闪屏花屏问题分析
内核与用户态显示切换用户涉及的知识点

演讲人：郭隆基

统信软件技术有限公司

January 5, 2024

目录

目录

问题背景

- 切换用户、锁屏界面登陆系统时，使用 Raedon Oland 显卡机型性能劣化
  - 为什么切换用户的时候有概率会黑一下屏？

问题背景

- 切换用户、锁屏界面登陆系统时，使用 Raedon Oland 显卡机型性能劣化
  - 为什么切换用户的时候有概率会黑一下屏？
- 多用户切换测试高概率闪屏
  - 和上面是同一个问题，极快的闪屏没有找到问题可以分析的路径

分析思路

分析闪屏问题的一般思路

具体步骤

- 扫盲；了解内核和应用层相关的模块
- 猜；跟据了解的知识猜测可能的方向，不一定准确。现象多而且复杂一定要做排除，很多现象是干扰，甚至和调式环境有关。
- 复现；找多种复现方式和手段，找到出问题与不出问题的区别
- 分析；方向准确后分析内核和应用层模块代码逻辑。也用一些手段做验证

调试手段

Figure: bpftrace 真好用

在分析我这个问题时我用的调试方法有 bpftrace gdb strace

熟练应用 bpftrace 编程和 gdb 的调试是解决问题提速的加分项，要学会写 bpfrace 程序。如果是主要看内核中的流程，那 qemu 也能提高效率。

目录

Edid

在 Oland 的显卡中有一个 VGA 循环读取 Edid 检测 VGA 是否有插拔的内核进程，一度以为是用户切换调用 drm_ctrc_set_config 也读取 edid ，两个读取 edid 冲突时就会造成黑屏。证伪的方法很简单，把这个循环时间调长。从 10s 调到 3600s ，发现问题仍然存在。之前的分析在：
https://guolongji.xyz/post/tty-x-switch/

其它干扰

举例

- 在 drm_do_probe_dde_edid 调用栈的函数中加一些打印，会影响问题复现的概率。
- 安装相关的 gdb 调式包会影响问题复现的概率，有时会让问题完全不复现。
- 因为切入，切出的 xorg 都会收到 SIGUSR1 信号。用 `kill -SIGUSR1 pid` 来模拟。这样测试有时也会黑屏，但与问题关系不大。

总结

总结

EEPROM 的读速度

- EEPROM 的读取速度和 FLASH 、SRAM 相比虽然慢一点，但是也差不了多少，只是写的速度很慢。

两个读取都是通过 DDC 读 EDID

- VGA 的 `INIT_DELAYED_WORK(dev->mode_config.output_poll_work, output_poll_execute);` 可以修改 `#define DRM_OUTPUT_POLL_PERIOD (10*HZ)` 将 10 改成 3600 。
- 另一个是通过 `drm_do_probe_dde_edid` 读取。
- 因为同步导致的花屏闪屏，调整上下文的时序，增加延时会导致现象的不同。

例子中的干扰项有一点迷惑性，要排除这些干扰需要经验的积累。

目录

bpftrace 分析关键函数

使用 sudo stackcount-bpfcc drm_crtc_helper_set_config 多次测试，使用 bpf 分析调用，发现出问题的时候就出现下面的调用栈：

调用栈

```
drm_crtc_helper_set_config
amdgpu_display_crtc_set_config
___drm_mode_set_config_internal
drm_crtc_force_disable
drm_framebuffer_remove
drm_mode_rmfb_work_fn
process_one_work
worker_thread
kthread
ret_from_fork
  kworker/4:3 [8160]
```

问题很容易复现，关键就是解释这个逻辑的过程

一句话

用一句话来解释的话就是 setcrtc 是异步的，而 Oland 显卡执行这个的时间不确定。Oland 显卡使用的 DRM_IOCTL_MODE_RMFB 这个 ioctl 不是原子性的，drm_framebuffer_remove 调用的 legacy_remove_fb 也不是原子性的，所以无法失败回退。不能重新调用执行正确的逻辑怎么办？可以为内核增加容错的逻辑用一种有不太好的方式掩盖更大的问题。

xorg 收到信号后会调用 setcrtc 的 ioctl，整体执行的时间比较长且此函数异步，下面又调用 rmfb 的 ioctl，导致 sigusr1 处于 pending 状态的时间也比较长。rmfb 的 ioctl 如要使用 drm_framebuffer_remove 的话，会返回 -EINTR 。xorg 没有重新调用 rmfb 这个 ioctl 的逻辑那就不用这个函数，另外写一个处理过程，再内核空间做完 rmfb 的动作。
不支持原子 setcrtc 的旧架构的显卡，可能读写寄存器有时快一点有时慢一点，就会有时导致 drm_framebuffer_read_refcount(fb) > 1 在这种情况下就不是调用 drm_framebuffer_put(fb) 做释放。而是让内核在内核空间的共享任务队列当中创建一个任务，以解决可能的逻辑问题。这个函数执行的时间比较长，就会黑屏。

