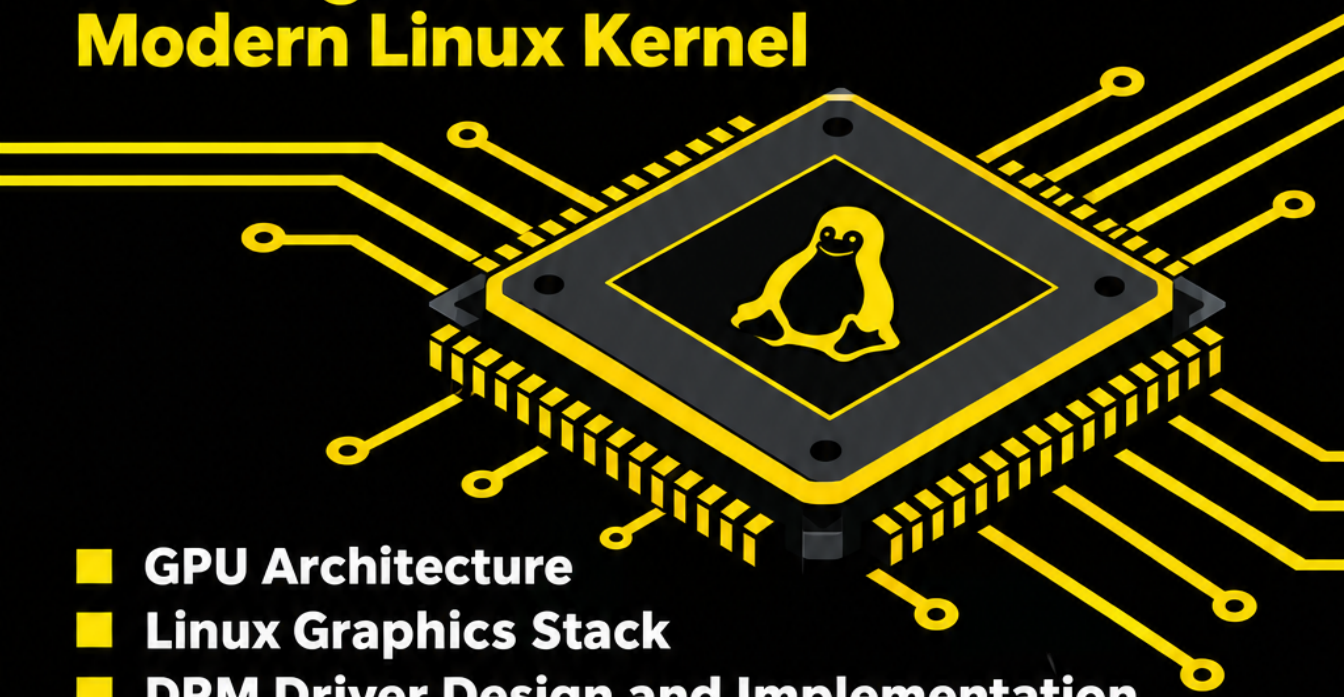
# Linux GPU Drivers from Scratch

## Building Real DRM Drivers for the Modern Linux Kernel



- GPU Architecture
- Linux Graphics Stack
- DRM Driver Design and Implementation
- Memory Management
- Command Submission
- Synchronization
- Display Support
- Error Recovery

NICHOLAS PEMBROKE

# Linux GPU Drivers from Scratch

## Building Real DRM Drivers for the Modern Linux Kernel

Steve Publications

This book is available at https://leanpub.com/linuxgpudriversfromscratch

This version was published on 2026-09-17


This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

# Contents

# Building Real DRM Drivers for the Modern Linux Kernel

This book takes you from the fundamentals of GPU architecture and the Linux graphics stack all the way to designing, implementing, and debugging a complete Direct Rendering Manager driver for the Linux kernel. If you can write C and understand the basics of kernel modules, this book will teach you how to bridge the gap between graphics hardware and userspace applications through production-quality Linux kernel code. Every chapter builds on the previous one, progressing from a minimal loadable module to a fully functional DRM driver with memory management, command submission, synchronization, display support, and error recovery. Real code, real build instructions, no shortcuts.

