Walking Through the Linux-Based Graphics Stack

Paul Kocialkowski
paul@bootlin.com
Paul Kocialkowski

- Embedded Linux engineer at Bootlin
  - Embedded Linux expertise
  - Development, consulting and training
  - Strong open-source focus
- Open-source contributor
  - Co-maintainer of the cedrus VPU driver in V4L2
  - Author of the ov5648 and ov8865 V4L2 camera sensor drivers
  - Author of the logicvc-drm DRM display controller driver
  - Contributor to the sun4i-drm DRM display controller driver
  - Developed the displaying and rendering graphics with Linux training
- Living in Toulouse, south-west of France
Talk Outline

Agenda:
- Big Picture Overview of Graphics
- Early Graphics
- Graphics on a Running System

Focus:
- System-level aspects
- Shed light on little-known aspects
- Code references to popular/reference projects
Big Picture Overview of Graphics
Rationale: where is the graphics data stored, how is it accessed?

Graphics data (pixels) storage:

- **Framebuffers** are the memory areas for pixels
- **Memory location** depends on the situation:
  - System memory or dedicated graphics memory
  - Paged (fragmented) or contiguous memory
- Specific formats, modifiers, compression, lack of meta-data

Graphics memory access:

- Hardware-side memory access: **DMA, IOMMU**
- System-side memory access: **bus mapping, cache**
Rationale: going from memory to photons

- Pixels mixing: **planes/layers** (rotation, scaling, format and more)
- Timings generation: **CRTC**
- Interface layer: **encoder** (controller, PHY)
- Transcoding: **bridge**
- Surface: **panel, monitor**, various technologies
Graphics Hardware: Rendering

Rationale: generating pixels from primitives

- **GPUs** are the all-in-one approach for rendering 3D and 2D
  - Vector drawing units exist but are rarely used
  - Pixels mixers also left out in most cases

- Specific hardware features for the task:
  - Programmable pipeline with shaders: vertex, geometry, fragment
  - Dedicated vector/SIMD instruction set(s)
  - Texture mapping units, cache
  - Tiled framebuffer representations

- Requires a dedicated compiler for shaders
- Configured via a command stream in memory
- **High complexity** and power usage
Graphics APIs: Linux kernel

Rationale: providing low-level applications access to hardware features

Linux kernel subsystms and uAPIs:

- **Fbdev**: covers display, legacy: missing many many features
- **DRM**: modern subsystem for graphics
  - **KMS**: covers display, up-to-date
  - **KMS atomic**: extension for atomic state changes
  - **GEM**: memory management, zero-copy (PRIME), fences (Syncobj)
  - **Render**: covers rendering, driver-specific

Low-level libraries:

- **libdrm**: wrapper for DRM syscalls
Graphics APIs: Displaying in Userspace

Rationale: allowing applications to display their contents

Low-level display server APIs:
- **X11**: legacy protocol with various issues, various extensions
- **Wayland**: modern protocol, various extensions

Associated low-level libraries:
- **Xlib, XCB**: X11 protocol and extensions wrapper
- **libwayland-{display,server}**: Wayland protocols marshalling

Higher-level graphics libraries/toolkits:
- **Qt, GTK, EFL**: widget-based toolkits
- **SDL**: drawing-oriented toolkit
Graphics APIs: 2D Rendering in Userspace

Rationale: providing high-level access to 2D rendering/operations

Base drawing libraries:
- **Cairo**: vector drawing
- **Skia**: vector drawing

Pixel-level libraries:
- **Pixman**: pixel-level operations
- **FFmpeg swscale**: format, scaling
- **G’MIC**: processing

Font rendering:
- **FreeType**: Font rendering
- **Harfbuzz**: Font rendering

UI rendering:
- **Graphics toolkits**
- **ImGui, nuklear**: Immediate-mode
Graphics APIs: 3D Rendering in Userspace

Rationale: providing high-level access to 3D rendering

Standard APIs/formats:

► **OpenGL (ES)**: Stateful high-level rendering
  - **GLSL**: OpenGL shading language
► **EGL**: Window system integration
  - **GBM**: EGL-DRM KMS glue
► **Vulkan**: Stateless lower-level, low-overhead rendering
  - **SPIR-V**: Intermediate representation for shaders

Implementations:

► **Mesa 3D**: reference free software, using DRM
► **Proprietary**: hardware-specific, various issues
Early Graphics
 framebuffer 