具体细节在 drm_mode_rmfb 这个函数当中。这个问题是机制的问题，并不好解决。

xorg 断点一

下面是切出的 xorg 收到信号和 drmModeSetCrtc 断点的堆栈。切出的时候会调用 setcrtc 的 ioctl 一次。会调用三次 rmfb （和创建的有户数相等）。所以是切出的时候导致的黑屏。

```
(gdb) b drmModeSetCrtc
Breakpoint 1 at 0x7f649eb0e5d0
(gdb) c
Continuing.

Thread 1 "Xorg" received signal SIGUSR1, User defined signal 1.
0x00007f649e6c6aff in epoll_wait (epfd=3, events=events@entry=0x7ffc70454b90, maxevents=maxevents@entry=256, timeout=-1) at ../sysdeps/unix/sysv/linux/epoll_wait.c:30
30      in ../sysdeps/unix/sysv/linux/epoll_wait.c
(gdb) c
Continuing.

Thread 1 "Xorg" hit Breakpoint 1, 0x00007f649eb0e5d0 in drmModeSetCrtc () from /lib/x86_64-linux-gnu/libdrm.so.2
(gdb) bt
#0  0x00007f649eb0e5d0 in drmModeSetCrtc () from /lib/x86_64-linux-gnu/libdrm.so.2
#1  0x00007f649df90508 in drmmode_set_mode (crtc=crtc@entry=0x2601cf0, fb=fb@entry=0x319b6a0, mode=mode@entry=0x2601d08, x=x@entry=0, y=y@entry=0) at ../../src/drmmode_display.c:1353
#2  0x00007f649df8b445 in AMDGPULeaveVT_KMS (pScrn=0x1edc200) at ../../src/amdgpu_kms.c:2202
#3  0x0000000000049012c in xf86XVLeaveVT (pScrn=0x1edc200) at ../../../../../../hw/xfree86/common/xf86xv.c:1226
#4  0x00007f649e2df449 in glxDRILeaveVT (scrn=0x1edc200) at ../../../../glx/glxdri2.c:815
#5  0x0000000000047bc95 in xf86VTLeave () at ../../../../../../hw/xfree86/common/xf86Events.c:396
#6  0x0000000000443435 in WakeupHandler (result=result@entry=-1) at ../../../../dix/dixutils.c:429
#7  0x0000000000598bb1 in WaitForSomething (are_ready=0) at ../../../../os/WaitFor.c:210
#8  0x000000000043e6cc in Dispatch () at ../../../../include/list.h:220
#9  0x0000000000442906 in dix_main (argc=12, argv=0x7ffc704559c8, envp=<optimized out>) at ../../../../dix/main.c:276
#10 0x00007f649e5f11fb in __libc_start_main (main=0x42b6a0 <main>, argc=12, argv=0x7ffc704559c8, init=<optimized out>, fini=<optimized out>, rtld_fini=<optimized out>, stack_end=0x7ffc704559b8)
    at ../csu/libc-start.c:308
#11 0x000000000042b6da in _start () at ../../../../../../hw/xfree86/common/xf86Events.c:638
(gdb) c
Continuing.
```

