DRM Memory Management

Modern Linux systems require large amount of graphics memory to store frame buffers, textures, vertices and other graphics-related data. Given the very dynamic nature of many of that data, managing graphics memory efficiently is thus crucial for the graphics stack and plays a central role in the DRM infrastructure.

The DRM core includes two memory managers, namely Translation Table Manager (TTM) and Graphics Execution Manager (GEM). TTM was the first DRM memory manager to be developed and tried to be a one-size-fits-them all solution. It provides a single userspace API to accommodate the need of all hardware, supporting both Unified Memory Architecture (UMA) devices and devices with dedicated video RAM (i.e. most discrete video cards). This resulted in a large, complex piece of code that turned out to be hard to use for driver development.

GEM started as an Intel-sponsored project in reaction to TTM's complexity. Its design philosophy is completely different: instead of providing a solution to every graphics memory-related problems, GEM identified common code between drivers and created a support library to share it. GEM has simpler initialization and execution requirements than TTM, but has no video RAM management capabilities and is thus limited to UMA devices.

The Translation Table Manager (TTM)

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\(linux-master) (Documentation) (gpu)drm-mm.rst`, line 31)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/gpu/drm/ttm/ttm_module.c
   :doc: TTM
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\(linux-master) (Documentation) (gpu)drm-mm.rst`, line 34)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/drm/ttm/ttm_caching.h
   :internal:
```

TTM device object reference

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\(linux-master) (Documentation) (gpu)drm-mm.rst`, line 40)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/drm/ttm/ttm_device.h
   :internal:
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\(linux-master) (Documentation) (gpu)drm-mm.rst`, line 43)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/gpu/drm/ttm/ttm_device.c
   :export:
```

TTM resource placement reference

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\(linux-master) (Documentation) (gpu)drm-mm.rst`, line 49)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/drm/ttm/ttm_placement.h
   :internal:
```

TTM resource object reference

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\(linux-master) (Documentation) (gpu)drm-mm.rst`, line 55)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/drm/ttm/ttm_resource.h
   :internal:
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\(linux-master) (Documentation) (gpu)drm-mm.rst`, line 58)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/gpu/drm/ttm/ttm_resource.c
   :export:
```

TTM TT object reference

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\(linux-master) (Documentation) (gpu)drm-mm.rst`, line 64)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/drm/ttm/ttm_tt.h
   :internal:
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\(linux-master) (Documentation) (gpu)drm-mm.rst`, line 67)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/gpu/drm/ttm/ttm_tt.c
   :export:
```

TTM page pool reference

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\(linux-master) (Documentation) (gpu)drm-mm.rst`, line 73)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/drm/ttm/ttm_pool.h
   :internal:
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\(linux-master) (Documentation) (gpu)drm-mm.rst`, line 76)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/gpu/drm/ttm/ttm_pool.c
   :export:
```

The Graphics Execution Manager (GEM)

The GEM design approach has resulted in a memory manager that doesn't provide full coverage of all (or even all common) use cases in its userspace or kernel API. GEM exposes a set of standard memory-related operations to userspace and a set of helper functions to drivers, and let drivers implement hardware-specific operations with their own private API.

The GEM userspace API is described in the GEM - the Graphics Execution Manager article on LWN. While slightly outdated, the document provides a good overview of the GEM API principles. Buffer allocation and read and write operations, described as part of the common GEM API, are currently implemented using driver-specific ioctls.

GEM is data-agnostic. It manages abstract buffer objects without knowing what individual buffers contain. APIs that require knowledge of buffer contents or purpose, such as buffer allocation or synchronization primitives, are thus outside of the scope of

GEM and must be implemented using driver-specific ioctls.

On a fundamental level, GEM involves several operations:

- Memory allocation and freeing
- Command execution
- Aperture management at command execution time

Buffer object allocation is relatively straightforward and largely provided by Linux's shmem layer, which provides memory to back each object.

Device-specific operations, such as command execution, pinning, buffer read & write, mapping, and domain ownership transfers are left to driver-specific ioctls.