# Chapter 1: Before We Begin

You have strong C skills and have written at least a Linux kernel module. You know what printk is. You understand device drivers in principle. But you have never looked at a GPU driver and wondered how it works. This chapter is for you. It sets up the development environment, explains what we will build across this book, and clarifies what is and is not inside the scope of a Linux GPU driver.

## Understanding What a GPU Driver Is

A graphics processing unit driver in Linux is not a single thing. It is an integration point where several subsystems meet: the PCI or platform bus layer, the Direct Rendering Manager core, memory management frameworks, interrupt handling, synchronization primitives, and the userspace interface layer. A GPU driver sits at the center and ties them all together.

At the most fundamental level, a GPU driver does these things:

- Discovers the GPU hardware on the bus and claims it
- Maps the hardware registers into kernel virtual memory so the CPU can read and write them
- Allocates and manages GPU-visible memory, which may include VRAM, system RAM, or both
- Receives command streams from userspace and validates, submits, and tracks their execution on the GPU
- Handles GPU interrupts to know when work completes or when something goes wrong
- Configures the display pipeline: resolutions, refresh rates, pixel formats, connectors, and scanout buffers
- Manages power states: clocks, voltage, suspend/resume, and runtime power gating
- Coordinates with other kernel components that need GPU resources, such as video codecs or display controllers

And critically, a GPU driver exposes a well-defined interface to userspace. Applications never talk directly to the GPU. They call into libraries like Mesa, which translate OpenGL, Vulkan, or other API calls into command buffers and metadata, and then issue ioctl system calls into the kernel driver. The kernel driver is responsible for making sure those commands are valid, mapping them to hardware registers and memory, and getting them executed safely.

In the Linux kernel, all modern GPU drivers implement the Direct Rendering Manager interface. DRM is the kernel subsystem that provides the common infrastructure. It handles device file management, ioctl dispatch, reference counting, and userspace communication patterns. It provides standardized abstractions for display pipelines and memory management. And it defines the contract between kernel and userspace that allows graphics stacks like Wayland and X.org to work across different hardware vendors.

This book is about writing DRM drivers. When we say GPU driver in this book, we mean a DRM driver.

## Prerequisites and Assumptions

Before proceeding, let us be clear about what this book assumes you already know. If you are not comfortable with any of these topics, you may want to review them before starting.

You should be proficient in C. We will write kernel code that uses pointers extensively, manipulates bit fields, defines structures with precise memory layouts, and handles errors through return codes rather than exceptions. You should be comfortable reading dense C code and writing it yourself.

You should have written at least one Linux kernel module. You should understand what module_init and module_exit do. You should know how to use Kbuild, the kernel build system, to compile a module. You should understand the difference between build-time and run-time symbols, and why kernel code cannot use the standard C library.

You should have basic familiarity with the Linux kernel memory model. You should know the difference between vmalloc and physically contiguous memory. You should understand what a page is and what page faults are. You should be aware of the distinction between kernel space and user space.

You should know the basics of how Linux device drivers are structured. You should understand the device model: buses, devices, drivers, and how they

match. You should know what a probe function does. You should understand why drivers use platform devices, PCI devices, or device tree nodes as their hardware anchor points.

You should be comfortable with basic Unix tooling: compiling, reading kernel logs with dmesg, loading and unloading modules with insmod and rmmod, and using text editors and version control in a terminal environment.

If you have all of that, you are ready. You do not need prior knowledge of graphics APIs, GPU architecture, or display technology. We will build up all of that from first principles.

## Choosing a Kernel Version

The Linux kernel changes every release. Functions are added, deprecated, and removed. Structures grow new fields. Coding conventions evolve. For a book that aims to be practically useful, we must pick a specific kernel version as our target.

This book uses Linux kernel 6.8 as the primary reference. Kernel 6.8 is a long-term support release, meaning it receives maintenance updates for at least two and a half years. It is current enough to reflect modern DRM infrastructure while stable enough that the APIs will not change dramatically. The code in this book targets the kernel 6.8 release series. The DRM subsystem in 6.8 includes mature atomic modesetting support, the modern devm-based device resource management APIs, dma_fence synchronization, and updated GEM and TTM memory managers.

If you are running a different kernel version, you may encounter differences. Kernel 6.6 LTS is close enough that most code will work with minimal changes. Kernel 6.10 and beyond may have additional features but should remain backward compatible for the interfaces we use. If you plan to contribute your driver to upstream kernel, you should target the latest development kernel at the time of submission and be prepared to adapt to new requirements.