xorg 断点二

下面是切入的 xorg 收到信号和 drmModeSetCrtc 断点的堆栈，有黑屏切入的时候会调用 setcrtc 的 ioctl 两次，不会调用 rmfb 。

```
Thread 1 "Xorg" hit Breakpoint 1, 0x00007f377dea25d0 in drmModeSetCrtc () from /lib/x86_64-linux-gnu/libdrm.so.2
(gdb) c
Continuing.

Thread 1 "Xorg" received signal SIGUSR1, User defined signal 1.
0x00007f377da5aaff in epoll_wait (epfd=3, events=0x7ffd2944b860, maxevents=256, timeout=588677) at ../sysdeps/unix/sysv/linux/epoll_wait.c:30
30      in ../sysdeps/unix/sysv/linux/epoll_wait.c
(gdb) c
Continuing.
Thread 1 "Xorg" hit Breakpoint 1, 0x00007f377dea25d0 in drmModeSetCrtc () from /lib/x86_64-linux-gnu/libdrm.so.2
(gdb) bt
#0  0x00007f377dea25d0 in drmModeSetCrtc () from /lib/x86_64-linux-gnu/libdrm.so.2
#1  0x00007f377d3237e3 in drmmode_crtc_dpms (crtc=0x250b0e0, mode=3) at ../../src/drmmode_display.c:405
#2  0x00007f377d327119 in drmmode_set_desired_modes (pScrn=pScrn@entry=0x1de8070, drmmode=drmmode@entry=0x1de94e8, set_hw=set_hw@entry=1) at ../../src/drmmode_display.c:3641
#3  0x00007f377d31f12d in AMDGPUEnterVT_KMS (pScrn=0x1de8070) at ../../src/amdgpu_kms.c:2141
#4  0x000000000048e4e0 in ?? ()
#5  0x00000000004b8f78 in ?? ()
#6  0x00000000004834d9 in ?? ()
#7  0x00007f377d6734af in ?? () from /usr/lib/xorg/modules/extensions/libglx.so
#8  0x000000000047bf29 in xf86VTEnter ()
#9  0x0000000000443435 in WakeupHandler ()
#10 0x00000000000598bb1 in WaitForSomething ()
#11 0x000000000043e6cc in ?? ()
#12 0x00000000000442906 in ?? ()
#13 0x00007f377d9851fb in __libc_start_main (main=0x42b6a0, argc=12, argv=0x7ffd2944c698, init=<optimized out>, fini=<optimized out>, rtld_fini=<optimized out>, stack_end=0x7ffd2944c688)
    at ../csu/libc-start.c:308
#14 0x000000000042b6da in _start ()
(gdb) c
Continuing.
Thread 1 "Xorg" hit Breakpoint 1, 0x00007f377dea25d0 in drmModeSetCrtc () from /lib/x86_64-linux-gnu/libdrm.so.2
(gdb) bt
#0  0x00007f377dea25d0 in drmModeSetCrtc () from /lib/x86_64-linux-gnu/libdrm.so.2
#1  0x00007f377d324508 in drmmode_set_mode (crtc=crtc@entry=0x250a4e0, fb=fb@entry=0x2a33370, mode=mode@entry=0x250a5c8, x=x@entry=0, y=y@entry=0) at ../../src/drmmode_display.c:1353
#2  0x00007f377d324a6d in drmmode_set_mode_major (crtc=0x250a4e0, mode=0x250a5c8, rotation=<optimized out>, x=<optimized out>, y=<optimized out>) at ../../src/drmmode_display.c:1452
#3  0x00007f377d3271b7 in drmmode_set_desired_modes (pScrn=pScrn@entry=0x1de8070, drmmode=drmmode@entry=0x1de94e8, set_hw=set_hw@entry=1) at ../../src/drmmode_display.c:3688
#4  0x00007f377d31f12d in AMDGPUEnterVT_KMS (pScrn=0x1de8070) at ../../src/amdgpu_kms.c:2141
#5  0x000000000048e4e0 in ?? ()
#6  0x00000000004b8f78 in ?? ()
#7  0x00000000004834d9 in ?? ()
#8  0x00007f377d6734af in ?? () from /usr/lib/xorg/modules/extensions/libglx.so
#9  0x000000000047bf29 in xf86VTEnter ()
#10 0x0000000000443435 in WakeupHandler ()
#11 0x00000000000598bb1 in WaitForSomething ()
#12 0x000000000043e6cc in ?? ()
#13 0x00000000000442906 in ?? ()
#14 0x00007f377d9851fb in __libc_start_main (main=0x42b6a0, argc=12, argv=0x7ffd2944c698, init=<optimized out>, fini=<optimized out>, rtld_fini=<optimized out>, stack_end=0x7ffd2944c688)
    at ../csu/libc-start.c:308
#15 0x000000000042b6da in _start ()
```

内核逻辑

这里有一个比较绕的 fbs 参数传递的地方。

```
int drm_mode_rmfb(struct drm_device *dev, u32 fb_id,
        struct drm_file *file_priv)
{
    /* ... */
    /*
     * we now own the reference that was stored in the fbs list
     *
     * drm_framebuffer_remove may fail with -EINTR on pending signals,
     * so run this in a separate stack as there's no way to correctly
     * handle this after the fb is already removed from the lookup table.
     */
    if (drm_framebuffer_read_refcount(fb) > 1) {
        struct drm_mode_rmfb_work arg;

        INIT_WORK_ONSTACK(&arg.work, drm_mode_rmfb_work_fn);
        INIT_LIST_HEAD(&arg.fbs);
        list_add_tail(&fb->filp_head, &arg.fbs);

        schedule_work(&arg.work);
        flush_work(&arg.work);
        destroy_work_on_stack(&arg.work);
    } else
        drm_framebuffer_put(fb);
    /* ... */
}
```

内核逻辑

用户态每 open 一个 fb 并创建一个 fd 的时候，内核都会创建一个 drm_file 用于关联用户空间的 fd 。drm_file 有一个成员是 fbs ，用于存储与用户态的 fd 相关联的全部 fb 。

```
struct drm_mode_rmfb_work {
    struct work_struct work;
    struct list_head fbs;
};

static void drm_mode_rmfb_work_fn(struct work_struct *w)
{
    struct drm_mode_rmfb_work *arg = container_of(w, typeof(*arg), work);

    while (!list_empty(&arg->fbs)) {
        struct drm_framebuffer *fb =
            list_first_entry(&arg->fbs, typeof(*fb), filp_head);

        list_del_init(&fb->filp_head);
        drm_framebuffer_remove(fb);
    }
}
```

内核逻辑

直接调用 drm_framebuffer_remove 去删除 fb 的话，会被信号打断。

```
void drm_framebuffer_remove(struct drm_framebuffer *fb)
{
    /* ... */
    if (drm_framebuffer_read_refcount(fb) > 1) {
        if (drm_drv_uses_atomic_modeset(dev)) {
            int ret = atomic_remove_fb(fb);
            WARN(ret, "atomic remove_fb failed with %i\n", ret);
        } else
            legacy_remove_fb(fb);
    }

    drm_framebuffer_put(fb);
}
```

