,
                                   s32 new_value,
                                   unsigned long timeout_us);
```

Checking effective requirements

```
int pm_qos_request(int pm_qos_class);
```

For the "latency" classes the values mean maximum allowed latency in microseconds. However, it is not precisely specified what latency they mean.

Device PM QoS

Request types (v3.8 material)

`DEV_PM_QOS_LATENCY`, `DEV_PM_QOS_FLAGS`

Device PM QoS

Request types (v3.8 material)

`DEV_PM_QOS_LATENCY`, `DEV_PM_QOS_FLAGS`

Adding, modifying, removing requests

```
int dev_pm_qos_add_request(struct device *dev,
                           struct dev_pm_qos_request *req,
                           enum dev_pm_qos_req_type type,
                           s32 value);
int dev_pm_qos_update_request(struct dev_pm_qos_request *req, s32 new_value);
int dev_pm_qos_update_flags(struct device *dev, s32 mask, bool set);
int dev_pm_qos_remove_request(struct dev_pm_qos_request *req);
int dev_pm_qos_add_ancestor_request(struct device *dev,
                                    struct dev_pm_qos_request *req, s32 value);
```

Device PM QoS Continued

Exposing/hiding

```
int dev_pm_qos_expose_latency_limit(struct device *dev, s32 value);
void dev_pm_qos_hide_latency_limit(struct device *dev);
int dev_pm_qos_expose_flags(struct device *dev, s32 value);
void dev_pm_qos_hide_flags(struct device *dev);
```

Device PM QoS Continued

Exposing/hiding

```
int dev_pm_qos_expose_latency_limit(struct device *dev, s32 value);
void dev_pm_qos_hide_latency_limit(struct device *dev);
int dev_pm_qos_expose_flags(struct device *dev, s32 value);
void dev_pm_qos_hide_flags(struct device *dev);
```

Checking effective requirements

```
s32 dev_pm_qos_read_value(struct device *dev);
enum pm_qos_flags_status dev_pm_qos_flags(struct device *dev, s32 mask);
```

Device PM QoS Continued

Exposing/hiding

```
int dev_pm_qos_expose_latency_limit(struct device *dev, s32 value);
void dev_pm_qos_hide_latency_limit(struct device *dev);
int dev_pm_qos_expose_flags(struct device *dev, s32 value);
void dev_pm_qos_hide_flags(struct device *dev);
```

Checking effective requirements

