TODO list

This section contains a list of smaller janitorial tasks in the kernel DRM graphics subsystem useful as newbie projects. Or for slow rainy days.

Difficulty

To make it easier task are categorized into different levels:

Starter: Good tasks to get started with the DRM subsystem.

Intermediate: Tasks which need some experience with working in the DRM subsystem, or some specific GPU/display graphics knowledge. For debugging issue it's good to have the relevant hardware (or a virtual driver set up) available for testing.

Advanced: Tricky tasks that need fairly good understanding of the DRM subsystem and graphics topics. Generally need the relevant hardware for development and testing.

Expert: Only attempt these if you've successfully completed some tricky refactorings already and are an expert in the specific area

Subsystem-wide refactorings

Remove custom dumb_map_offset implementations

All GEM based drivers should be using drm_gem_create_mmap_offset() instead. Audit each individual driver, make sure it'll work with the generic implementation (there's lots of outdated locking leftovers in various implementations), and then remove it.

Contact: Daniel Vetter, respective driver maintainers

Level: Intermediate

Convert existing KMS drivers to atomic modesetting

3.19 has the atomic modeset interfaces and helpers, so drivers can now be converted over. Modern compositors like Wayland or Surfaceflinger on Android really want an atomic modeset interface, so this is all about the bright future.

There is a conversion guide for atomic and all you need is a GPU for a non-converted driver (again virtual HW drivers for KVM are still all suitable).

As part of this drivers also need to convert to universal plane (which means exposing primary & cursor as proper plane objects). But that's much easier to do by directly using the new atomic helper driver callbacks.

Contact: Daniel Vetter, respective driver maintainers

Level: Advanced

Clean up the clipped coordination confusion around planes

We have a helper to get this right with drm_plane_helper_check_update(), but it's not consistently used. This should be fixed, preferrably in the atomic helpers (and drivers then moved over to clipped coordinates). Probably the helper should also be moved from drm_plane_helper.c to the atomic helpers, to avoid confusion - the other helpers in that file are all deprecated legacy helpers.

Contact: Ville SyrjÃ¤lÃ¤, Daniel Vetter, driver maintainers

Level: Advanced

Improve plane atomic_check helpers

Aside from the clipped coordinates right above there's a few suboptimal things with the current helpers:

- drm_plane_helper_funcs->atomic_check gets called for enabled or disabled planes. At best this seems to confuse drivers, worst it means they blow up when the plane is disabled without the CRTC. The only special handling is resetting values in the plane state structures, which instead should be moved into the drm_plane_funcs->atomic_duplicate_state functions.
- Once that's done, helpers could stop calling ->atomic_check for disabled planes.
- Then we could go through all the drivers and remove the more-or-less confused checks for plane_state->fb and plane_state->crtc.

Contact: Daniel Vetter

Level: Advanced

Convert early atomic drivers to async commit helpers

For the first year the atomic modeset helpers didn't support asynchronous / nonblocking commits, and every driver had to hand-roll them. This is fixed now, but there's still a pile of existing drivers that easily could be converted over to the new infrastructure.

One issue with the helpers is that they require that drivers handle completion events for atomic commits correctly. But fixing these bugs is good anyway.

Somewhat related is the legacy_cursor_update hack, which should be replaced with the new atomic_async_check/commit functionality in the helpers in drivers that still look at that flag.

Contact: Daniel Vetter, respective driver maintainers

Level: Advanced

Fallout from atomic KMS

`drm_atomic_helper.c` provides a batch of functions which implement legacy IOCTLs on top of the new atomic driver interface. Which is really nice for gradual conversion of drivers, but unfortunately the semantic mismatches are a bit too severe. So there's some follow-up work to adjust the function interfaces to fix these issues:

- atomic needs the lock acquire context. At the moment that's passed around implicitly with some horrible hacks, and it's also allocate with `GFP_NOFAIL` behind the scenes. All legacy paths need to start allocating the acquire context explicitly on stack and then also pass it down into drivers explicitly so that the legacy-on-atomic functions can use them.