内核逻辑

对 crtc 和 plane 这些资源做释放。

```
static void legacy_remove_fb(struct drm_framebuffer *fb)
{
    struct drm_device *dev = fb->dev;
    struct drm_crtc *crtc;
    struct drm_plane *plane;

    drm_modeset_lock_all(dev);
    /* remove from any CRTC */
    drm_for_each_crtc(crtc, dev) {
        if (crtc->primary->fb == fb) {
            /* should turn off the crtc */
            if (drm_crtc_force_disable(crtc))
                DRM_ERROR("failed to reset crtc %p when fb was deleted\n", crtc);
        }
    }

    drm_for_each_plane(plane, dev) {
        if (plane->fb == fb)
            drm_plane_force_disable(plane);
    }
    drm_modeset_unlock_all(dev);
}
```

内核逻辑

setcrtc 的 ioctl 过程当中会增加 fb 的 refcount 。

```
static int ___drm__mode__set__config__internal(struct drm__mode__set *set,
                struct drm__modeset__acquire__ctx *ctx)
{
   /* ... */
   drm__for__each__crtc(tmp, crtc->dev) {
      struct drm__plane *plane = tmp->primary;

      if (plane->fb)
         drm__framebuffer__get(plane->fb);
      if (plane->old__fb)
         drm__framebuffer__put(plane->old__fb);
      plane->old__fb = NULL;
   }

   return ret;
}
```

内核逻辑

在 drm 框架当中，framebuffer 的引用计数在 drm_framebuffer_lookup 的时候会用，在用户空间 open 的时候会用，还有在内核 drm_client 做渲染的时候也会用。drm_mode_rmfb 的时候也要引用计数变为零之后才会释放。为了防止内核或系统卡死，framebuffer refcount 的处理要比较小心。

目录

切出的 xorg IOCTL 过程

切出会调用 rmfb ioctl 三次（和创建的用户数相等），不论是否黑屏。

```
--- SIGUSR1 {si_signo=SIGUSR1, si_code=SI_KERNEL} ---
ioctl(13, DRM_IOCTL_MODE_CURSOR, 0x7ffc704555d0) = 0
ioctl(14, DRM_IOCTL_AMDGPU_GEM_CREATE or DRM_IOCTL_VIA_ALLOCMEM, 0x7ffc70454da0) = 0
ioctl(14, _IOC(_IOC_READ|_IOC_WRITE, 0x64, 0x48, 0x28), 0x7ffc70454db0) = 0
ioctl(14, DRM_IOCTL_AMDGPU_GEM_METADATA, 0x7ffc70454fb0) = 0
ioctl(14, DRM_IOCTL_PRIME_HANDLE_TO_FD, 0x7ffc7045530c) = 0
ioctl(17, DRM_IOCTL_PRIME_FD_TO_HANDLE, 0x7ffc70455340) = 0
ioctl(14, DRM_IOCTL_PRIME_HANDLE_TO_FD, 0x7ffc7045530c) = 0
ioctl(17, DRM_IOCTL_PRIME_FD_TO_HANDLE, 0x7ffc70455340) = 0
ioctl(14, DRM_IOCTL_PRIME_HANDLE_TO_FD, 0x7ffc70455380) = 0
ioctl(17, DRM_IOCTL_PRIME_FD_TO_HANDLE, 0x7ffc704553cc) = 0
ioctl(14, DRM_IOCTL_PRIME_HANDLE_TO_FD, 0x7ffc704552bc) = 0
ioctl(17, DRM_IOCTL_PRIME_FD_TO_HANDLE, 0x7ffc704552fc) = 0
ioctl(14, DRM_IOCTL_PRIME_HANDLE_TO_FD, 0x7ffc704552bc) = 0
ioctl(17, DRM_IOCTL_PRIME_FD_TO_HANDLE, 0x7ffc704552fc) = 0
ioctl(14, DRM_IOCTL_PRIME_HANDLE_TO_FD, 0x7ffc704552ec) = 0
ioctl(14, DRM_IOCTL_PRIME_HANDLE_TO_FD, 0x7ffc704552dc) = 0
ioctl(17, DRM_IOCTL_PRIME_FD_TO_HANDLE, 0x7ffc7045531c) = 0
ioctl(13, DRM_IOCTL_PRIME_FD_TO_HANDLE, 0x7ffc7045555c) = 0
ioctl(13, DRM_IOCTL_AMDGPU_GEM_METADATA, 0x7ffc70455440) = 0
ioctl(13, DRM_IOCTL_MODE_ADDFB, 0x7ffc70455590) = 0
ioctl(14, DRM_IOCTL_AMDGPU_CS, 0x7ffc704552b0) = 0
ioctl(14, DRM_IOCTL_AMDGPU_CS, 0x7ffc70455320) = 0
ioctl(14, DRM_IOCTL_AMDGPU_WAIT_CS, 0x7ffc70455550) = 0
ioctl(13, DRM_IOCTL_MODE_SETCRTC, 0x7ffc70455510) = 0
ioctl(14, _IOC(_IOC_READ|_IOC_WRITE, 0x64, 0x48, 0x28), 0x7ffc704554f0) = 0
ioctl(14, DRM_IOCTL_GEM_CLOSE, 0x7ffc70455540) = 0
ioctl(13, DRM_IOCTL_MODE_RMFB, 0x7ffc7045556c) = 0
ioctl(14, _IOC(_IOC_READ|_IOC_WRITE, 0x64, 0x48, 0x28), 0x7ffc704554f0) = 0
ioctl(14, DRM_IOCTL_GEM_CLOSE, 0x7ffc70455540) = 0
ioctl(13, DRM_IOCTL_MODE_RMFB, 0x7ffc7045558c) = 0
ioctl(13, DRM_IOCTL_DROP_MASTER, 0)     = 0
ioctl(12, VT_RELDISP, 0x1)              = 0
ioctl(13, DRM_IOCTL_MODE_RMFB, 0x7ffc7045575c) = 0
```