GEM Initialization

Drivers that use GEM must set the DRIVER_GEM bit in the struct :c:type:`struct drm_driver <drm_driver>` driver_features field. The DRM core will then automatically initialize the GEM core before calling the load operation. Behind the scene, this will create a DRM Memory Manager object which provides an address space pool for object allocation.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\(linux-master) (Documentation) (gpu)drm-mm.rst`, line 119); *backlink*

Unknown interpreted text role "c:type".

In a KMS configuration, drivers need to allocate and initialize a command ring buffer following core GEM initialization if required by the hardware. UMA devices usually have what is called a "stolen" memory region, which provides space for the initial framebuffer and large, contiguous memory regions required by the device. This space is typically not managed by GEM, and must be initialized separately into its own DRM MM object.

GEM Objects Creation

GEM splits creation of GEM objects and allocation of the memory that backs them in two distinct operations.

GEM objects are represented by an instance of struct :c:type:`struct drm_gem_object <drm_gem_object>`. Drivers usually need to extend GEM objects with private information and thus create a driver-specific GEM object structure type that embeds an instance of struct :c:type:`struct drm_gem_object <drm_gem_object>`.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\(linux-master) (Documentation) (gpu)drm-mm.rst`, line 140); *backlink*

Unknown interpreted text role "c:type".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\(linux-master) (Documentation) (gpu)drm-mm.rst`, line 140); *backlink*

Unknown interpreted text role "c:type".

To create a GEM object, a driver allocates memory for an instance of its specific GEM object type and initializes the embedded struct :c:type:`struct drm_gem_object <drm_gem_object>` with a call to drm_gem_object_init(). The function takes a pointer to the DRM device, a pointer to the GEM object and the buffer object size in bytes.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\(linux-master) (Documentation) (gpu)drm-mm.rst`, line 146); *backlink*

Unknown interpreted text role "c:type".

GEM uses shmem to allocate anonymous pageable memory. drm_gem_object_init() will create an shmfs file of the requested size and store it into the struct :c:type:`struct drm_gem_object <drm_gem_object>` filp field. The memory is used as either main storage for the object when the graphics hardware uses system memory directly or as a backing store otherwise.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\(linux-master) (Documentation) (gpu)drm-mm.rst`, line 153); *backlink*

Unknown interpreted text role "c:type".

Drivers are responsible for the actual physical pages allocation by calling shmem_read_mapping_page_gfp() for each page. Note that they can decide to allocate pages when initializing the GEM object, or to delay allocation until the memory is needed (for instance when a page fault occurs as a result of a userspace memory access or when the driver needs to start a DMA transfer involving the memory).

Anonymous pageable memory allocation is not always desired, for instance when the hardware requires physically contiguous system memory as is often the case in embedded devices. Drivers can create GEM objects with no shmfs backing (called private GEM objects) by initializing them with a call to drm_gem_private_object_init() instead of drm_gem_object_init(). Storage for private GEM objects must be managed by drivers.

GEM Objects Lifetime

All GEM objects are reference-counted by the GEM core. References can be acquired and release by calling drm_gem_object_get() and drm_gem_object_put() respectively.

When the last reference to a GEM object is released the GEM core calls the :c:type:`struct drm_gem_object_funcs <gem_object_funcs>` free operation. That operation is mandatory for GEM-enabled drivers and must free the GEM object and all associated resources.

> **System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\(linux-master) (Documentation) (gpu)drm-mm.rst`, line 181); *backlink***
>
> Unknown interpreted text role "c:type".

void (*free) (struct drm_gem_object *obj); Drivers are responsible for freeing all GEM object resources. This includes the resources created by the GEM core, which need to be released with drm_gem_object_release().

GEM Objects Naming

Communication between userspace and the kernel refers to GEM objects using local handles, global names or, more recently, file descriptors. All of those are 32-bit integer values; the usual Linux kernel limits apply to the file descriptors.

GEM handles are local to a DRM file. Applications get a handle to a GEM object through a driver-specific ioctl, and can use that handle to refer to the GEM object in other standard or driver-specific ioctls. Closing a DRM file handle frees all its GEM handles and dereferences the associated GEM objects.