```
s32 dev_pm_qos_read_value(struct device *dev);
enum pm_qos_flags_status dev_pm_qos_flags(struct device *dev, s32 mask);
```

Defined flags (v3.8 material)

`PM_QOS_FLAG_NO_POWER_OFF`, `PM_QOS_FLAG_REMOTE_WAKEUP`

Generic PM Domains Framework

Covers the case of multiple shared power on/off "switches" that can be manipulated through register writes.

Generic PM Domains Framework

Covers the case of multiple shared power on/off "switches" that can be manipulated through register writes.

A domain consists of several devices (possibly including CPU cores) that share a power on/off "switch". Domain hierarchies are supported.

Generic PM Domains Framework

Covers the case of multiple shared power on/off "switches" that can be manipulated through register writes.

A domain consists of several devices (possibly including CPU cores) that share a power on/off "switch". Domain hierarchies are supported.

Functions for adding/removing devices to/from domains are provided and device PM callbacks are wrapped in code that takes the power on/off "switch" sharing into account.

Generic PM Domains Framework

Covers the case of multiple shared power on/off "switches" that can be manipulated through register writes.

A domain consists of several devices (possibly including CPU cores) that share a power on/off "switch". Domain hierarchies are supported.

Functions for adding/removing devices to/from domains are provided and device PM callbacks are wrapped in code that takes the power on/off "switch" sharing into account.

Device PM QoS latency constraints are taken into consideration.

Generic PM Domains and Latency Constraints

4 latency parameters for each device

Stop latency (l_{stop}), start latency (l_{start}), state saving latency (l_{save}), state restoration latency ($l_{restore}$).

The domain runtime suspend callback performs the "stop" operation if

$$l_{start} + l_{stop} < c$$

(for the given device) where

c – effective latency constraint (takes children into account)

Generic PM Domains and Latency Constraints Continued

When all devices in a domain have been stopped, the power "switch" is turned off if every device in the domain can be "off" long enough for the "switch" to go through a full cycle ("off" and "on").

Generic PM Domains and Latency Constraints Continued

When all devices in a domain have been stopped, the power "switch" is turned off if every device in the domain can be "off" long enough for the "switch" to go through a full cycle ("off" and "on").

2 latency parameters of the power "switch"

"off" latency (s_{off}), "on" latency (s_{on})

Generic PM Domains and Latency Constraints Continued

When all devices in a domain have been stopped, the power "switch" is turned off if every device in the domain can be "off" long enough for the "switch" to go through a full cycle ("off" and "on").

2 latency parameters of the power "switch"

"off" latency (s_{off}), "on" latency (s_{on})

Inequality checked for every device in the domain

$$\underbrace{\sum_{devices} l_{save}}_{unnecessary?} + s_{off} + s_{on} + l_{restore} + l_{start} < c$$

Generic PM Domains and PM QoS Flags

The power "switch" will not be turned off if `PM_QOS_FLAG_NO_POWER_OFF` or `PM_QOS_FLAG_REMOTE_WAKEUP` is effectively set for at least one device in the domain.

Generic PM Domains and PM QoS Flags

The power "switch" will not be turned off if `PM_QOS_FLAG_NO_POWER_OFF` or `PM_QOS_FLAG_REMOTE_WAKEUP` is effectively set for at least one device in the domain.

That may be overkill in some situations, but they are difficult to take into account in the code.

Active State PM QoS

Turns out to be very hard to implement in a sufficiently generic way.

Active State PM QoS

Turns out to be very hard to implement in a sufficiently generic way.

Example problems

- How to represent performance? Throughput or percentage of the peak? Abstract units??
- If throughput, then
  - Raw or user-visible? What units (e.g. megabytes or frames per second)?
  - We may have to deal with chains of devices, so how to represent them?
- Is there any measure suitable for all kinds of hardware components?
- Can it be portable between different platforms/systems?
- How to combine requests?

Conclusion

- PM QoS is essential for all balanced runtime PM strategies.

Conclusion

- PM QoS is essential for all balanced runtime PM strategies.
- Linux kernel developers realize its importance and therefore it has been under active development for quite some time (in particular, device PM QoS flags are a very recent addition).

Conclusion

- PM QoS is essential for all balanced runtime PM strategies.
- Linux kernel developers realize its importance and therefore it has been under active development for quite some time (in particular, device PM QoS flags are a very recent addition).
- It is supported by some kernel core PM code, like the generic PM domains framework, and more support is coming.

Conclusion

- PM QoS is essential for all balanced runtime PM strategies.
- Linux kernel developers realize its importance and therefore it has been under active development for quite some time (in particular, device PM QoS flags are a very recent addition).
- It is supported by some kernel core PM code, like the generic PM domains framework, and more support is coming.
- Active state PM QoS is notoriously hard to implement. [Ideas?]

Conclusion

- PM QoS is essential for all balanced runtime PM strategies.
- Linux kernel developers realize its importance and therefore it has been under active development for quite some time (in particular, device PM QoS flags are a very recent addition).
- It is supported by some kernel core PM code, like the generic PM domains framework, and more support is coming.
- Active state PM QoS is notoriously hard to implement. [Ideas?]
- Nevertheless, some isolated attempts to use it have been made and more of them are likely to happen.

Conclusion

- PM QoS is essential for all balanced runtime PM strategies.
- Linux kernel developers realize its importance and therefore it has been under active development for quite some time (in particular, device PM QoS flags are a very recent addition).
- It is supported by some kernel core PM code, like the generic PM domains framework, and more support is coming.
- Active state PM QoS is notoriously hard to implement. [Ideas?]
- Nevertheless, some isolated attempts to use it have been made and more of them are likely to happen.

Questions?

INTEL OPEN SOURCE TECHNOLOGY CENTER

References

Sundar Iyer et al, *PM-QoS? Naah..It is PnP QoS*
(http://www.linuxplumbersconf.org/2012/wp-content/uploads/2012/09/2012-lpc-pmconstraints-PM-QoS-power-performance-iyer.pdf).

R. J. Wysocki, *Generic PM Domains and Platform Device Drivers*
(http://events.linuxfoundation.org/images/stories/pdf/lcjp2012_wysocki.pdf).

R. J. Wysocki, *Power Management Using PM Domains on SH7372*
(https://events.linuxfoundation.org/events/embedded-linux-conference-europe/wysocki).

R. J. Wysocki, *Runtime PM vs System Sleep*
(http://www.linuxplumbersconf.org/2011/ocw/system/presentations/27/original/system_sleep_vs_runtime_PM.pdf).

R. J. Wysocki, *Runtime Power Management Framework for I/O Devices in the Linux Kernel*
(http://events.linuxfoundation.org/slides/2010/linuxcon2010_wysocki.pdf).

Documentation and Source Code

- Documentation/power/devices.txt
- Documentation/power/runtime_pm.txt
- Documentation/power/pm_qos_interface.txt
- include/linux/device.h
- include/linux/pm.h
- include/linux/pm_domain.h
- include/linux/pm_qos.h
- include/linux/pm_runtime.h
- drivers/base/power/*
- kernel/power/qos.c

Legal Information

Thanks!

Thank you for attention!