切入的 xorg IOCTL 过程

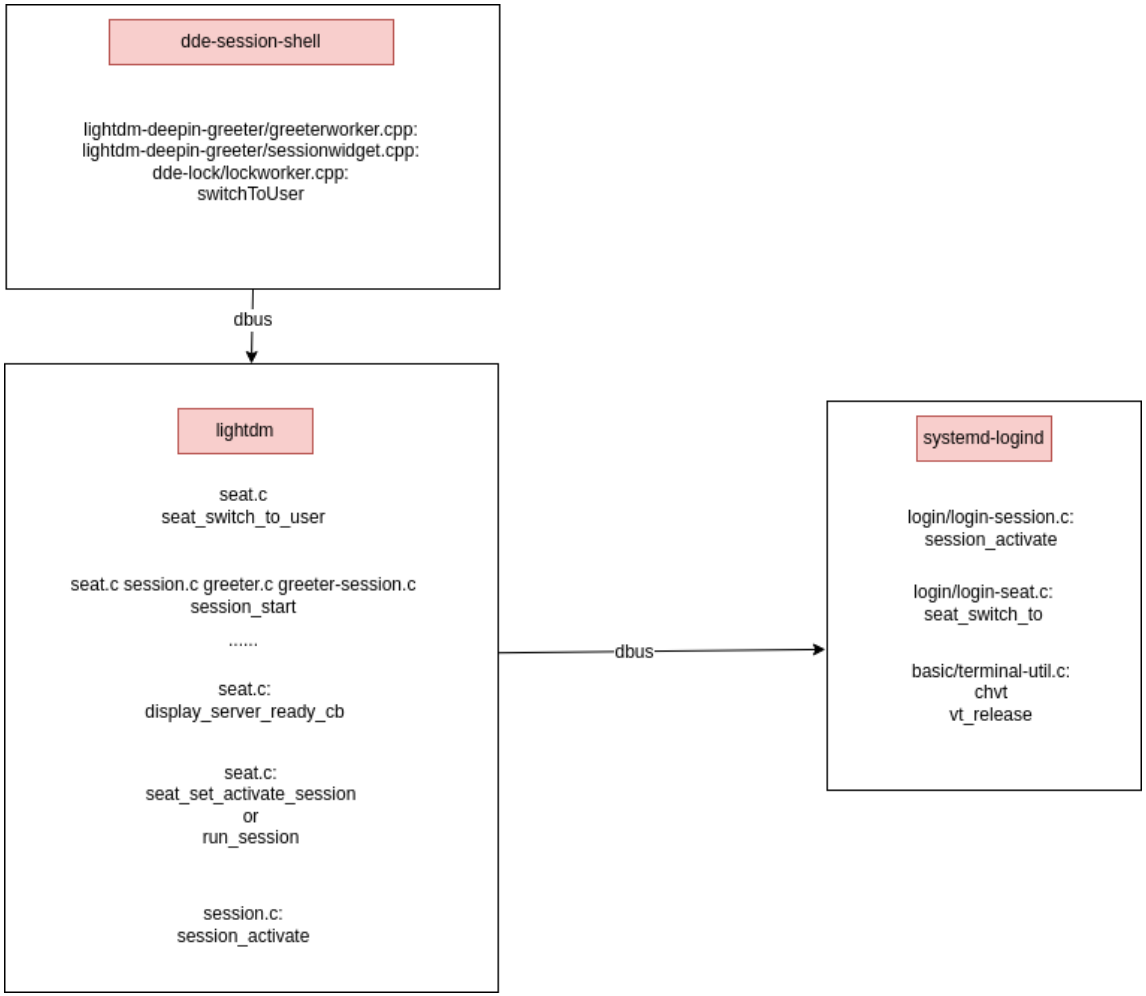
```
root@uos-PC:/home/uos# strace -e ioctl -p 899
strace: Process 899 attached
--- SIGUSR1 {si_signo=SIGUSR1, si_code=SI_KERNEL} ---
ioctl(12, VT_RELDISP, 0x2)              = 0
ioctl(13, DRM_IOCTL_SET_MASTER, 0)      = 0
ioctl(13, DRM_IOCTL_MODE_SETCRTC, 0x7ffc70455570) = 0
ioctl(14, DRM_IOCTL_AMDGPU_GEM_CREATE or DRM_IOCTL_VIA_ALLOCMEM, 0x7ffc70454be0) = 0
ioctl(14, _IOC(_IOC_READ|_IOC_WRITE, 0x64, 0x48, 0x28), 0x7ffc70454bf0) = 0
ioctl(14, DRM_IOCTL_AMDGPU_GEM_METADATA, 0x7ffc70454df0) = 0
ioctl(14, DRM_IOCTL_PRIME_HANDLE_TO_FD, 0x7ffc7045514c) = 0
ioctl(17, DRM_IOCTL_PRIME_FD_TO_HANDLE, 0x7ffc7045518c) = 0
ioctl(14, DRM_IOCTL_PRIME_HANDLE_TO_FD, 0x7ffc7045514c) = 0
ioctl(17, DRM_IOCTL_PRIME_FD_TO_HANDLE, 0x7ffc7045518c) = 0
ioctl(14, DRM_IOCTL_PRIME_HANDLE_TO_FD, 0x7ffc704551cc) = 0
ioctl(17, DRM_IOCTL_PRIME_FD_TO_HANDLE, 0x7ffc7045520c) = 0
ioctl(14, DRM_IOCTL_PRIME_HANDLE_TO_FD, 0x7ffc704550fc) = 0
ioctl(17, DRM_IOCTL_PRIME_FD_TO_HANDLE, 0x7ffc7045513c) = 0
ioctl(14, DRM_IOCTL_PRIME_HANDLE_TO_FD, 0x7ffc704550fc) = 0
ioctl(17, DRM_IOCTL_PRIME_FD_TO_HANDLE, 0x7ffc7045513c) = 0
ioctl(14, DRM_IOCTL_PRIME_HANDLE_TO_FD, 0x7ffc7045512c) = 0
