Graceful Under Pressure: Prioritizing Shutdown to Protect Your Data in Embedded Systems

Oleksij Rempel – ore@pengutronix.de

https://www.pengutronix.de

my_self = kzalloc()

- Oleksij Rempel, Linux Kernel Hacker
- Expertise in: Medical, Industrial and Agricultural devices
- Addressing challenges: Limited CPU/bandwidth, power efficiency, diagnostic
- Prioritizing long-term sustainable, secure and Open Source Embedded Linux (mainline).

Power State Zero - It's All About Time (& load).

- Short interruptions can be potentially handled by power supply:
  - Microseconds – capacitors may handle it.
  - Milliseconds – larger capacitors and less load.
- Anything else need Battery, UPS, Generator, etc. and SW support.

Not All Power Solutions Fit All Sizes.

- Size: Big systems fit big solutions; small devices can't.
- Cost: High cost for big systems; impractical for small.
- Embedded systems usually do not have stable power supply. (Cars, tractors, systems emergency stop switches)

Why Can't You Just Use the 'Golden Brick'

- **Size Constraints**: Compact and lightweight design limits component choices.
- **Environmental Durability**: Must endure extreme temperatures, humidity, vibration, and dust.
- **Cost-effectiveness** leads to trade-offs in performance and quality.
- **Supply Chain Disruptions**: Alternative components due to supply issues can affect performance and reliability.

Impact on Embedded Designs

- Affected Systems: Any embedded design lacking its own battery backup is vulnerable to power fluctuations.
- Common Storage Types at Risk:
  - RAW NAND: Susceptible to corruption during power loss without proper management.
  - eMMC: May face issues if power fail management isn't robust.
  - SD Cards: Can experience data loss or corruption during unexpected power drops.

Current state

- Some modern NANDs and eMMCs claim to be hardened against this kinds of issues.
- Some system integrators have learned from past experiences. Hardware and software-based countermeasures are now included in new system requirements.
- An upstream solution is needed to prevent reliance on custom, proprietary hacks.