  Except for some driver code this is done. This task should be finished by adding WARN_ON(!drm_drv_uses_atomic_modeset) in drm_modeset_lock_all().
- A bunch of the vtable hooks are now in the wrong place: DRM has a split between core vfunc tables (named `drm_foo_funcs`), which are used to implement the userspace ABI. And then there's the optional hooks for the helper libraries (name `drm_foo_helper_funcs`), which are purely for internal use. Some of these hooks should be move from `_funcs` to `_helper_funcs` since they are not part of the core ABI. There's a `FIXME` comment in the kerneldoc for each such case in `drm_crtc.h`.

Contact: Daniel Vetter

Level: Intermediate

Get rid of dev->struct_mutex from GEM drivers

`dev->struct_mutex` is the Big DRM Lock from legacy days and infested everything. Nowadays in modern drivers the only bit where it's mandatory is serializing GEM buffer object destruction. Which unfortunately means drivers have to keep track of that lock and either call `unreference` or `unreference_locked` depending upon context.

Core GEM doesn't have a need for `struct_mutex` any more since kernel 4.8, and there's a GEM object `free` callback for any drivers which are entirely `struct_mutex` free.

For drivers that need `struct_mutex` it should be replaced with a driver- private lock. The tricky part is the BO free functions, since those can't reliably take that lock any more. Instead state needs to be protected with suitable subordinate locks or some cleanup work pushed to a worker thread. For performance-critical drivers it might also be better to go with a more fine-grained per-buffer object and per-context lockings scheme. Currently only the `msm` and *i915* drivers use `struct_mutex`.

Contact: Daniel Vetter, respective driver maintainers

Level: Advanced

Move Buffer Object Locking to dma_resv_lock()

Many drivers have their own per-object locking scheme, usually using mutex_lock(). This causes all kinds of trouble for buffer sharing, since depending which driver is the exporter and importer, the locking hierarchy is reversed.

To solve this we need one standard per-object locking mechanism, which is dma_resv_lock(). This lock needs to be called as the outermost lock, with all other driver specific per-object locks removed. The problem is tha rolling out the actual change to the locking contract is a flag day, due to struct dma_buf buffer sharing.

Level: Expert

Convert logging to drm_* functions with drm_device paramater

For drivers which could have multiple instances, it is necessary to differentiate between which is which in the logs. Since DRM_INFO/WARN/ERROR don't do this, drivers used dev_info/warn/err to make this differentiation. We now have drm_* variants of the drm print functions, so we can start to convert those drivers back to using drm-formatted specific log messages.

Before you start this conversion please contact the relevant maintainers to make sure your work will be merged - not everyone agrees that the DRM dmesg macros are better.

Contact: Sean Paul, Maintainer of the driver you plan to convert

Level: Starter

Convert drivers to use simple modeset suspend/resume

Most drivers (except i915 and nouveau) that use drm_atomic_helper_suspend/resume() can probably be converted to use drm_mode_config_helper_suspend/resume(). Also there's still open-coded version of the atomic suspend/resume code in older atomic modeset drivers.

Contact: Maintainer of the driver you plan to convert

Level: Intermediate

Convert drivers to use drm_fbdev_generic_setup()

Most drivers can use drm_fbdev_generic_setup(). Driver have to implement atomic modesetting and GEM vmap support. Historically, generic fbdev emulation expected the framebuffer in system memory or system-like memory. By employing struct iosys_map, drivers with frambuffers in I/O memory can be supported as well.

Contact: Maintainer of the driver you plan to convert

Level: Intermediate

Reimplement functions in drm_fbdev_fb_ops without fbdev

A number of callback functions in drm_fbdev_fb_ops could benefit from being rewritten without dependencies on the fbdev module. Some of the helpers could further benefit from using struct iosys_map instead of raw pointers.

