**Linux Kernel Power Management – Internal Concepts & Senior Engineer Interview Topics**

**1. Power Management Concepts in the Linux Kernel**

* **System vs. Runtime Power Management**
  + System-level power management: Suspend, hibernate, shutdown
  + Runtime PM: Managing individual device power without suspending the entire system
* **ACPI vs. Device Tree in Power Management**
* **CPU Power Management**
  + CPUFreq (Dynamic frequency scaling)
  + CPUIdle (Low-power CPU idle states)
  + DVFS (Dynamic Voltage and Frequency Scaling)
  + P-states, C-states, and S-states (ACPI states)
* **Device Power Management**
  + Runtime PM framework (drivers/base/power/runtime.c)
  + PM domains (drivers/base/power/domain.c)
  + Generic PM (GenPD) for SoC components
  + Wakeup sources and IRQ wakeup handling
* **System Suspend & Resume**
  + Suspend-to-RAM (S3), Hibernate (S4)
  + Kernel suspend flow (suspend\_ops, suspend\_prepare, suspend\_finish)
  + Freezing and thawing tasks (freeze\_processes())
* **Memory Power Management**
  + Page cache optimization for power
  + DRAM self-refresh (Low Power DDR modes: LPDDR, LPDDR2, LPDDR4)

**2. Linux Kernel Power Management Frameworks**

* **PM QoS (Quality of Service)**
  + Latency constraints (/dev/cpu\_dma\_latency)
  + Performance vs. power trade-offs
* **Energy-Aware Scheduling (EAS)**
  + Integration with CPUFreq and CPUIdle
  + Energy Model (drivers/energy\_model/)
* **cpuidle & cpufreq Governors**
  + performance, ondemand, schedutil, conservative
  + Writing custom governors (drivers/cpufreq/)
* **Device Tree Bindings for Power Management**
  + Power domains (power-domains property in DT)
  + Regulator framework (regulator.0 for voltage control)
* **Runtime PM (drivers/base/power/runtime.c)**
  + pm\_runtime\_get\_sync(), pm\_runtime\_put\_sync()
  + Wakeup management and autosuspend

**3. Debugging & Profiling Power Issues**

* **Linux PM Debugging Interfaces**
  + /sys/power/state, /sys/devices/system/cpu/cpu\*/cpufreq/
  + Wakeup sources (/sys/kernel/debug/wakeup\_sources)
* **Ftrace & PM Debugging**
  + echo 1 > /sys/kernel/debug/tracing/events/power/enable
  + Tracing suspend/resume latency
* **powertop & pm-graph**
  + Identifying power-hungry processes
  + Analyzing CPU idle state residency
* **Dynamic Debug (dynamic\_debug for PM)**
  + dmesg | grep 'PM:'
* **Performance Monitoring with perf**
  + perf stat -a -e power/energy-cores/

**4. Interview Questions for a Senior Engineer Role**

**Conceptual Questions**

1. Explain the difference between system-wide suspend and runtime PM.
2. How does CPUFreq work? How is it different from CPUIdle?
3. What is the role of the regulator framework in Linux?
4. Explain the role of wakeup sources in suspend/resume.
5. How does the kernel handle power domain dependencies?

**Scenario-Based & Debugging Questions**

1. You observe high power consumption on an embedded board. How do you debug it?
2. CPUFreq governor selection: When would you use **schedutil** vs. **ondemand**?
3. A device fails to wake up from suspend. How would you debug it?
4. Why would a CPU refuse to enter deep idle states (C6/C7)?
5. How do you trace the suspend/resume flow in Linux?

**Kernel Code & Implementation Questions**

1. How would you add runtime PM support to a new driver?
2. What modifications are needed to add a new cpufreq governor?
3. Explain the sequence of function calls when a system enters **Suspend-to-RAM**.
4. How does the Linux kernel decide when to enable **autosuspend** for a device?
5. Given the pm\_runtime\_get\_sync() function, how does it interact with device drivers?

**Resources for Deep Dive**

* **Linux Kernel Documentation**
  + Documentation/power/
* **Kernel Source Code**
  + CPUFreq: drivers/cpufreq/
  + CPUIdle: drivers/cpuidle/
  + PM Core: drivers/base/power/
* **Tools**
  + powertop, perf, ftrace, pm-graph