ioctl(14, DRM_IOCTL_PRIME_HANDLE_TO_FD, 0x7ffc7045511c) = 0
ioctl(17, DRM_IOCTL_PRIME_FD_TO_HANDLE, 0x7ffc7045515c) = 0
ioctl(13, DRM_IOCTL_PRIME_FD_TO_HANDLE, 0x7ffc7045539c) = 0
ioctl(13, DRM_IOCTL_AMDGPU_GEM_METADATA, 0x7ffc70455280) = 0
ioctl(13, DRM_IOCTL_MODE_ADDFB, 0x7ffc704553d0) = 0
ioctl(14, DRM_IOCTL_AMDGPU_GEM_CREATE or DRM_IOCTL_VIA_ALLOCMEM, 0x7ffc70454be0) = 0
ioctl(14, _IOC(_IOC_READ|_IOC_WRITE, 0x64, 0x48, 0x28), 0x7ffc70454bf0) = 0
ioctl(14, DRM_IOCTL_AMDGPU_GEM_METADATA, 0x7ffc70454df0) = 0
ioctl(14, DRM_IOCTL_PRIME_HANDLE_TO_FD, 0x7ffc7045514c) = 0
ioctl(17, DRM_IOCTL_PRIME_FD_TO_HANDLE, 0x7ffc7045518c) = 0
ioctl(14, DRM_IOCTL_PRIME_HANDLE_TO_FD, 0x7ffc7045514c) = 0
ioctl(17, DRM_IOCTL_PRIME_FD_TO_HANDLE, 0x7ffc7045518c) = 0
ioctl(14, DRM_IOCTL_PRIME_HANDLE_TO_FD, 0x7ffc704551cc) = 0
ioctl(17, DRM_IOCTL_PRIME_FD_TO_HANDLE, 0x7ffc7045520c) = 0
ioctl(14, DRM_IOCTL_PRIME_HANDLE_TO_FD, 0x7ffc704550fc) = 0
ioctl(17, DRM_IOCTL_PRIME_FD_TO_HANDLE, 0x7ffc7045513c) = 0
ioctl(14, DRM_IOCTL_PRIME_HANDLE_TO_FD, 0x7ffc704550fc) = 0
ioctl(17, DRM_IOCTL_PRIME_FD_TO_HANDLE, 0x7ffc7045513c) = 0
ioctl(14, DRM_IOCTL_PRIME_HANDLE_TO_FD, 0x7ffc7045512c) = 0
ioctl(14, DRM_IOCTL_PRIME_HANDLE_TO_FD, 0x7ffc7045511c) = 0
```