The most important thing is consistency. Pick a kernel version and stick with it while reading this book. When you see a function name or structure definition in our code examples, know that it corresponds to kernel 6.8. If you find something different in your kernel version, check the commit history or documentation for that kernel version to understand what changed.

## Setting Up the Build Environment

Let us walk through setting up your development environment step by step. You will need a Linux system where you can build and run kernel modules. You can use your host system if it runs Linux, or a virtual machine. For safety, especially when experimenting with GPU drivers that can hang your system, a virtual machine or a secondary machine is recommended.

First, install the required build tools. On a Debian or Ubuntu based system, run:

```
sudo apt update
sudo apt install build-essential libncurses-dev bison flex libssl-dev
↪  libelf-dev dwarves git ccache
```

On a Fedora or RHEL based system, run:

```
sudo dnf group install "Development Tools"
sudo dnf install ncurses-devel bison flex openssl-devel elfutils-libelf-devel
↪  dwarves git ccache
```

On an Arch based system, run:

```
sudo pacman -S base-devel ncurses bison flex openssl libelf pahole git ccache
```

These packages provide the compilers, build tools, and development libraries needed to build the Linux kernel. ccache is optional but highly recommended to speed up incremental builds.

Next, obtain the kernel source. You can clone the official kernel repository:

```
git clone --depth 1
↪  https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git
cd linux
```

Or download a specific version from kernel.org. For kernel 6.8, visit https://www.kernel.org/pub/linux/kernel/v6.x/linux-6.8.tar.xz, download it, and extract it:

```
tar xf linux-6.8.tar.xz
cd linux-6.8
```

Once you have the source, configure the kernel. If you already have a running kernel on your system, you can use its configuration as a starting point:

```
zcat /proc/config.gz > .config
```

Or on systems that install the kernel config elsewhere:

```
cp /boot/config-$(uname -r) .config
```

Then run the configuration tool to review and adjust settings:

```
make menuconfig
```

In the configuration menu, ensure the following options are enabled:

Enable DRM support under Device Drivers -> Graphics support -> DRM support. Enable the GPU subsystem options you will need. For PCI-based GPUs, enable PCI support and PCI hotplug. For GEM, enable DRM memory management under the DRM menu. For debugging, consider enabling Dynamic Debug and ftrace under Kernel hacking.

After configuring, save the configuration and exit.

You do not need to build the entire kernel for driver development. You can build just the DRM subsystem and your driver as loadable modules. This is much faster than building the whole kernel.

To build the base kernel without modules, run:

```
make -j$(nproc)
```

To build only the modules you need, run:

```
make modules -j$(nproc)
```

This will take a while on the first build. On subsequent builds with incremental changes, ccache will speed things up significantly.

## Testing in a Virtual Environment

Writing GPU drivers carries risk. A buggy driver can hang the GPU, lock up the display, or in worst cases freeze the entire system. For development and testing, it is wise to use a virtual environment where you can recover easily.

QEMU is the standard choice for virtual GPU development. QEMU includes several virtual GPU devices that present themselves as PCI devices and support DRM drivers. The most useful ones are:

virtio-gpu: A paravirtualized GPU that is widely used and well supported. It supports DRM, KMS, and GEM. This is a good choice for testing display and memory management code.

bochs: A simpler VGA-compatible device that the kernel's bochs driver supports. Useful for minimal testing.

virtio-gpu is preferred for most development because it closely matches the behavior of real GPUs while remaining safe to use in a virtual machine.

To test a driver with QEMU and virtio-gpu, boot a Linux kernel with the virtio-gpu device enabled. Here is a minimal QEMU command:

```
qemu-system-x86_64 -m 2048 \
  -kernel /path/to/vmlinuz \
  -append "root=/dev/vda1 console=tty0" \
  -drive file=/path/to/rootfs.img,format=raw,if=virtio \
  -device virtio-gpu-pci \
  -display none
```

This boots a virtual machine with 2048 MB of RAM, your kernel, a virtio disk, and a virtio-gpu device. The -display none option prevents QEMU from opening a graphics window, which is useful for headless testing. You can connect via a serial console if needed.

If you are developing a driver for hardware that does not exist yet, or for hardware you do not have access to, QEMU can still help. You can emulate a custom PCI device in QEMU that mimics the behavior of your target GPU. This allows you to develop the driver infrastructure in isolation from the actual silicon. We will discuss this approach in more detail later.

