Buffer Sharing and Synchronization

The dma-buf subsystem provides the framework for sharing buffers for hardware (DMA) access across multiple device drivers and subsystems, and for synchronizing asynchronous hardware access.

This is used, for example, by drm "prime" multi-GPU support, but is of course not limited to GPU use cases.

The three main components of this are: (1) dma-buf, representing a sg_table and exposed to userspace as a file descriptor to allow passing between devices, (2) fence, which provides a mechanism to signal when one device has finished access, and (3) reservation, which manages the shared or exclusive fence(s) associated with the buffer.

Shared DMA Buffers

This document serves as a guide to device-driver writers on what is the dma-buf buffer sharing API, how to use it for exporting and using shared buffers.

Any device driver which wishes to be a part of DMA buffer sharing, can do so as either the 'exporter' of buffers, or the 'user' or 'importer' of buffers.

Say a driver A wants to use buffers created by driver B, then we call B as the exporter, and A as buffer-user/importer.

The exporter

- implements and manages operations in :c:type:`struct dma_buf_ops <dma_buf_ops>` for the buffer,

  > **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\[linux-master][Documentation][driver-api]dma-buf.rst`, line 31); *backlink*
  >
  > Unknown interpreted text role "c:type".

- allows other users to share the buffer by using dma_buf sharing APIs,
- manages the details of buffer allocation, wrapped in a :c:type:`struct dma_buf <dma_buf>`,

  > **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\[linux-master][Documentation][driver-api]dma-buf.rst`, line 34); *backlink*
  >
  > Unknown interpreted text role "c:type".

- decides about the actual backing storage where this allocation happens,
- and takes care of any migration of scatterlist - for all (shared) users of this buffer.

The buffer-user

- is one of (many) sharing users of the buffer.
- doesn't need to worry about how the buffer is allocated, or where.
- and needs a mechanism to get access to the scatterlist that makes up this buffer in memory, mapped into its own address space, so it can access the same area of memory. This interface is provided by :c:type:`struct dma_buf_attachment <dma_buf_attachment>`.

  > **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\[linux-master][Documentation][driver-api]dma-buf.rst`, line 44); *backlink*
  >
  > Unknown interpreted text role "c:type".

Any exporters or users of the dma-buf buffer sharing framework must have a 'select DMA_SHARED_BUFFER' in their respective Kconfigs.

Userspace Interface Notes

Mostly a DMA buffer file descriptor is simply an opaque object for userspace, and hence the generic interface exposed is very minimal. There's a few things to consider though:

- Since kernel 3.12 the dma-buf FD supports the llseek system call, but only with offset=0 and whence=SEEK_END|SEEK_SET. SEEK_SET is supported to allow the usual size discover pattern size = SEEK_END(0); SEEK_SET(0). Every other llseek operation will report -EINVAL.

If llseek on dma-buf FDs isn't support the kernel will report -ESPIPE for all cases. Userspace can use this to detect support for discovering the dma-buf size using llseek.

- In order to avoid fd leaks on exec, the FD_CLOEXEC flag must be set on the file descriptor. This is not just a resource leak, but a potential security hole. It could give the newly exec'd application access to buffers, via the leaked fd, to which it should otherwise not be permitted access.

  The problem with doing this via a separate fcntl() call, versus doing it atomically when the fd is created, is that this is inherently racy in a multi-threaded app[3]. The issue is made worse when it is library code opening/creating the file descriptor, as the application may not even be aware of the fd's.

  To avoid this problem, userspace must have a way to request O_CLOEXEC flag be set when the dma-buf fd is created. So any API provided by the exporting driver to create a dmabuf fd must provide a way to let userspace control setting of O_CLOEXEC flag passed in to dma_buf_fd().
- Memory mapping the contents of the DMA buffer is also supported. See the discussion below on CPU Access to DMA Buffer Objects for the full details.
- The DMA buffer FD is also pollable, see Implicit Fence Poll Support below for details.
- The DMA buffer FD also supports a few dma-buf-specific ioctls, see DMA Buffer ioctls below for details.