To create a handle for a GEM object drivers call drm_gem_handle_create(). The function takes a pointer to the DRM file and the GEM object and returns a locally unique handle. When the handle is no longer needed drivers delete it with a call to drm_gem_handle_delete(). Finally the GEM object associated with a handle can be retrieved by a call to drm_gem_object_lookup().

Handles don't take ownership of GEM objects, they only take a reference to the object that will be dropped when the handle is destroyed. To avoid leaking GEM objects, drivers must make sure they drop the reference(s) they own (such as the initial reference taken at object creation time) as appropriate, without any special consideration for the handle. For example, in the particular case of combined GEM object and handle creation in the implementation of the dumb_create operation, drivers must drop the initial reference to the GEM object before returning the handle.

GEM names are similar in purpose to handles but are not local to DRM files. They can be passed between processes to reference a GEM object globally. Names can't be used directly to refer to objects in the DRM API, applications must convert handles to names and names to handles using the DRM_IOCTL_GEM_FLINK and DRM_IOCTL_GEM_OPEN ioctls respectively. The conversion is handled by the DRM core without any driver-specific support.

GEM also supports buffer sharing with dma-buf file descriptors through PRIME. GEM-based drivers must use the provided helpers functions to implement the exporting and importing correctly. See ?. Since sharing file descriptors is inherently more secure than the easily guessable and global GEM names it is the preferred buffer sharing mechanism. Sharing buffers through GEM names is only supported for legacy userspace. Furthermore PRIME also allows cross-device buffer sharing since it is based on dma-bufs.

GEM Objects Mapping

Because mapping operations are fairly heavyweight GEM favours read/write-like access to buffers, implemented through driver-specific ioctls, over mapping buffers to userspace. However, when random access to the buffer is needed (to perform software rendering for instance), direct access to the object can be more efficient.

The mmap system call can't be used directly to map GEM objects, as they don't have their own file handle. Two alternative methods currently co-exist to map GEM objects to userspace. The first method uses a driver-specific ioctl to perform the mapping operation, calling do_mmap() under the hood. This is often considered dubious, seems to be discouraged for new GEM-enabled drivers, and will thus not be described here.

The second method uses the mmap system call on the DRM file handle. void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset); DRM identifies the GEM object to be mapped by a fake offset passed through the mmap offset argument. Prior to being mapped, a GEM object must thus be associated with a fake offset. To do so, drivers must call drm_gem_create_mmap_offset() on the object.

Once allocated, the fake offset value must be passed to the application in a driver-specific way and can then be used as the mmap offset argument.

The GEM core provides a helper method drm_gem_mmap() to handle object mapping. The method can be set directly as the mmap file operation handler. It will look up the GEM object based on the offset value and set the VMA operations to the :c:type:`struct drm_driver <drm_driver>` gem_vm_ops field. Note that drm_gem_mmap() doesn't map memory to userspace, but relies on the driver-provided fault handler to map pages individually.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\(linux-master) (Documentation) (gpu)drm-mm.rst`, line 266); *backlink*

Unknown interpreted text role "c:type".

To use drm_gem_mmap(), drivers must fill the struct :c:type:`struct drm_driver <drm_driver>` gem_vm_ops field with a pointer to VM operations.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\(linux-master) (Documentation) (gpu)drm-mm.rst`, line 274); *backlink*

Unknown interpreted text role "c:type".

The VM operations is a :c:type:`struct vm_operations_struct <vm_operations_struct>` made up of several fields, the more interesting ones being:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\(linux-master) (Documentation) (gpu)drm-mm.rst`, line 277); *backlink*

Unknown interpreted text role "c:type".

```
struct vm_operations_struct {
        void (*open)(struct vm_area_struct * area);
        void (*close)(struct vm_area_struct * area);
        vm_fault_t (*fault)(struct vm_fault *vmf);
};
```

The open and close operations must update the GEM object reference count. Drivers can use the drm_gem_vm_open() and drm_gem_vm_close() helper functions directly as open and close handlers.

