Linaro Connect
MADRID 2024 | MAY 14-17 2024

Optimizing suspend/resume

Saravana Kannan (Google)

Key phases of suspend/resume

Suspend:

- suspend_enter
  - sync_filesystems
  - freeze_processes
- dpm_prepare
- dpm_suspend
- dpm_suspend_late
- dpm_suspend_noirq
- Power off CPUs

Resume:

- Power on CPUs
- dpm_resume_noirq
- dpm_resume_early
- dpm_resume
- dpm_complete
- thaw_processes

Optimizing suspend_enter

sync_filesystems()

- In systems where going in/out of suspend is frequent, syncing filesystems on every suspend is wasteful.
- On average 30ms to 66ms on phones and watches.
- Can be about 10-35% of the duration to suspend.
- Virtual Memory subsystem can periodically flush dirty pages
  - dirty_expire_centisecs - Maximum age of a page
  - dirty_writeback_centisecs - Flusher thread period
  - Keep in mind that these periods don’t include time in suspend

Solution: Disable sync on suspend using /sys/power/sync_on_suspend

Question: Can/should we make dirty_expire_centisecs take time in suspend into account?

Optimizing suspend_enter

freeze_processes()

What's going on with the 2nd half?

It can't be all just setting the flags for all the kernel threads, can it?

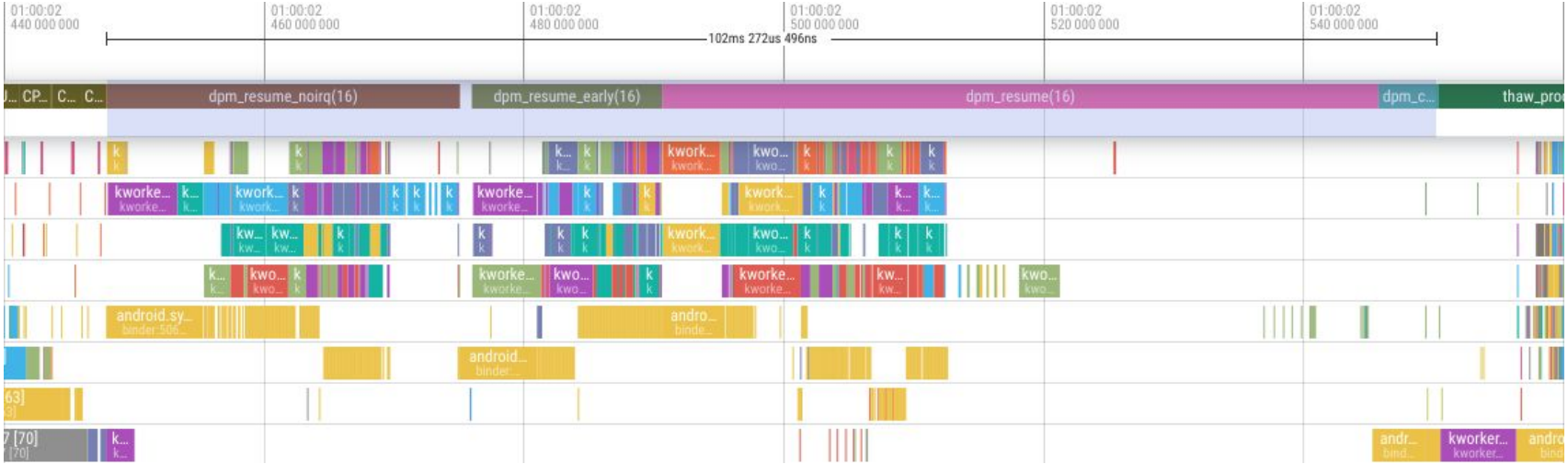
Optimizing dpm_resume*() stages

What does resume look like without any optimization?

It takes 82ms but the CPUs are mostly idle.

Optimizing dpm_resume*() stages

What does resume look like if we enable async suspend/resume for all devices?



It actually works and is stable. Thanks to fw_devlink tracking all dependencies.

The CPUs are very busy, but it takes 102ms. That's 20ms worse!

Optimizing dpm_suspend*() stages

What does suspend look like without any optimization?

It takes 126ms but the CPUs are mostly idle.

Optimizing dpm_suspend*() stages

What does suspend look like if we enable async suspend/resume for all devices?

It takes 94ms. That's 32ms better, but there might be room to improve.

Ironically, async suspends are triggered in a less async manner than async resumes.

Why is async hurting/not helping much?

Async suspend/resume has additional overhead:

- Work queuing, kworker wakeups and context switches.
- Lots of wait_for_completion on consumers, children, suppliers, and parent to finish.

And lots of devices have no ops or quick (microseconds) ops.

Async ends up being more expensive than sync for these devices.

Proposal: Async only slow devices

1. Have user define what's a sync threshold.
2. Kernel tracks worst case time to suspend/resume each device.
3. Any time a device's worst case time exceeds sync threshold:
    a. Set the async flag for it.
    b. Set the async flag for all it's consumers, consumers of consumers, etc.
    c. Set the async flag for all it's suppliers, suppliers of suppliers, etc.

3a & 3b are needed to avoid an async device waiting on a sync consumer/supplier that is deep in the sync devices list.

Concerns:

- Doing all this ends up with many "quick" devices using async and ends up adding too much overhead.
- Still doesn't parallelize as much as possible.

Proposal: Bottom-up breadth-first

1. Suspend leaf node devices in parallel (1 thread per-CPU).
2. This creates more leaf nodes in the graph.
3. Goto 1 until all nodes are suspended.

Avoids the overhead:

- No work queuing, kworker wakeups or context switch overhead.
- Removes wait_for_completion() by picking up only ready to go devices.

Am I missing something?

Any known issues which approach?

Lazy resume and early suspend w/ runtime PM

Why resume the display if the screen isn't going to get turned on?
Why serialize resuming storage and rendering a frame on screen?
Why resume a device if it's not needed right now?

fw_devlink=rpm is default right now.
Enforces runtime PM dependency too.
So, runtime PM is more likely to be stable.

Questions:
AFAIK, lazy resume with runtime PM is already supported
Why is it not more prevalent?
How can we make it easier for driver devs?

S2Idle woes

S2Idle works and is so much faster than S2RAM.

_IF_ the firmware supports it and isn't buggy.

S2Idle vs S2RAM gap increases with ever increasing CPU counts.

Getting firmware updates for existing devices is next to impossible.

Can we get S2idle like behavior by using S2RAM firmware calls?

- Add a fake C-state that just calls S2RAM hotplug API
- Power up CPU using S2RAM firmware calls before sending IPI to wake up the CPUs from the fake C-state

Can we get something like this working?

Linaro Connect
MADRID 2024 | MAY 12-17 2024

Thank you