Contact: Thomas Zimmermann <tzimmermann@suse.de>, Daniel Vetter

Level: Advanced

Benchmark and optimize blitting and format-conversion function

Drawing to dispay memory quickly is crucial for many applications' performance.

On at least x86-64, sys_imageblit() is significantly slower than cfb_imageblit(), even though both use the same blitting algorithm and the latter is written for I/O memory. It turns out that cfb_imageblit() uses movl instructions, while sys_imageblit apparently does not. This seems to be a problem with gcc's optimizer. DRM's format-conversion helpers might be subject to similar issues.

Benchmark and optimize fbdev's sys_() helpers and DRM's format-conversion helpers. In cases that can be further optimized, maybe implement a different algorithm. For micro-optimizations, use movl/movq instructions explicitly. That might possibly require architecture-specific helpers (e.g., storel() storeq()).

Contact: Thomas Zimmermann <tzimmermann@suse.de>

Level: Intermediate

drm_framebuffer_funcs and drm_mode_config_funcs.fb_create cleanup

A lot more drivers could be switched over to the drm_gem_framebuffer helpers. Various hold-ups:

- Need to switch over to the generic dirty tracking code using drm_atomic_helper_dirtyfb first (e.g. qxl).
- Need to switch to drm_fbdev_generic_setup(), otherwise a lot of the custom fb setup code can't be deleted.
- Many drivers wrap drm_gem_fb_create() only to check for valid formats. For atomic drivers we could check for valid formats by calling drm_plane_check_pixel_format() against all planes, and pass if any plane supports the format. For non-atomic that's not possible since like the format list for the primary plane is fake and we'd therefor reject valid formats.
- Many drivers subclass drm_framebuffer, we'd need a embedding compatible version of the varios drm_gem_fb_create functions. Maybe called drm_gem_fb_create/_with_dirty/_with_funcs as needed.

Contact: Daniel Vetter

Level: Intermediate

Generic fbdev defio support

The defio support code in the fbdev core has some very specific requirements, which means drivers need to have a special framebuffer for fbdev. The main issue is that it uses some fields in struct page itself, which breaks shmem gem objects (and other things). To support defio, affected drivers require the use of a shadow buffer, which may add CPU and memory overhead.

Possible solution would be to write our own defio mmap code in the drm fbdev emulation. It would need to fully wrap the existing mmap ops, forwarding everything after it has done the write-protect/mkwrite trickery:

- In the drm_fbdev_fb_mmap helper, if we need defio, change the default page prots to write-protected with something like this:

```
        vma->vm_page_prot = pgprot_wrprotect(vma->vm_page_prot);
```

- Set the mkwrite and fsync callbacks with similar implementions to the core fbdev defio stuff. These should all work on plain ptes, they don't actually require a struct page. uff. These should all work on plain ptes, they don't actually require a struct page.
- Track the dirty pages in a separate structure (bitfield with one bit per page should work) to avoid clobbering struct page.

Might be good to also have some igt testcases for this.

Contact: Daniel Vetter, Noralf Tronnes

Level: Advanced

idr_init_base()

DRM core&drivers uses a lot of idr (integer lookup directories) for mapping userspace IDs to internal objects, and in most places ID=0 means NULL and hence is never used. Switching to idr_init_base() for these would make the idr more efficient.

Contact: Daniel Vetter

Level: Starter

struct drm_gem_object_funcs

GEM objects can now have a function table instead of having the callbacks on the DRM driver struct. This is now the preferred way. Callbacks in drivers have been converted, except for struct drm_driver.gem_prime_mmap.

Level: Intermediate

Rename CMA helpers to DMA helpers

CMA (standing for contiguous memory allocator) is really a bit an accident of what these were used for first, a much better name would be DMA helpers. In the text these should even be called coherent DMA memory helpers (so maybe CDM, but no one knows what that means) since underneath they just use dma_alloc_coherent.

