Allocator Attribution for DMA-BUFs in Android
Linux Plumbers Conference 2021
Android Microconference
Problems that we are trying to solve

- No way to limit the total amount of DMA-BUF memory allocated on behalf of a process.
- Origins of buffer leaks hard to identify.
Why not...

- Use the memcg cgroup controller?
  - Per-app memcgs have considerable **overhead** and enabling them only for tracking DMA-BUF usage would be too high of a cost.
  - 15% minor page fault path performance regression reported by partners with per-app memcgs using page fault test benchmarks.
  - Details of a per-app memcg performance regression reported upstream can be seen [here](#).
Why not...

- Use the memcg cgroup controller?
  - memcg performs accounting in units of page. In the DMA-BUF buffer sharing model, a process takes a reference to the entire buffer (hence keeping it alive) even if it is only accessing parts of it. Per-page memory tracking feels like an unnecessary overhead for DMA-BUF memory accounting.

  - There is also no need to use cgroups to track which processes are holding fd/map references to a DMA-BUF since this information is already available from procfs.
Why not...

- A userspace service to keep track of buffer allocations and release?
  - Allocation done using DMA-BUF heap IOCTLs.
  - Buffer release happens when the last reference to the buffer dropped.
  - No way for a userspace service to intercept either allocation or release.
  - In case the process gets killed/restarted, we lose all accounting so far.
Why not...

- A new cgroup controller?
  - Efforts to add a GPU cgroup controller already in progress upstream!
  - Authored by Kenny Ho and Brian Welty!
Evaluating the GPU cgroup controller for Android

- API from latest RFC is closely tied to the DRM framework.

```c
int drm_cgroup_try_charge(struct drmcg *drmcg, struct drm_device *dev,
                           enum drmcg_res_type type, u64 usage);
void drm_cgroup_uncharge(struct drmcg *drmcg, struct drm_device *dev,
                        enum drmcg_res_type type, u64 usage);
```
Proposed Solutions

- Modify the API to be generic
  - Ensuring that it works for DRM while also accommodating use by DMA-BUF heaps.
  - Allow usage by non-GPU/graphics DMA-BUFs (such as those used by a camera driver).

- Perhaps resembling the following:

```c
int buffer_cg_try_charge(struct buffer_cg *buffer_cg,  
                         struct buffer_cg_device *device, u64 usage);
void buffer_cg_uncharge(struct buffer_cg *gpu_cg, struct buffer_cg_device *device,  
                         u64 usage);
int buffer_cg_register_device(struct buffer_cg_device *buffer_cg_dev);
void buffer_cg_unregister_device(struct buffer_cg_device *buffer_cg_dev);
```
Evaluating the GPU cgroup controller for Android

- Buffer is charged to allocating process and no way to move the charge once allocated.

- Majority of graphics allocations happen through Gralloc HAL process in Android.
  - Gralloc HAL presents a unified API to client.
  - Integral to the system/vendor separation paradigm in Android.
  - On a client request, Gralloc HAL allocates a buffer and sends the DMA-BUF fd to the client over IPC.
  - It does not retain any references to the buffer.
Proposed Solutions

- Find a way to charge a buffer to a cgroup other than own.

Option 1

- Explicit charge migration
  - Use the cgroup interface to move charge of a buffer to a different cgroup.
  - For example: writing the dmabuf fd to
    
    ```
    /sys/kernel/fs/cg1/cgroup.gpu.dma_buf_to_charge
    ```
  - Here dmabuf fd is the fd to the buffer held by the writing process.
  - Not upstreamable as per initial discussions with cgroup maintainers.
Proposed Solutions

- Find a way to charge a buffer to a cgroup other than own.

Option 2
- Use a mechanism similar to fadvise with FADV_DONTNEED where that allocator can declare that it will not use the buffer. The buffer will then be charged to the process who accesses it.

Issues
- Results are non-deterministic.
  - The process who receives the fd over IPC might not map/install the fd and pass it over to another process.
  - The buffer’s size would not apply towards the limit of the process who requested the allocation.
Proposed Solutions

● Find a way to charge a buffer to a cgroup other than own.

Option 3

● New DMA-BUF Heap allocation IOCTL that takes as argument fd to cgroup of client process
  ○ Charging to the client happens in IOCTL handler.
  ○ Sepolicy sufficient to guarantee security?
We are open to collaboration!

- Please reach out to us at android-kernel-team@google.com.
THANK YOU!