The fault operation handler is responsible for mapping individual pages to userspace when a page fault occurs. Depending on the memory allocation scheme, drivers can allocate pages at fault time, or can decide to allocate memory for the GEM object at the time the object is created.

Drivers that want to map the GEM object upfront instead of handling page faults can implement their own mmap file operation handler.

For platforms without MMU the GEM core provides a helper method drm_gem_cma_get_unmapped_area(). The mmap() routines will call this to get a proposed address for the mapping.

To use drm_gem_cma_get_unmapped_area(), drivers must fill the struct :c:type:`struct file_operations <file_operations>` get_unmapped_area field with a pointer on drm_gem_cma_get_unmapped_area().

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\(linux-master) (Documentation) (gpu)drm-mm.rst`, line 306); *backlink*

Unknown interpreted text role "c:type".

More detailed information about get_unmapped_area can be found in Documentation/admin-guide/mm/nommu-mmap.rst

Memory Coherency

When mapped to the device or used in a command buffer, backing pages for an object are flushed to memory and marked write combined so as to be coherent with the GPU. Likewise, if the CPU accesses an object after the GPU has finished rendering to the object, then the object must be made coherent with the CPU's view of memory, usually involving GPU cache flushing of various kinds. This core CPU<->GPU coherency management is provided by a device-specific ioctl, which evaluates an object's current domain and performs any necessary flushing or synchronization to put the object into the desired coherency domain (note that the object may be busy, i.e. an active render target; in that case, setting the domain blocks the client and waits for rendering to complete before performing any necessary flushing operations).

Command Execution

Perhaps the most important GEM function for GPU devices is providing a command execution interface to clients. Client programs construct command buffers containing references to previously allocated memory objects, and then submit them to GEM. At that point, GEM takes care to bind all the objects into the GTT, execute the buffer, and provide necessary synchronization between clients accessing the same buffers. This often involves evicting some objects from the GTT and re-binding others (a fairly expensive operation), and providing relocation support which hides fixed GTT offsets from clients. Clients must take care not to submit command buffers that reference more objects than can fit in the GTT; otherwise, GEM will reject them and no rendering will occur. Similarly, if several objects in the buffer require fence registers to be allocated for correct rendering (e.g. 2D blits on pre-965 chips), care must be taken not to require more fence registers than are available to the client. Such resource management should be abstracted from the client in libdrm.

GEM Function Reference

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\(linux-master) (Documentation) (gpu)drm-mm.rst`, line 352)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/drm/drm_gem.h
   :internal:
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\(linux-master) (Documentation) (gpu)drm-mm.rst`, line 355)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/gpu/drm/drm_gem.c
   :export:
```

GEM CMA Helper Functions Reference

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\(linux-master) (Documentation) (gpu)drm-mm.rst`, line 361)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/gpu/drm/drm_gem_cma_helper.c
   :doc: cma helpers
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\(linux-master) (Documentation) (gpu)drm-mm.rst`, line 364)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/drm/drm_gem_cma_helper.h
   :internal:
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\(linux-master) (Documentation) (gpu)drm-mm.rst`, line 367)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/gpu/drm/drm_gem_cma_helper.c
   :export:
```

GEM SHMEM Helper Function Reference

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\(linux-master) (Documentation) (gpu)drm-mm.rst`, line 373)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/gpu/drm/drm_gem_shmem_helper.c
   :doc: overview
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\(linux-master) (Documentation) (gpu)drm-mm.rst`, line 376)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/drm/drm_gem_shmem_helper.h
```

```
   :internal:
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\(linux-master) (Documentation) (gpu)drm-mm.rst`, line 379)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/gpu/drm/drm_gem_shmem_helper.c
   :export:
```

GEM VRAM Helper Functions Reference

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\(linux-master) (Documentation) (gpu)drm-mm.rst`, line 385)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/gpu/drm/drm_gem_vram_helper.c
   :doc: overview
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\(linux-master) (Documentation) (gpu)drm-mm.rst`, line 388)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/drm/drm_gem_vram_helper.h
   :internal:
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\(linux-master) (Documentation) (gpu)drm-mm.rst`, line 391)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/gpu/drm/drm_gem_vram_helper.c
   :export:
```