Basic Operation and Device DMA Access

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\[linux-master] [Documentation] [driver-api]dma-buf.rst`, line 97)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/dma-buf/dma-buf.c
   :doc: dma buf device access
```

CPU Access to DMA Buffer Objects

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\[linux-master] [Documentation] [driver-api]dma-buf.rst`, line 103)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/dma-buf/dma-buf.c
   :doc: cpu access
```

Implicit Fence Poll Support

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\[linux-master] [Documentation] [driver-api]dma-buf.rst`, line 109)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/dma-buf/dma-buf.c
   :doc: implicit fence polling
```

DMA-BUF statistics

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\[linux-master] [Documentation] [driver-api]dma-buf.rst`, line 114)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/dma-buf/dma-buf-sysfs-stats.c
   :doc: overview
```

DMA Buffer ioctls

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\[linux-master] [Documentation] [driver-api]dma-buf.rst`, line 120)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/uapi/linux/dma-buf.h
```

Kernel Functions and Structures Reference

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\[linux-master] [Documentation] [driver-api]dma-buf.rst`, line 125)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/dma-buf/dma-buf.c
   :export:
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\[linux-master] [Documentation] [driver-api]dma-buf.rst`, line 128)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/linux/dma-buf.h
   :internal:
```

Reservation Objects

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\[linux-master] [Documentation] [driver-api]dma-buf.rst`, line 134)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/dma-buf/dma-resv.c
   :doc: Reservation Object Overview
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\[linux-master] [Documentation] [driver-api]dma-buf.rst`, line 137)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/dma-buf/dma-resv.c
   :export:
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\[linux-master] [Documentation] [driver-api]dma-buf.rst`, line 140)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/linux/dma-resv.h
   :internal:
```

DMA Fences

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\[linux-master] [Documentation] [driver-api]dma-buf.rst`,

line 146)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/dma-buf/dma-fence.c
   :doc: DMA fences overview
```

DMA Fence Cross-Driver Contract

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\[linux-master][Documentation][driver-api]dma-buf.rst`, line 152)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/dma-buf/dma-fence.c
   :doc: fence cross-driver contract
```

DMA Fence Signalling Annotations

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\[linux-master][Documentation][driver-api]dma-buf.rst`, line 158)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/dma-buf/dma-fence.c
   :doc: fence signalling annotation
```

DMA Fences Functions Reference

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\[linux-master][Documentation][driver-api]dma-buf.rst`, line 164)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/dma-buf/dma-fence.c
   :export:
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\[linux-master][Documentation][driver-api]dma-buf.rst`, line 167)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/linux/dma-fence.h
   :internal:
```

DMA Fence Array

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\[linux-master][Documentation][driver-api]dma-buf.rst`, line 173)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/dma-buf/dma-fence-array.c
   :export:
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\[linux-master][Documentation][driver-api]dma-buf.rst`, line 176)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/linux/dma-fence-array.h
   :internal:
```

DMA Fence Chain

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\[linux-master] [Documentation] [driver-api]dma-buf.rst`, line 182)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/dma-buf/dma-fence-chain.c
   :export:
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\[linux-master] [Documentation] [driver-api]dma-buf.rst`, line 185)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/linux/dma-fence-chain.h
   :internal:
```

DMA Fence uABI/Sync File

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\[linux-master] [Documentation] [driver-api]dma-buf.rst`, line 191)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/dma-buf/sync_file.c
   :export:
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\[linux-master] [Documentation] [driver-api]dma-buf.rst`, line 194)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/linux/sync_file.h
   :internal:
```

Indefinite DMA Fences

At various times struct dma_fence with an indefinite time until dma_fence_wait() finishes have been proposed. Examples include:

- Future fences, used in HWC1 to signal when a buffer isn't used by the display any longer, and created with the screen update that makes the buffer visible. The time this fence completes is entirely under userspace's control.
- Proxy fences, proposed to handle &drm_syncobj for which the fence has not yet been set. Used to asynchronously delay command submission.
- Userspace fences or gpu futexes, fine-grained locking within a command buffer that userspace uses for synchronization across engines or with the CPU, which are then imported as a DMA fence for integration into existing winsys protocols.
- Long-running compute command buffers, while still using traditional end of batch DMA fences for memory management instead of context preemption DMA fences which get reattached when the compute job is rescheduled.

Common to all these schemes is that userspace controls the dependencies of these fences and controls when they fire. Mixing indefinite fences with normal in-kernel DMA fences does not work, even when a fallback timeout is included to protect against malicious userspace:

- Only the kernel knows about all DMA fence dependencies, userspace is not aware of dependencies injected due to memory management or scheduler decisions.
- Only userspace knows about all dependencies in indefinite fences and when exactly they will complete, the kernel has no visibility.