### Why do we need early graphics?

- Show a **sign of life** before init
- **Kernel and init logs** for debugging
- **LUKS** password entry in initramfs

#### fbs con implements a VT/TTY bridge with graphics:

- stdin is grabbed via the **input** subsystem
- stdout is rendered and displayed via **fbdev**
- Can be used for kernel logs: `console=tty1`
- Enabled with `CONFIG_FRAMEBUFFER_CONSOLE`
- Can also display a logo: `CONFIG_LOGO`

#### Framebuffer device provided by:

- **Boot software**: VESA, EFI, device-tree (simple-framebuffer)
- **Dedicated driver**: hardware-specific
- **DRM fb helper**: compatibility layer
Framebuffer Console: Code Highlights

Linux kernel:

▶ drivers/video/fbdev/core/fbcon.c:
  • struct consw fb_con
  • fbcon_set_bitops()
  • fbcon_prepare_logo()
  • do_fbcon_takeover()
  • fbcon_redraw()
  • fbcon_putchar()

▶ drivers/video/fbdev/core/bitblit.c:
  • bit_putcs()

▶ drivers/tty/vt/vt.c:
  • struct tty_operations con_ops
  • do_update_region()
  • do_take_over_console
Linux kernel:

- drivers/gpu/drm/drm_fb_helper.c:
  - struct fb_ops drm_fbdev_fb_ops
  - drm_fbdev_generic_setup
  - drm_fb_helper_generic_probe()
  - __drm_fb_helper_initial_config_and_unlock()
  - drm_fb_helper_single_fb_probe()
  - drm_fb_helper_pan_display
Users expect a **waiting screen** rather than logs

Not a kernel-level feature:

- Dedicated applications for the task
- Running after init, as root
- Typically in the initramfs

Using either **fbdev** or **DRM KMS** directly

Often show systemd boot progress

Various implementations exist:

- **Plymouth**: most advanced, progress, animations, supports DRM KMS and fbdev
- **Psplash**: from Yocto Project, progress, uses fbdev
- **Fbsplash**: themable, progress, uses fbdev
Walking Through the Linux-Based Graphics Stack

Running System
VT Mode

- fbcon **takes over VT** at boot
  - As soon as framebuffer is available

- **VT sharing** between fbcon/userspace:
  - Access to the display must be **exclusive**
  - Privileged operations
  - Fbcon needs to be detached
  - Requires active cooperation

- **VT modes** reflect the current VT state:
  - **KD_TEXT**: fbcon is attached to the VT
  - **KD_GRAPHICS**: ready for userspace graphics use
  - **Switched** upon request with **KDSETMODE ioctl**, using the TTY fd (controlling terminal or not)

- Similar mechanism exists for input
VT Switching

- **Multiple VTs/TTYs are spawned at boot:**
  - A single VT is **active** at a time (tty1 at boot)
  - Switching triggered with: Ctrl + Alt + F[n]
  - No userspace intervention for fbcon

- **Coordination required** when userspace uses graphics:
  - Kernel needs to notify application of VT switching
  - Signal-based release/acquire handlers registered with `VT_SETMODE` ioctl
  - Graphics **resources** need to be **released/re-acquired**
  - Kernel waits for acknowledge (can hang)

- **Implications for complex systems:**
  - Multiple graphics sessions can run in **parallel**!
  - Typically the case with the login manager
  - Other limitations might restrict this ability
VT Mode and Switching: Code Highlights

Linux kernel:

- drivers/tty/vt/vt_ioctl.c:
  - vt_k_ioctl()
  - vt_kdsetmode()
  - change_console()
  - complete_change_console()

- drivers/tty/vt/vt.c:
  - set_console()
  - console_callback()

Weston:

- libweston/weston-launch.c:
  - setup_tty()
  - handle_signal()

- libweston/launcher-direct.c:
  - setup_tty()
  - vt_handler()
Configuring graphics (and VT) are **privileged** operations

- Corresponds to DRM KMS **master** privilege:
  
  ```
  DRM_IOCTL_SET_MASTER/DRM_IOCTL_DROP_MASTER
  ```
  on DRM KMS fd
- Typically restricted to the **root** user
- Used to require running the display server as root
- (Very) problematic **security implications**

**Systemd** introduced **systemd-logind**:

- Runs as root and opens DRM KMS and VT TTY fds
- Provides a **D-Bus service** for applications (display servers):
  
  ```
  org.freedesktop.login1
  ```
- DRM KMS fd is passed over UNIX socket
- VT operations are made available as methods
- Applications can run as **regular users**!
Systemd:

