A brief overview of DRM/KMS and its status in NetBSD

Taylor 'Riastradh' Campbell
campbell@mumble.net
riastradh@NetBSD.org

AsiaBSDcon 2015
Tokyo, Japan
March 13, 2015

No, not that DRM!

- DRM: Direct rendering manager: Linux kernel interface for commanding GPU to render directly into framebuffer for display.
- Originally, DRM was only a kernel interface for mapping graphics card MMIO registers and waiting for vertical blank interrupts.
- Actual driver for display lived in userland: used DRM to disable kernel's idea of VGA console and grant exclusive access to display registers to X server, peeked and poked them in userland to detect and configure displays.
- Userland used legacy `/dev/agp` device to allocate physical memory for graphics and program it into the GPU's page tables.

DRM/KMS: DRM with a real kernel display driver

- Maybe userland shouldn't be mapping the device's MMIO registers, handling mode-setting, etc.: 'user mode-setting', or UMS.
- Would be nice if kernel could suspend/resume display without X's help.
- DRM/KMS: DRM with kernel mode-setting.

GEM and TTM: Graphics buffer management

- GEM: Graphics Extent Manager
- TTM: Texture and Tiling Manager
- Fancy names for two different sets of ioctls to manage swappable buffers shared by CPU and GPU.

DRM portability

- DRM implementation maintained in Linux.
- Used to be a coordinated porting effort to BSDs.
- Lost coordination in switch from UMS to KMS.
- New ports to *BSD all different now!
- NetBSD: shims to make most Linux code run unmodified and updates less painful.
- FreeBSD: modify all the Linux code, including indentation.
- OpenBSD and DragonflyBSD: somewhere in the middle.

Problems

- Userland can still wedge GPU.
- Linux kernel code is very large:

```
% wc -l drm/*.c
   29149 total
% wc -l drm/i915/*.c
   76242 total
% wc -l drm/nouveau/**/*.c
   95675 total
% wc -l drm/radeon/*.c
  152315 total
```

- . . . and I made some stupid mistakes porting it.

Status

- Intel graphics: works, minor bugs in display detection on some devices, minor rendering glitches on some devices.
- Radeon: works.
- Nouveau: compiles but does not work yet.
- Everything is much better as of this month after I fixed three stupid bugs I caused ages ago. . .

Bug 1: Timed waits: Linux code

- Linux has no easy API for interlocked waits.

```
unsigned long start = jiffies;
unsigned long end = start + timeout
unsigned long now;
DEFINE_WAIT(wait);
int ret;

for (;;) {
        prepare_to_wait(&dev->waitq, &wait,
            TASK_INTERRUPTIBLE);
        if (signal_pending(current)) {
                ret = -ERESTARTSYS;
                break;
        }
        ...
```

Bug 1: Timed waits: Linux code

```
...
now = jiffies;
if (now > end) {
        ret = (CONDITION) ? 1 : 0;
        break;
}
if (CONDITION) {
        ret = MAX(end - now, 1);
        break;
}
...
```

Bug 1: Timed waits: Linux code

```
            ...
            ret = schedule_timeout(timeout);
            if (ret < 0)
                    break;
            timeout = ret;
    }
    finish_wait(&dev->waitq, &wait);

    return ret;
```

- Where's the lock to read `dev->done` excluding interrupts?
- You're on your own.
- Every driver does it differently, usually with a complicated (read: wrong) dance involving atomics.

Bug 1: Timed waits: Linux code simplified

- Linux has a collection of macros to do this for you:

```
ret = wait_event(dev->waitq, dev->done)
ret = wait_event_interruptible(dev->waitq,
    dev->done);
ret = wait_event_timeout(dev->waitq, dev->done,
    timeout);
```

- Return negative error on interrupt.
- Return zero on success. . . if no timeout.
- Return *positive* on success if there is a timeout.
- Return zero on timeout.
- (What about lock for `dev->done`? Still on your own.)

Bug 1: Timed waits: Linux DRM code

- Old DRM code from last decade used a portability macro `DRM_WAIT_ON`:

  ```
  DRM_WAIT_ON(ret, dev->waitq, timeout, dev->done);
  ```

- Return negative error on interrupt.
- Return negative error `-ETIME` on timeout.
- Return zero on success.
- (Also: poll every tick, just for good measure.)
- (What about lock for `dev->done`? Still on your own.)

Bug 1: Timed waits: NetBSD code

- NetBSD has:

```
while (!dev->done) {
        error = cv_timedwait_sig(&dev->waitq,
            &dev->lock, timeout);
        if (error)
                return error;
        now = hardclock_ticks;
        timeout -= now - start;
        start = now;
}
```

Bug 1: Timed waits: NetBSD code

- No non-interlocked timed waits: no dances with atomics and no race conditions.
- Required putting in device interrupt spin locks where appropriate, since Linux doesn't have them.
- Return `EINTR`/`ERESTART` on interrupt.
- Return `EWOULDBLOCK` on timeout.
- Return zero on success.

Bug 1: Timed waits

- I focussed on getting locks correct for interlocked waits.
- Didn't pay enough attention to the return codes.
- Totally mixed them all up.
- Waits for i2c commands, graphics commands – always timed out or returned early.
- Sometimes worked by accident, hard to diagnose.
- Oops.

Bug 2: Cache-flushing needs memory barriers

- Intel `CLFLUSH` instruction flushes cache lines.

```
size_t clflush_size =
    cpu_info_primary.ci_cflush_lsize;
vaddr_t p;

for (p = start; p < end; p += clflush_size)
        x86_clflush(p);
```

- That should do it, right?

Bug 2: Cache-flushing needs memory barriers

- Intel `CLFLUSH` instruction flushes cache lines.

```
size_t clflush_size =
    cpu_info_primary.ci_cflush_lsize;
vaddr_t p;

x86_mfence();
for (p = start; p < end; p += clflush_size)
        x86_clflush(p);
x86_mfence();
```

- Except it is not instruction-ordered. It is ordered only by `MFENCE`. I forgot `MFENCE`. Oops.

Bug 3: Cacheability flags

- Entries in the GPU page table, or graphics translation table 'GTT', have cacheability flags.
- Everything should correct – but slow – if we disable caching, right?

Bug 3: Cacheability flags

- Held off turning on these bits for months while trying to find the source of unusable rendering glitches.
- Figured turning on caching would make things worse.
- Turned on the bits. Everything worked.
- Oops.