交互的过程

流程

- 上层应用切换用户一是用户按下 Ctrl+Alt+F1 到 F12 快捷键，触发键盘中断；二是通过 dde-session-shell （switchToUser) - lightdm - systemd-logind （用户会话与 tty 关联）
- 切换用户的本质是资源的切换，主要是 systemd-logind 分配 cgroup 的切换和 tty 的切换。
- tty 驱动程序检查输入的按键是否是 VT 切换快捷键。如果是，则调用 vt_ioctl() 函数进行 VT 切换。
- vt_ioctl() 函数将 VT 切换请求转发给 console 驱动程序进行处理。
- console 驱动程序根据请求切换到对应的 VT，并更新对应的 tty 和终端设备状态。
- 一旦 VT 切换完成，console 驱动程序调用 vt_event() 函数向用户空间发送 VT 切换事件。
- Xorg 服务器接收到 VT 切换事件，并调用 xf86VTEnter() 或 xf86VTLeave() 函数来响应事件。
- xf86VTEnter() 或 xf86VTLeave() 函数中，Xorg 服务器会暂停或恢复相关的图形设备、输入设备和输出设备，并执行相应的回调函数。

交互的过程

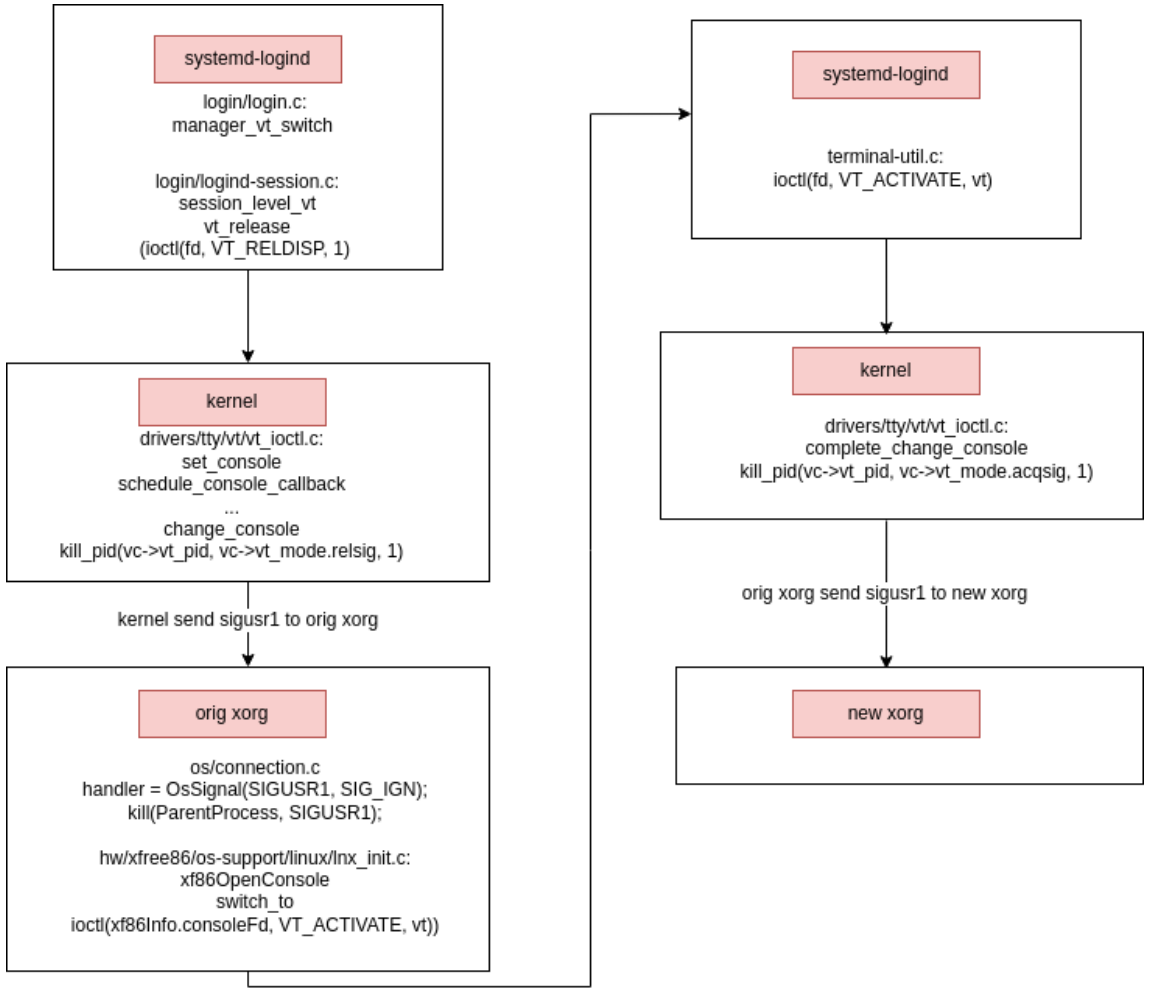

dde-session-shell
lightdm-deepin-greeter/greeterworker.cpp:
lightdm-deepin-greeter/sessionwidget.cpp:
dde-lock/lockworker.cpp:
switchToUser
dbus
lightdm
seat.c
seat_switch_to_user
seat.c session.c greeter.c greeter-session.c
session_start
.....
seat.c:
display_server_ready_cb
seat.c:
seat_set_activate_session
or
run_session
session.c:
session_activate
dbus
systemd-logind
login/login-session.c:
session_activate
login/login-seat.c:
seat_switch_to
basic/terminal-util.c:
chvt
vt_release