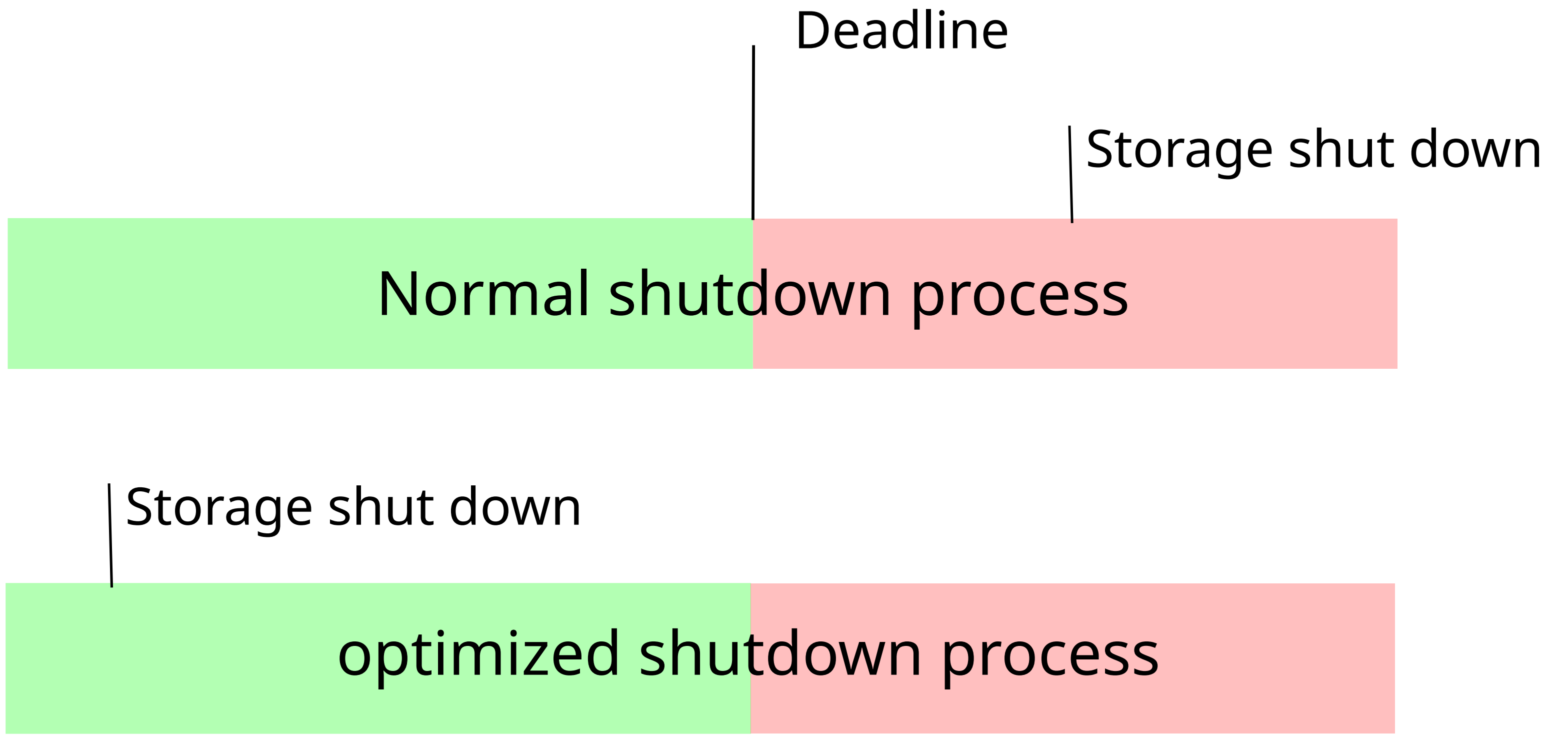
Real life example - NAND

- Automotive design is using raw NAND. It is older design, so older NANDs are used.
- Main requirement: system must be able to shutdown immediately.
- Problem: If system is writing to the NAND on power loss, corruption will occur. The system should stop writing and pull write protection pin.
- Hack is implemented in Linux kernel downstream.

Real life example - eMMC

- Similar story as with raw NAND with worse outcome – eMMCs was bricked.
- Solution, system should stop writing and send eMMC shutdown notification.

What is the problem?



Vendor Solutions for Power Failures

- Undervoltage Detection:
  - Use extra circuits to detect under-voltage and power off non-critical components (e.g., monitors).
- Custom Software:
  - Vendors modify/hack Linux kernel or use management co-processor to react on power drops. For example: safely shut down storage devices within short time window.

Initial Implementation for Upstream

- Use regulator framework to notify about system-critical events (upstream).
- Execute under-voltage hardware protection shutdown (upstream).
- Call eMMC shutdown with higher priority (not upstream). Needed to shutdown storage within 100ms.
- Save shutdown reason to RTC clock (not upstream).

Option 1 - Rework kernel/reboot.c

- https://lore.kernel.org/all/20231124145338.3112416-1-o.rempel@pengutronix.de/
- Implementation:
  - **Priority-Based Reverse Order Shutdown**: The function shuts down devices in reverse order of their initialization while also considering their assigned shutdown priority levels. This ensures that higher priority devices, such as storage, are shut down before lower priority ones.
  - **Inherited Priorities**: Devices inherit their shutdown priority from their parent devices to maintain a consistent and safe shutdown sequence.
  - **Priorities are statically assigned**. Currently only 2 prios, “storage” for eMMCs and “default” for all other devices.

Option 1 - Rework kernel/reboot.c

- Pros:
  - Integrated Solution: Directly modifies the existing reboot process in the kernel, ensuring a unified approach to shutdown priorities.
  - Centralized Control: Allows for global management of shutdown order, potentially improving consistency across different subsystems.
- Cons:
  - Complexity: Requires significant changes to the core kernel code, which could introduce new bugs or maintenance challenges.
  - Priority Assignment: Determining the correct priority for each component could be difficult and may require extensive testing and consensus

Option 2 - Use existing register_reboot_notifier()

- Implementation:
  - **Reboot Notifier Registration:** Drivers register reboot notifiers with priority by using register_reboot_notifier(), ensuring they are included in the reboot notifier list for shutdown.
  - **Kernel Shutdown Sequence**: When the system prepares to power off, **kernel_power_off()** is invoked, starting the shutdown process.
  - **Reboot Notifier Execution:** blocking_notifier_call_chain(&reboot_notifier_list, ...) is called before the general device shutdown, allowing registered drivers to execute critical code earlier than in Option 1.
  - **Driver Shutdown Sequence:** After the reboot notifiers are executed, the standard device shutdown process follows, shutting down drivers in reverse order of their registration as in Option 1.
  - Currently there is no interface to configure priority from user space or devicetree/ACPI/firmware :)

Option 2 - Use register_reboot_notifier()

- Pros:
  - Existing Mechanism: Leverages existing kernel infrastructure, reducing the need for major code changes.
  - Flexibility: Allows different drivers or subsystems to register their own priorities, making the system more modular.
- Cons:
  - Decentralized Management: Handling priorities across multiple subsystems could lead to inconsistencies or conflicts.
  - Priority Assignment: Similar to the first option, assigning priorities effectively can be challenging and may vary based on use cases.

Key Challenges in Both Options

- Priority Assignment
  - Assigning priorities via Device Tree is not feasible for systems using ACPI.
  - Assigning static, board-specific priorities in user space is cumbersome.
  - Changing default priority for storage devices, may break other systems.

Other options ?