GEM TTM Helper Functions Reference

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\(linux-master) (Documentation) (gpu)drm-mm.rst`, line 397)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/gpu/drm/drm_gem_ttm_helper.c
   :doc: overview
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\(linux-master) (Documentation) (gpu)drm-mm.rst`, line 400)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/gpu/drm/drm_gem_ttm_helper.c
   :export:
```

VMA Offset Manager

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\(linux-master) (Documentation) (gpu)drm-mm.rst`, line 406)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/gpu/drm/drm_vma_manager.c
   :doc: vma offset manager
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\(linux-master) (Documentation) (gpu)drm-mm.rst`, line 409)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/drm/drm_vma_manager.h
   :internal:
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\(linux-master) (Documentation) (gpu)drm-mm.rst`, line 412)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/gpu/drm/drm_vma_manager.c
   :export:
```

PRIME Buffer Sharing

PRIME is the cross device buffer sharing framework in drm, originally created for the OPTIMUS range of multi-gpu platforms. To userspace PRIME buffers are dma-buf based file descriptors.

Overview and Lifetime Rules

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\(linux-master) (Documentation) (gpu)drm-mm.rst`, line 427)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/gpu/drm/drm_prime.c
   :doc: overview and lifetime rules
```

PRIME Helper Functions

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\(linux-master) (Documentation) (gpu)drm-mm.rst`, line 433)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/gpu/drm/drm_prime.c
   :doc: PRIME Helpers
```

PRIME Function References

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\(linux-master) (Documentation) (gpu)drm-mm.rst`, line 439)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/drm/drm_prime.h
   :internal:
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\(linux-master) (Documentation) (gpu)drm-mm.rst`, line 442)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/gpu/drm/drm_prime.c
   :export:
```

DRM MM Range Allocator

Overview

**System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-

master\Documentation\gpu\(linux-master) (Documentation) (gpu)drm-mm.rst, line 451)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/gpu/drm/drm_mm.c
   :doc: Overview
```

LRU Scan/Eviction Support

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\(linux-master) (Documentation) (gpu)drm-mm.rst, line 457)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/gpu/drm/drm_mm.c
   :doc: lru scan roster
```

DRM MM Range Allocator Function References

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\(linux-master) (Documentation) (gpu)drm-mm.rst, line 463)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/drm/drm_mm.h
   :internal:
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\(linux-master) (Documentation) (gpu)drm-mm.rst, line 466)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/gpu/drm/drm_mm.c
   :export:
```

DRM Cache Handling and Fast WC memcpy()

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\(linux-master) (Documentation) (gpu)drm-mm.rst, line 472)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/gpu/drm/drm_cache.c
   :export:
```

DRM Sync Objects

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\(linux-master) (Documentation) (gpu)drm-mm.rst, line 478)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/gpu/drm/drm_syncobj.c
   :doc: Overview
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\(linux-master) (Documentation) (gpu)drm-mm.rst, line 481)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/drm/drm_syncobj.h
   :internal:
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-

master\Documentation\gpu\ (linux-master) (Documentation) (gpu)drm-mm.rst, line 484)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/gpu/drm/drm_syncobj.c
   :export:
```

GPU Scheduler

Overview

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\ (linux-master) (Documentation) (gpu)drm-mm.rst, line 493)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/gpu/drm/scheduler/sched_main.c
   :doc: Overview
```

Scheduler Function References

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\ (linux-master) (Documentation) (gpu)drm-mm.rst, line 499)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/drm/gpu_scheduler.h
   :internal:
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\ (linux-master) (Documentation) (gpu)drm-mm.rst, line 502)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/gpu/drm/scheduler/sched_main.c
   :export:
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\gpu\ (linux-master) (Documentation) (gpu)drm-mm.rst, line 505)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/gpu/drm/scheduler/sched_entity.c
   :export:
```