关于 sigusr1 信号

即是切出信号也是切入信号

- 点击切换按钮会调用 vt_ioctl 的 VT_ACTIVATE 和 VT_RELDISP 。
- 首先调用 VT_ACTIVATE 的 ioctl 用于激活目标虚拟终端，过程是主要在 vt 驱动中：set_console - schedule_console_callback - console_work - console_callback - change_console vt 切换的过程 1. 内核会向切出的 xorg 发 sigusr1 信号通知它要做处理。
- 然后调用 VT_RELDISP 的 ioctl 用于释放当前终端。complete_change_console - kill_pid(vc->vt_pid, vc->vt_mode.acqsig, 1) vt 切换的过程 2. 切出的 xorg 向切入的 xorg 发 sigusr1 信号。

交互的过程


systemd-logind
login/login.c:
manager_vt_switch
login/logind-session.c:
session_level_vt
vt_release
(ioctl(fd, VT_RELDISP, 1)
kernel
drivers/tty/vt/vt_ioctl.c:
set_console
schedule_console_callback
...
change_console
kill_pid(vc->vt_pid, vc->vt_mode.relsig, 1)
kernel send sigusr1 to orig xorg
orig xorg
os/connection.c
handler = OsSignal(SIGUSR1, SIG_IGN);
kill(ParentProcess, SIGUSR1);
hw/xfree86/os-support/linux/lnx_init.c:
xf86OpenConsole
switch_to
ioctl(xf86Info.consoleFd, VT_ACTIVATE, vt))
systemd-logind
terminal-util.c:
ioctl(fd, VT_ACTIVATE, vt)
kernel
drivers/tty/vt/vt_ioctl.c:
complete_change_console
kill_pid(vc->vt_pid, vc->vt_mode.acqsig, 1)
orig xorg send sigusr1 to new xorg
new xorg

目录

最近的一次修正提交

不支持原子 rmfb 的显卡都会走这个逻辑，虚拟机的 umd 的显示驱动 vmwgfx 要求 fb 要彻底删除。rmfb 的逻辑涉及的范围很广。而且这个逻辑的小修改自 2016 年以来一直没有新的改动，所以是没有大问题的。在公共部分加一些延时的 trick 来修正 Oland 的显卡不是好办法。

```
commit f2d580b9a8149735cbc4b59c4a8df60173658140
Author: Maarten Lankhorst <maarten.lankhorst@linux.intel.com>
Date:   Wed May 4 14:38:26 2016 +0200

    drm/core: Do not preserve framebuffer on rmfb, v4.

    It turns out that preserving framebuffers after the rmfb call breaks
    vmwgfx userspace. This was originally introduced because it was thought
    nobody relied on the behavior, but unfortunately it seems there are
    exceptions.

    drm_framebuffer_remove may fail with -EINTR now, so a straight revert
    is impossible. There is no way to remove the framebuffer from the lists
    and active planes without introducing a race because of the different
    locking requirements. Instead call drm_framebuffer_remove from a
    workqueue, which is unaffected by signals.

    Changes since v1:
    - Add comment.
    Changes since v2:
    - Add fastpath for refcount = 1. (danvet)
    Changes since v3:
    - Rebased.
    - Restore lastclose framebuffer removal too.

    Cc: stable@vger.kernel.org #v4.4+
    Fixes: 13803132818c ("drm/core: Preserve the framebuffer after removing it.")
    Testcase: kms_rmfb_basic
    References: https://lists.freedesktop.org/archives/dri-devel/2016-March/102876.html
    Cc: Thomas Hellstrom <thellstrom@vmware.com>
    Cc: David Herrmann <dh.herrmann@gmail.com>
    Reviewed-by: Daniel Vetter <daniel.vetter@ffwll.ch>
    Tested-by: Thomas Hellstrom <thellstrom@vmware.com> #v3
    Tested-by: Tvrtko Ursulin <tvrtko.ursulin@intel.com>
    Signed-off-by: Daniel Vetter <daniel.vetter@ffwll.ch>
    Link: http://patchwork.freedesktop.org/patch/msgid/6c63ca37-0e7e-ac7f-a6d2-c7822e3d611f@linux.intel.com
```

切换用户过程中，宏观性能下降，闪屏的问题都是不支持 atomic_remove_fb 这个函数的遗留问题。这个问题是显卡的特性，经过一段时间的查询和思考是不能修改的。所以建议 CCB 评审关闭。

占老板还说过一个问题调用 dpms 的时候关 crtc，渲染还没有结束，会导致花屏。这种问题其实也应该归为原子性的问题，如果是支持 drm_atomic_connector_commit_dpms 特性的显卡就不会出现问题。显卡在发展中，历史上出现的一些小的不合理的架构设计也在不断完善，因为硬件架构设计不支持的特性不能在内核当中解决。

目录

Future Work

- 总结调式经验，加快分析速度，提升分析能力，多看显卡驱动的代码。
- 加深 drm 代码和 xorg 主要功能过程的的理解。
- 因为用户态和内核态都有涉及，增加用户态 mesa 的代码熟悉。

Thank you

Thank you for listening!

Q&A

Questions?