Contact: Laurent Pinchart, Daniel Vetter

Level: Intermediate (mostly because it is a huge tasks without good partial milestones, not technically itself that challenging)

connector register/unregister fixes

- For most connectors it's a no-op to call drm_connector_register/unregister directly from driver code, drm_dev_register/unregister take care of this already. We can remove all of them.
- For dp drivers it's a bit more a mess, since we need the connector to be registered when calling drm_dp_aux_register. Fix this by instead calling drm_dp_aux_init, and moving the actual registering into a late_register callback as recommended in the kerneldoc.

Level: Intermediate

Remove load/unload callbacks from all non-DRIVER_LEGACY drivers

The load/unload callbacks in struct &drm_driver are very much midlayers, plus for historical reasons they get the ordering wrong (and we can't fix that) between setting up the &drm_driver structure and calling drm_dev_register().

- Rework drivers to no longer use the load/unload callbacks, directly coding the load/unload sequence into the driver's probe function.
- Once all non-DRIVER_LEGACY drivers are converted, disallow the load/unload callbacks for all modern drivers.

Contact: Daniel Vetter

Level: Intermediate

Replace drm_detect_hdmi_monitor() with drm_display_info.is_hdmi

Once EDID is parsed, the monitor HDMI support information is available through drm_display_info.is_hdmi. Many drivers still call drm_detect_hdmi_monitor() to retrieve the same information, which is less efficient.

Audit each individual driver calling drm_detect_hdmi_monitor() and switch to drm_display_info.is_hdmi if applicable.

Contact: Laurent Pinchart, respective driver maintainers

Level: Intermediate

Consolidate custom driver modeset properties

Before atomic modeset took place, many drivers where creating their own properties. Among other things, atomic brought the requirement that custom, driver specific properties should not be used.

For this task, we aim to introduce core helpers or reuse the existing ones if available:

A quick, unconfirmed, examples list.

Introduce core helpers: - audio (amdgpu, intel, gma500, radeon) - brightness, contrast, etc (armada, nouveau) - overlay only (?) - broadcast rgb (gma500, intel) - colorkey (armada, nouveau, rcar) - overlay only (?) - dither (amdgpu, nouveau, radeon) - varies across drivers - underscan family (amdgpu, radeon, nouveau)

Already in core: - colorspace (sti) - tv format names, enhancements (gma500, intel) - tv overscan, margins, etc. (gma500, intel) - zorder (omapdrm) - same as zpos (?)

Contact: Emil Velikov, respective driver maintainers

Level: Intermediate

Use struct iosys_map throughout codebase

Pointers to shared device memory are stored in struct iosys_map. Each instance knows whether it refers to system or I/O memory. Most of the DRM-wide interface have been converted to use struct iosys_map, but implementations often still use raw pointers.

The task is to use struct iosys_map where it makes sense.

- Memory managers should use struct iosys_map for dma-buf-imported buffers.
- TTM might benefit from using struct iosys_map internally.
- Framebuffer copying and blitting helpers should operate on struct iosys_map.

Contact: Thomas Zimmermann <tzimmermann@suse.de>, Christian KÃ¶nig, Daniel Vetter

Level: Intermediate

Review all drivers for setting struct drm_mode_config.{max_width,max_height} correctly

The values in struct drm_mode_config.{max_width,max_height} describe the maximum supported framebuffer size. It's the virtual screen size, but many drivers treat it like limitations of the physical resolution.

The maximum width depends on the hardware's maximum scanline pitch. The maximum height depends on the amount of addressable video memory. Review all drivers to initialize the fields to the correct values.

Contact: Thomas Zimmermann <tzimmermann@suse.de>

Level: Intermediate

Request memory regions in all drivers

Go through all drivers and add code to request the memory regions that the driver uses. This requires adding calls to request_mem_region(), pci_request_region() or similar functions. Use helpers for managed cleanup where possible.

Drivers are pretty bad at doing this and there used to be conflicts among DRM and fbdev drivers. Still, it's the correct thing to do.