Furthermore the kernel has to be able to hold up userspace command submission for memory management needs, which means we must support indefinite fences being dependent upon DMA fences. If the kernel also support indefinite fences in the kernel like a

DMA fence, like any of the above proposal would, there is the potential for deadlocks.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\[linux-master][Documentation][driver-api]dma-buf.rst`, line 236)

Unknown directive type "kernel-render".

```
.. kernel-render:: DOT
   :alt: Indefinite Fencing Dependency Cycle
   :caption: Indefinite Fencing Dependency Cycle

   digraph "Fencing Cycle" {
      node [shape=box bgcolor=grey style=filled]
      kernel [label="Kernel DMA Fences"]
      userspace [label="userspace controlled fences"]
      kernel -> userspace [label="memory management"]
      userspace -> kernel [label="Future fence, fence proxy, ..."]

      { rank=same; kernel userspace }
   }
```

This means that the kernel might accidentally create deadlocks through memory management dependencies which userspace is unaware of, which randomly hangs workloads until the timeout kicks in. Workloads, which from userspace's perspective, do not contain a deadlock. In such a mixed fencing architecture there is no single entity with knowledge of all dependencies. Thefore preventing such deadlocks from within the kernel is not possible.

The only solution to avoid dependencies loops is by not allowing indefinite fences in the kernel. This means:

- No future fences, proxy fences or userspace fences imported as DMA fences, with or without a timeout.
- No DMA fences that signal end of batchbuffer for command submission where userspace is allowed to use userspace fencing or long running compute workloads. This also means no implicit fencing for shared buffers in these cases.

Recoverable Hardware Page Faults Implications

Modern hardware supports recoverable page faults, which has a lot of implications for DMA fences.

First, a pending page fault obviously holds up the work that's running on the accelerator and a memory allocation is usually required to resolve the fault. But memory allocations are not allowed to gate completion of DMA fences, which means any workload using recoverable page faults cannot use DMA fences for synchronization. Synchronization fences controlled by userspace must be used instead.

On GPUs this poses a problem, because current desktop compositor protocols on Linux rely on DMA fences, which means without an entirely new userspace stack built on top of userspace fences, they cannot benefit from recoverable page faults. Specifically this means implicit synchronization will not be possible. The exception is when page faults are only used as migration hints and never to on-demand fill a memory request. For now this means recoverable page faults on GPUs are limited to pure compute workloads.

Furthermore GPUs usually have shared resources between the 3D rendering and compute side, like compute units or command submission engines. If both a 3D job with a DMA fence and a compute workload using recoverable page faults are pending they could deadlock:

- The 3D workload might need to wait for the compute job to finish and release hardware resources first.
- The compute workload might be stuck in a page fault, because the memory allocation is waiting for the DMA fence of the 3D workload to complete.

There are a few options to prevent this problem, one of which drivers need to ensure:

- Compute workloads can always be preempted, even when a page fault is pending and not yet repaired. Not all hardware supports this.
- DMA fence workloads and workloads which need page fault handling have independent hardware resources to guarantee forward progress. This could be achieved through e.g. through dedicated engines and minimal compute unit reservations for DMA fence workloads.
- The reservation approach could be further refined by only reserving the hardware resources for DMA fence workloads when they are in-flight. This must cover the time from when the DMA fence is visible to other threads up to moment when fence is completed through dma_fence_signal().
- As a last resort, if the hardware provides no useful reservation mechanics, all workloads must be flushed from the GPU when switching between jobs requiring DMA fences or jobs requiring page fault handling: This means all DMA fences must complete before a compute job with page fault handling can be inserted into the scheduler queue. And vice versa, before a DMA fence can be made visible anywhere in the system, all compute workloads must be preempted to guarantee all pending GPU page faults are flushed.
- Only a fairly theoretical option would be to untangle these dependencies when allocating memory to repair hardware page faults, either through separate memory blocks or runtime tracking of the full dependency graph of all DMA fences. This results very wide impact on the kernel, since resolving the page on the CPU side can itself involve a page fault. It is much more

feasible and robust to limit the impact of handling hardware page faults to the specific driver.

Note that workloads that run on independent hardware like copy engines or other GPUs do not have any impact. This allows us to keep using DMA fences internally in the kernel even for resolving hardware page faults, e.g. by using copy engines to clear or copy memory needed to resolve the page fault.

In some ways this page fault problem is a special case of the *Infinite DMA Fences* discussions: Infinite fences from compute workloads are allowed to depend on DMA fences, but not the other way around. And not even the page fault problem is new, because some other CPU thread in userspace might hit a page fault which holds up a userspace fence - supporting page faults on GPUs doesn't anything fundamentally new.