## The Tools You Will Use

Beyond the build tools, you will need several utilities for development and debugging:

dmesg reads the kernel ring buffer. It is your first line of defense for understanding what the kernel thinks is happening.

lsmod lists loaded modules. Use it to verify your driver loaded.

modinfo shows module information including parameters, dependencies, and license.

insmod and rmmod load and unload kernel modules. For more sophisticated module management, use modprobe.

lspci lists PCI devices. You will use this to verify that your GPU is detected and to check which driver is bound to it.

cat /sys/class/drm shows DRM devices. When your driver registers successfully, you will see device nodes like card0 and renderD128 in /dev.

strace traces system calls. Use it to understand how userspace programs interact with your driver through ioctls.

ftrace and tracepoints provide kernel-level tracing. We will discuss these in depth in the debugging chapter.

gdb with kernel debugging symbols allows you to debug kernel modules. This requires setting up a special debugging environment, which we will cover later.

## How to Read This Book

This book is organized as a progressive journey. Each chapter builds on the previous ones. The chapters fall into broad phases:

The first phase covers foundations. We explain GPU architecture, the Linux graphics stack, DRM, KMS, GEM, TTM, and PCI. These chapters provide the vocabulary and concepts you need to understand the code. You should read these chapters carefully and refer back to them when the terminology feels unfamiliar.

The second phase is where we start writing code. We build a minimal DRM driver from scratch. This driver will be loadable as a kernel module, will register with the DRM subsystem, and will create device nodes. It will not do anything useful yet, but it will compile, load, and unload cleanly. From there, we progressively add capabilities: memory management, display support, command submission, synchronization, and error recovery.

The third phase covers advanced topics: concurrency, security, performance optimization, and upstream development. These chapters are essential for turning a working driver into a production-quality driver that can be maintained and contributed to the mainline kernel.

Throughout the book, we distinguish between two kinds of code:

DRM core infrastructure code: This is the code in drivers/gpu/drm/ that provides shared functionality. You will use DRM APIs but rarely write DRM core code yourself, unless you are improving the DRM subsystem.

Hardware-specific driver code: This is the code in your driver that knows about your particular GPU: register offsets, memory layout, command formats, clock configurations, power states, and all the details that make your driver unique to your hardware.

The book focuses on hardware-specific driver code, since that is what you will write. But we explain enough about DRM core to help you understand the infrastructure you are building on.

Every code example in this book is complete and compilable. When we show a file, it is the full file contents, not an abbreviated snippet. When we show a Makefile or Kconfig entry, it is the exact text you should use. If a piece of code requires hardware-specific information that we cannot provide because it depends on your particular GPU specification, we will mark it clearly and explain what you need to fill in.

Now let us begin. The next chapter introduces GPU architecture from first principles, so you understand what hardware your driver will be talking to.

# Chapter 2: GPU Architecture Fundamentals

You cannot write a good GPU driver if you do not understand what a GPU actually does. GPUs look like accelerators from the outside, but the reality is more nuanced. A GPU is a specialized processor optimized for massively parallel workloads. It has its own memory hierarchy, execution units, command processors, and display engines. It communicates with the CPU through memory-mapped registers, DMA transfers, and interrupts. Understanding these components is not academic: every one of them maps to code you will write.

## From CPU to GPU: Why Graphics Need Special Hardware

Central processing units are general-purpose machines. They execute instructions sequentially with deep pipelines, complex branch prediction, and large caches. They excel at control flow: making decisions, handling interrupts, and running arbitrary algorithms.

Graphics rendering and compute workloads are different. They operate on huge arrays of data with regular, predictable patterns. For example, shading a triangle involves applying the same mathematical operations to thousands of pixels independently. The CPU could do this, but it would spend most of its time managing control flow and loading data rather than computing results.

GPUs trade flexibility for throughput. They have many more arithmetic units than a CPU, fewer control units, simpler caches, and a memory hierarchy optimized for bulk data transfer. Where a CPU might have eight to sixty-four cores, a GPU has hundreds to thousands of smaller cores organized into groups that execute the same instruction on different data.

For the driver writer, the key insight is this: GPUs are designed to run command streams submitted by the CPU. The CPU prepares work, writes it

into memory, and tells the GPU where to find it. The GPU then executes asynchronously while the CPU continues its own work. The driver is responsible for preparing those command streams, submitting them, tracking their completion, and recovering when they fail.

