Atomic Modesetting for Drivers

Daniel Vetter, Intel OTC

Anatomy of an Atomic Modeset

1. Build up new state
2. Compute derived state and check the update
3. Commit the new state to the hardware, possibly asynchronously

State Building

- per-object states structures tracked in `struct drm_atomic_state`
- `->atomic_duplicate/destroy_*` per-object
- `->atomic_set/get_property` only for private properties
- start with pure helpers, subclass as needed

State Checking

- global `->atomic_check` entry point
- plus big modular helper library
- helper supports legacy `->mode_fixup` and new `->atomic_check` hooks
- read the kerneldoc!

State Precomputing&Checking

- often check and commit need to compute the same values, e.g. DP link settings
- ➔ subclass state structures and store derived state for reuse in the commit phase
- almosts everything ends up being subclassed, tons of examples

Cross-State Structures Checking

- `->atomic_check` hooks can look at any other state
- always use provided functions and check errors to avoid wait/wound mutex headaches and unecessary serialization
- `CONFIG_DEBUG_WW_MUTEX_SLOWPATH`
- overwrite global `->atomic_check` if needed
- tons of examples already

Handling Global State

- for shared resources across CRTCs
- use driver-private w/w mutex or `dev->mode_config->connection_mutex`
- `->atomic_state_alloc/clear/free` to subclass global `struct drm_atomic_state`
- currently only i915: display core clock, shared PLLs, ...

State Committing

- global `->atomic_commit` entry point
- commit not allowed to fail due to invalid state
- core guarantees to call `->atomic_check` first
- helpers by default optimized for backwards compat
- modular helpers to accomodate more drivers, read docs!

Helper Design

- plane updates orthogonal to modeset changes
- no parital enables/disable, reducing complexity
- DPMS implemented entirely in helpers
- lots of old hooks depracated, most others optional
- legacy state updated by default, but can be ignored

➔ much fewer boilerplate required

Atomic Commit Flow

- `->prepare_fb` for memory alloc, pinning
- swap new state into objects (must be done synchronously)
- wait for fences and buffers
- actual hardware commit, built from helpers and driver code
- wait for vblank
- `->cleanup_fb` to for memory release, unpin

Hardware Commit Helpers

- CRTC, encoders and bridges for modesets with just enable/disable hooks
- 3-phase plane updates:

  1. CRTC `->atomic_begin` for vblank evade, blocking updates
  2. per-plane `->atomic_update/disable`
  3. CRTC `->atomic_flush` to set GO bit, unblock updates

Bootstrapping Atomic State

- atomic updates always incremental
- assume that software state perfectly matches hardware
- driver load and resume need to ensure matching state, use `->reset` hooks
- need not actually reset, hardware state readout for fastboot also possible

Legacy Entry Points

- helpers to implement them with atomic for all of them
- allows drivers to keep old features that don't make sense to port to atomic around

Ongoing for 4.4

- suspend/resume helpers
- atomic fbdev
- `active_only` plane update helpers
- better support for runtime PM in general

Future Work

- generic async commit
- state readout for fastboot à la i915
- more helpers as use-cases crop up …
- generic validation tests in i-g-t perhaps

KMS Extensions

- easy to do with properties
- color manager, plane blending, ...
- should put them into core drm state structures to avoid property proliferation
- same rules as any other kernel ABI

Android Support?

- just fences missing, but:
- hardware composer wants per-buffer relase fence, even before the next flip is scheduled

➔ trivial fencing deadlock

- … and no one has an open-source atomic hwc

Documentation

- conversion HOWTO for legacy drivers:
  http://blog.ffwll.ch/2014/11/atomic-modeset-support-for-kms-drivers.html
- LWN design overview: https://lwn.net/Articles/653071/
  https://lwn.net/Articles/653466/
- DRM DocBook: https://01.org/linuxgraphics/gfx-docs/drm/

Q & A