- src/login/logind-session-device.c:
  - session_device_open()
- src/login/logind-session.c:
  - manager_vt_switch()
- src/login/logind-session.c:
  - session_open_vt()/session_prepare_vt()
  - session_restore_vt()/session_leave_vt()
- src/login/logind-session-dbus.c:
  - method_take_device()/method_release_device()

Weston:

- libweston/launcher-logind.c:
  - launcher_logind_take_device()/launcher_logind_release_device
  - launcher_logind_activate_vt()
Login manager

Rationale: users needs to login in multi-user/general-purpose setups

- Login managers provide a **graphical equivalent** to getty
- Run their **own display server** under their own user
- Started at the end of the boot process (on first VT)
- Allow selecting between different **sessions**:
  - **X.org**: `/usr/share/xsessions/` desktop files
  - **Wayland**: `/usr/share/wayland-sessions/` desktop files
- Starts display server in **user context**:
  - Usually authenticated via PAM
  - Usually in a dedicated VT
Rationale: applications want to submit pixels to the display server

- **Actually transfer of pixels is deprecated**:
  - Zero-copy buffer sharing with display server is used instead
  - Buffers are identified by API-specific identifiers (e.g. fds)

- **Buffer sharing** has two major instances:
  - **SHM**: Typically drawn by the CPU
  - **EGL**: Typically drawn by the GPU

- **Allocation** is often managed by APIs
  - Zero-copy import may be possible:
    - e.g. EGL_EXT_image_dma_buf_import
  - Might cause hardware access issues (but usually works)

- **Coordination** with the display server for presentation:
  - Damage region provided by application (e.g. wl_surface_damage)
  - Sync point when ready for presentation (e.g. wl_surface_commit)
Weston:

▶ clients/simple-damage.c:
  • create_window()
  • redraw()

▶ clients/simple-shm.c:
  • create_display()
  • redraw()

▶ clients/simple-egl.c:
  • create_surface()
  • init_egl()
  • redraw()

▶ clients/simple-dmabuf-egl.c:
  • create_dmabuf_buffer()
  • redraw()
Display Server: Compositing

Rationale: display servers need to gather applications buffers

▶ A **unique buffer** is submitted to the display hardware:
  - Contains the contents of all visible applications
  - Stacked according to window manager policy
  - Needs to be redrawn upon (visible) application indication

▶ Compositing is a **very demanding** task:
  - Full redraw must be avoided at all costs!
  - Can run up to display frame rate (e.g. 60 Hz)
  - Damage is tracked and used for clipping regions

▶ **Hardware acceleration** is leveraged (if not necessary):
  - Typically rendered with the GPU, buffers as textures
  - Hardware planes can be leveraged, but usually not (primary only)
  - Cursor is typically composited by the hardware with a dedicated plane
Weston:

- **libweston/pixman-renderer.c:**
  - pixman_renderer_repaint_output()
  - draw_view()
  - repaint_region()
  - composite_clipped()

- **libweston/renderer-gl/gl-renderer.c:**
  - gl_renderer_repaint_output()
  - draw_view()
  - repaint_region()
  - texture_region()
Display Server: Page Flipping

Rationale: achieving glitch-free display contents update

➤ **Tearing** is a well-known issue with display sync:
  - Display hardware scans out buffer at given address
  - Scanout happens continuously at refresh rate
  - Display server needs to update the presented contents
  - Concurrent read (hardware) and write (display server) causes a glitch

➤ Tearing is resolved with a **double-buffering** approach:
  - Front buffer is shown, back buffer is being prepared
  - Roles are exchanged at next vertical sync (vblank) point
  - More buffers can be used but increase latency

➤ **DRM KMS** ensures **page flipping** happens at vblank:
  - Scheduled using `DRM_IOCTL_MODE_PAGE_FLIP` (with target)
  - Scheduled with atomic commit using `DRM_IOCTL_MODE_ATOMIC`
  - Can notify userspace (blocking or async event) when done
Weston:

- libweston/backend-drm/kms.c:
  - drm_output_apply_state_atomic()
  - drm_pending_state_apply_atomic()
  - drm_output_apply_state_legacy()
  - drm_output_set_cursor()
  - atomic_flip_handler()/page_flip_handler()
Questions? Suggestions? Comments?

Paul Kocialkowski
paul@bootlin.com

Slides under CC-BY-SA 3.0
https://bootlin.com/pub/conferences/