## The GPU Command Processor

At the heart of every GPU is a command processor. This is a dedicated hardware block that interprets and dispatches commands. It is the GPU equivalent of a CPU's instruction fetch and decode unit, but designed for complex graphics and compute operations rather than general-purpose instructions.

The command processor reads command buffers from memory. A command buffer is a sequence of binary commands that describe work to be done. Each command specifies an operation, such as clearing a region of memory, drawing a triangle, dispatching a compute shader, or configuring a pipeline state. Commands include operands that reference resources: vertex buffers, texture maps, shaders, render targets.

From the driver's perspective, the command processor is the primary mechanism for getting work onto the GPU. The driver builds command buffers in system memory or VRAM, writes them to a well-defined location, and then rings a doorbell to tell the command processor that new commands are available.

Different GPU vendors use different command formats. NVIDIA, AMD, Intel, and ARM each have their own instruction set. The driver must know the exact format: which bits specify the opcode, which bits specify operands, how commands are aligned and padded. This is highly hardware-specific knowledge that comes from the GPU's specification documents.

Some GPUs have multiple command processors or engines. For example, one engine might handle 3D rendering while another handles video encoding. Each engine has its own command buffer format and doorbell register. The driver must manage submission to each engine independently while maintaining correct ordering between them.

## Ring Buffers and Submission Queues

Command buffers are typically organized as ring buffers, also called circular buffers. A ring buffer is a fixed-size memory region treated as circular. The driver writes commands at a write pointer and the GPU reads from a read pointer. When the read pointer catches up to the write pointer, the GPU has processed all submitted commands. When the write pointer catches up to the read pointer, the buffer is full and the driver must wait before submitting more.

Ring buffers are used because they are efficient. They avoid the need to constantly allocate and free memory for individual command buffers. The driver and GPU can operate on the same memory region without coordination, as long as they respect the read and write pointers.

A typical ring buffer has the following structure:

A fixed block of memory, usually page-aligned, allocated by the driver during initialization.

A head pointer that indicates where the GPU is currently reading. This is typically maintained by the GPU hardware and readable through a status register.

A tail pointer that indicates where the driver has written up to. This is written by the driver, often through a doorbell register, to notify the GPU of new work.

Free space tracking to prevent the tail from wrapping past the head. The driver must check that there is enough room for a command before writing it.

Here is how the submission flow works in practice:

The driver checks if there is enough free space in the ring buffer for the next command. If not, the driver waits until the GPU advances its read pointer.

The driver writes the command into the ring buffer at the current tail position and updates the tail pointer.

The driver writes the new tail value to a doorbell register. This is a memory-mapped register that causes the GPU's command processor to wake up and start processing if it was idle.

The GPU reads commands starting from its current read pointer position. As it processes each command, it advances the read pointer.

The GPU signals completion through an interrupt or by updating a status register that the driver can poll.

The ring buffer is simple but fundamental. Every GPU driver uses some variant of this pattern for command submission.

## GPU Memory Hierarchy

GPU memory is organized in a hierarchy that balances bandwidth, latency, and capacity. The exact layout varies by GPU, but the general structure is consistent across architectures.

At the top of the hierarchy are on-chip caches. These are small, fast memories that store recently accessed data. Texture caches store texels for rendering. Instruction caches store shader code. Shared memory or local memory allows threads within a warp or wavefront to communicate efficiently. These caches are fully managed by hardware and invisible to the driver.

Below the caches is the GPU's main memory. This can be:

Video RAM, or VRAM: Dedicated memory chips soldered onto the graphics card, connected via a high-bandwidth memory bus. VRAM offers the highest bandwidth but is limited in capacity and only directly accessible by the GPU. Modern VRAM technologies include GDDR6, GDDR6X, and HBM.

System RAM: On integrated GPUs and some unified memory architectures, the GPU shares system memory with the CPU. The GPU accesses system RAM through the system interconnect, which has lower bandwidth than dedicated VRAM but offers virtually unlimited capacity.

Unified memory: In some architectures, the CPU and GPU share a single physical memory space with a coherent cache hierarchy. The driver does not need to manage data migration between CPU and GPU memory explicitly.

