Linux Kernel Power Management : An Overview

Thara Gopinath / Viresh Kumar

(Adapted from Introduction to Kernel Power Management

Introduction

- Goal of Power Management
  - consume as little power as possible in a given system state, configuration, use case

- Complex SoCs
  - Power management solution evolved over time

Static vs Dynamic PM

- Static PM
  - How to save power when system is inactive for long duration?
  - Linux kernel status – evolved
- Dynamic PM
  - System in active state.
  - Short pulses of inactivity
  - Linux kernel status - evolving

STATIC PM

Suspend Resume

- Generic
  - Freezing user space, Suspend time keeping
  - Putting i/o devices to low-power deep idle states
- Suspend to idle (“freeze”)
  - Minimal latency. Put processors in deepest cpuidle C-state
- Power-on suspend (“standby”)
  - Non-boot CPUs taken offline.
- Suspend to Ram (“mem”)
  - Memory in self refresh. Everything else powered off
- Suspend to Disk (“disk”)
  - Snapshot image of memory saved in persistent storage.
  - Memory powered down.
  - System powered down.
  - On wakeup boot from hibernation image
- echo mem > /sys/power/state

Suspend Resume: What subsystems should do?

- Register device hooks

```
struct dev_pm_ops {
        int (*prepare)(struct device *dev);
        void (*complete)(struct device *dev);
        int (*suspend)(struct device *dev);
        int (*resume)(struct device *dev);

             ....
        int (*suspend_late)(struct device
        *dev);  int (*resume_early)(struct
        device *dev);

        ...
    };
```

Suspend Resume: What subsystems should do?

- Enable Wakeups if needed
  - device_init_wakeup(dev, bool)
  - enable_irq_wake()
  - disable_irq_wake()

- When wakeup occurs (e.g. in ISR):
  - pm_wakeup_event()

- Wakeups can be enabled/disabled from user space as well
  - /sys/devices/.../power/wakeup

DYNAMIC PM

Dynamic PM: Active Power Management

- Use case based – system is running something
- Best possible performance with power savings - DVFS
- Frameworks
  - CPUfreq, Devfreq
  - OPP
  - Genpd (performance states)
  - Runtime PM
  - Clocks
  - Regulators

CPUFreq and DevFreq

- CPUFreq
  - DVFS for CPUs
  - Select best OPP for CPUs based on constraints and requirement
  - Governors
    - Schedutil , Performance, Powersave, ondemand, userspace
  - Generic driver: cpufreq-dt.c

- DevFreq
  - DVFS for devices
  - Select best OPP for devices based on constraints and requirement
  - Governors
    - Ondemand, performance, powersave, passive

Operating Performance Points (OPP)

- Frequency voltage tuples – performance states of a system

```
{     1.2 GHz,      1040 mv},
{   1.8 GHz,        1140 mv},
{   2.4 GHz,        1240 mv},
```

- Usually defined in device tree (for a cpu, gpu, device, etc.)
- Framework - backbone of DVFS.

Dynamic PM: Idle Power Management

- Periodic idling in between tasks
- Think in the range of milliseconds and microseconds rather than seconds.
- Frameworks
  - CPUIdle
  - Device Idle
    - Runtime PM
    - Generic Power Domain

CPU Idle

- C-States – ACPI term
  - Latency and power saving – defines depth
  - Defined in device tree
- Governors
  - menu, ladder
- Driver
  - Platform specific
  - “compatible string” used to match
  - Eg. drivers/cpuidle/cpuidle-arm.c

```
Idle-states {
        entry-method = "psci";
        CPU_SLEEP: cpu-sleep {
        compatible = "arm,idle-state";
        local-timer-stop;
        arm,psci-suspend-param =
        <0x0010000>;
        entry-latency-us = <700>;
        exit-latency-us = <250>;
        min-residency-us = <1000>;
        };
....}
```

CPU Idle : Kernel Mechanisms

- Frequently used governor – menu

- Chooses the C-State to enter based on
  - Latency limitations imposed by QoS framework
    - vs entry/exit time, min residency
  - Next predictable event (timers)
    - vs entry/exit time, min residency
  - Magic
    - multipliers evolved over time

- IRQ prediction (In progress)

Device Idle: Runtime PM

- Per Device Idle
- Driven by device
- No dependency taken care.
  - Except parent device
- Unlike system suspend no user-space idling

Runtime PM: What devices should do?

- Register Callbacks/Hooks

```
struct dev_pm_ops {
        ...
        int (*runtime_suspend)(struct device *dev);  int
        (*runtime_resume)(struct device *dev);  int
        (*runtime_idle)(struct device *dev);
};
```

- Request to be active
  - pm_runtime_get()
  - pm_runtime_get_sync()
- Request to be put to idle
  - pm_runtime_put()
  - pm_runtime_put_sync()
  - pm_runtime_put_autosuspend()

Runtime PM: Kernel Mechanisms

- Use Count based

- Use Count = 0
  - → runtime_suspend()
    - Prepare for suspend
    - Save context
    - Enable wake up

- Use Count = 1
  - → runtime_resume()
    - Restore context

Device Idle: Generic Power Domain

- Devices can be grouped into PDs
  - Physical grouping in SoC
  - Logical grouping by software
- Based on Runtime PM
- Defined as part of device tree

```
gcc: clock-controller@1800000 {
        compatible = "qcom,gcc-msm8916";
        #clock-cells = <1>;
        #reset-cells = <1>;
        #power-domain-cells = <1>;  reg =
        <0x1800000 0x80000>;
};
mdss: mdss@1a00000 {
        compatible = "qcom,mdss";
    ...
            power-domains = <&gcc
MDSS_GDSC>;
}
```

GenPD: What domain drivers should do?

- Register Power domains
  - pm_genpd_init()
  - of_genpd_add_provider_onecell()

- Specify the following hooks

```
struct generic_pm_domain {
        ...
        int (*power_off) (struct generic_pm_domain
        *domain);  int (*power_on) (struct
        generic_pm_domain *domain);
        int (*set_performance_state)(struct
        generic_pm_domain *genpd, unsigned int state);
        ...
```

GenPD: Kernel Mechanism

- Based on runtime pm
- Allows subdomains to be registered
- When all devices in domain are runtime suspended...
  - genpd->power_off()
- When first device in domain is runtime resumed...
  - genpd->power_on()
- When a device wants to update performance constraint on the power domain
  - dev_pm_genpd_set_performance_state()
- genpd governors
  - allow custom decision making before power gating
  - e.g. per-device QoS constraints

PM Quality of Service(QoS)

- Two Different classes

- System wide

  PM_QOS_CPU_DMA_LATENCY,
  PM_QOS_NETWORK_LATENCY,
  PM_QOS_NETWORK_THROUGHPUT,
  PM_QOS_MEMORY_BANDWIDTH,

- Device specific
  - DEV_PM_QOS_RESUME_LATENCY,
  - DEV_PM_QOS_LATENCY_TOLERANCE
  - CPUs are devices like others

References

1. https://baylibre.com/pub/conferences/2015/KernelRecipes/Hilman-Intro_Kernel_PM-KR2015.svg