Contact: Thomas Zimmermann <tzimmermann@suse.de>

Level: Starter

Core refactorings

Make panic handling work

This is a really varied tasks with lots of little bits and pieces:

- The panic path can't be tested currently, leading to constant breaking. The main issue here is that panics can be triggered from hardirq contexts and hence all panic related callback can run in hardirq context. It would be awesome if we could test at least the fbdev helper code and driver code by e.g. trigger calls through drm debugfs files. hardirq context could be achieved by using an IPI to the local processor.
- There's a massive confusion of different panic handlers. DRM fbdev emulation helpers had their own (long removed), but on top of that the fbcon code itself also has one. We need to make sure that they stop fighting over each other. This is worked around by checking `oops_in_progress` at various entry points into the DRM fbdev emulation helpers. A much cleaner approach here would be to switch fbcon to the threaded printk support.
- `drm_can_sleep()` is a mess. It hides real bugs in normal operations and isn't a full solution for panic paths. We need to make sure that it only returns true if there's a panic going on for real, and fix up all the fallout.
- The panic handler must never sleep, which also means it can't ever `mutex_lock()`. Also it can't grab any other lock

unconditionally, not even spinlocks (because NMI and hardirq can panic too). We need to either make sure to not call such paths, or trylock everything. Really tricky.

- A clean solution would be an entirely separate panic output support in KMS, bypassing the current fbcon support. See [PATCH v2 0/3] drm: Add panic handling.
- Encoding the actual oops and preceding dmesg in a QR might help with the dread "important stuff scrolled away" problem. See [RFC][PATCH] Oops messages transfer using QR codes for some example code that could be reused.

Contact: Daniel Vetter

Level: Advanced

Clean up the debugfs support

There's a bunch of issues with it:

- The drm_info_list ->show() function doesn't even bother to cast to the drm structure for you. This is lazy.
- We probably want to have some support for debugfs files on crtc/connectors and maybe other kms objects directly in core. There's even drm_print support in the funcs for these objects to dump kms state, so it's all there. And then the ->show() functions should obviously give you a pointer to the right object.
- The drm_info_list stuff is centered on drm_minor instead of drm_device. For anything we want to print drm_device (or maybe drm_file) is the right thing.
- The drm_driver->debugfs_init hooks we have is just an artifact of the old midlayered load sequence. DRM debugfs should work more like sysfs, where you can create properties/files for an object anytime you want, and the core takes care of publishing/unpuplishing all the files at register/unregister time. Drivers shouldn't need to worry about these technicalities, and fixing this (together with the drm_minor->drm_device move) would allow us to remove debugfs_init.

Previous RFC that hasn't landed yet: https://lore.kernel.org/dri-devel/20200513114130.28641-2-wambui.karugax@gmail.com/

Contact: Daniel Vetter

Level: Intermediate

Object lifetime fixes

There's two related issues here

- Cleanup up the various ->destroy callbacks, which often are all the same simple code.
- Lots of drivers erroneously allocate DRM modeset objects using devm_kzalloc, which results in use-after free issues on driver unload. This can be serious trouble even for drivers for hardware integrated on the SoC due to EPROBE_DEFERRED backoff.

Both these problems can be solved by switching over to drmm_kzalloc(), and the various convenience wrappers provided, e.g. drmm_crtc_alloc_with_planes(), drmm_universal_plane_alloc(), ... and so on.

Contact: Daniel Vetter

Level: Intermediate

Remove automatic page mapping from dma-buf importing

When importing dma-bufs, the dma-buf and PRIME frameworks automatically map imported pages into the importer's DMA area. drm_gem_prime_fd_to_handle() and drm_gem_prime_handle_to_fd() require that importers call dma_buf_attach() even if they never do actual device DMA, but only CPU access through dma_buf_vmap(). This is a problem for USB devices, which do not support DMA operations.

To fix the issue, automatic page mappings should be removed from the buffer-sharing code. Fixing this is a bit more involved, since the import/export cache is also tied to &drm_gem_object.import_attach. Meanwhile we paper over this problem for USB devices by fishing out the USB host controller device, as long as that supports DMA. Otherwise importing can still needlessly fail.

Contact: Thomas Zimmermann <tzimmermann@suse.de>, Daniel Vetter

Level: Advanced

Better Testing

Enable trinity for DRM

And fix up the fallout. Should be really interesting ...

Level: Advanced

Make KMS tests in i-g-t generic

The i915 driver team maintains an extensive testsuite for the i915 DRM driver, including tons of testcases for corner-cases in the modesetting API. It would be awesome if those tests (at least the ones not relying on Intel-specific GEM features) could be made to run on any KMS driver.

Basic work to run i-g-t tests on non-i915 is done, what's now missing is mass- converting things over. For modeset tests we also first need a bit of infrastructure to use dumb buffers for untiled buffers, to be able to run all the non-i915 specific modeset tests.

Level: Advanced

Extend virtual test driver (VKMS)

See the documentation of :ref:`VKMS <vkms>` for more details. This is an ideal internship task, since it only requires a virtual machine and can be sized to fit the available time.

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\[linux-master][Documentation][gpu]todo.rst`, **line 631**); ***backlink***
>
> Unknown interpreted text role "ref".

Level: See details

Backlight Refactoring

Backlight drivers have a triple enable/disable state, which is a bit overkill. Plan to fix this:

1. Roll out backlight_enable() and backlight_disable() helpers everywhere. This has started already.
2. In all, only look at one of the three status bits set by the above helpers.
3. Remove the other two status bits.

Contact: Daniel Vetter

Level: Intermediate

Driver Specific

AMD DC Display Driver

AMD DC is the display driver for AMD devices starting with Vega. There has been a bunch of progress cleaning it up but there's still plenty of work to be done.

See drivers/gpu/drm/amd/display/TODO for tasks.

Contact: Harry Wentland, Alex Deucher

vmwgfx: Replace hashtable with Linux' implementation

The vmwgfx driver uses its own hashtable implementation. Replace the code with Linux' implementation and update the callers. It's mostly a refactoring task, but the interfaces are different.

Contact: Zack Rusin, Thomas Zimmermann <tzimmermann@suse.de>

Level: Intermediate

Bootsplash

There is support in place now for writing internal DRM clients making it possible to pick up the bootsplash work that was rejected because it was written for fbdev.

- [v6,8/8] drm/client: Hack: Add bootsplash example https://patchwork.freedesktop.org/patch/306579/
- [RFC PATCH v2 00/13] Kernel based bootsplash https://lore.kernel.org/r/20171213194755.3409-1-mstaudt@suse.de

Contact: Sam Ravnborg

Level: Advanced

Outside DRM

Convert fbdev drivers to DRM

There are plenty of fbdev drivers for older hardware. Some hardware has become obsolete, but some still provides good(-enough) framebuffers. The drivers that are still useful should be converted to DRM and afterwards removed from fbdev.

Very simple fbdev drivers can best be converted by starting with a new DRM driver. Simple KMS helpers and SHMEM should be able to handle any existing hardware. The new driver's call-back functions are filled from existing fbdev code.

More complex fbdev drivers can be refactored step-by-step into a DRM driver with the help of the DRM fbconv helpers. [1] These helpers provide the transition layer between the DRM core infrastructure and the fbdev driver interface. Create a new DRM driver on top of the fbconv helpers, copy over the fbdev driver, and hook it up to the DRM code. Examples for several fbdev drivers are available at [1] and a tutorial of this process available at [2]. The result is a primitive DRM driver that can run X11 and Weston.

- [1] https://gitlab.freedesktop.org/tzimmermann/linux/tree/fbconv
- [2] https://gitlab.freedesktop.org/tzimmermann/linux/blob/fbconv/drivers/gpu/drm/drm_fbconv_helper.c

Contact: Thomas Zimmermann <tzimmermann@suse.de>

Level: Advanced