For the driver, memory hierarchy matters because different memory regions have different characteristics:

VRAM is fast but limited. The driver should keep actively used resources in VRAM.

System RAM is slower but abundant. The driver can use it for large buffers, cached resources, or less frequently accessed data.

Cache coherence must be managed. If the CPU writes to a buffer that the GPU will read, the CPU must ensure its writes are visible before the GPU accesses them. This involves cache flushes and memory barriers.

The driver implements a memory manager that decides where to place each buffer based on its usage pattern, size, and performance requirements. We will cover GPU memory management in detail later.

## Execution Units

Execution units are where GPU computations happen. They are organized hierarchically:

Individual cores or stream processors perform arithmetic and logic operations.

Groups of cores form clusters. On NVIDIA GPUs, these are called Streaming Multiprocessors. On AMD GPUs, Compute Units. On Intel GPUs, Execution Engines or Sub-slices.

Clusters are grouped into larger units. These might be slices, dies, or compute complexes depending on the architecture.

Each cluster has its own local resources: shared memory, register files, and caches. Threads running on the same cluster can share data through shared memory, which is much faster than accessing global memory.

Threads are the fundamental unit of GPU execution. GPUs launch workloads as groups of threads that execute the same program. These thread groups are called warps on NVIDIA GPUs, wavefronts on AMD GPUs, and subslices on Intel GPUs. All threads in a group execute the same instruction at the same time, but with different data. If threads diverge due to a branch, the GPU serializes the execution paths.

The driver's role in execution is to prepare work for execution units. This involves:

Compiling or selecting shaders. Modern drivers use JIT compilation to generate machine code for shaders written in high-level languages.

Packaging work as dispatch commands. A dispatch specifies the kernel to run, the number of threads, and the resources needed.

Managing execution contexts. Each application or process may have its own execution context with its own shader programs, resources, and state.

Scheduling work across execution units to maximize utilization.

Most of this is handled by the command processor and GPU hardware once the driver submits properly formatted commands. The driver's main responsibility is to ensure that commands are well-formed and that resources are correctly configured.

## Display Engines

Many GPUs include display engine hardware. This is the part of the GPU that takes pixel data and sends it to a display device. Display engines are relatively independent of the rendering pipeline and have their own command processors and memory paths.

A typical display engine includes:

Scanout engines that read pixel data from a framebuffer and output it at a specified timing.

Timers and counters that generate horizontal and vertical blanking signals to synchronize with the display.

Pixel format converters that translate between internal GPU formats and display formats.

Scaling units that resize images to match the display resolution.

Overlay units that composite multiple image layers.

The driver must configure display engines to produce the desired output. This involves setting the resolution, refresh rate, pixel format, and input buffer. In the Linux DRM subsystem, this is handled by KMS. We will cover display engines and KMS in detail in later chapters.

For headless or compute-only GPUs, display engines may not be present. The driver still registers as a DRM device, but only provides rendering and compute capabilities without display support.

## The Role of the Driver in the Control Chain

Let us trace the path from an application request to actual GPU execution. This illustrates where the driver fits in the overall system.

An application calls a graphics API function, such as glDrawArrays in OpenGL or vkCmdDraw in Vulkan.

The userspace graphics driver library, typically part of Mesa, translates the API call into GPU-specific commands. These commands are written into a command buffer in memory.

The userspace library calls a DRM ioctl to submit the command buffer to the kernel driver.

The kernel driver validates the command buffer: checking that resources are properly bound, commands are well-formed, and memory accesses are within bounds.

The kernel driver copies or maps the command buffer into a location the GPU can access.

The kernel driver writes submission metadata to GPU registers and rings the doorbell.

The GPU command processor reads the commands and dispatches work to execution units.

The GPU execution units perform the actual rendering or computation.

When complete, the GPU writes a completion marker to memory or triggers an interrupt.

The kernel driver receives the interrupt or detects the completion marker, and updates synchronization state.

If the rendering was for display, the display engine scans out the resulting framebuffer to the monitor.

The kernel driver returns to userspace and the application can submit more work or wait for completion.

The driver is involved at every step where the CPU and GPU interact. It is the translator between the abstract world of graphics APIs and the concrete world of hardware registers and memory layouts.

## Hardware Register Interfaces

GPUs expose control and status registers through memory-mapped I/O. These registers are the primary interface between the CPU and GPU. The driver reads and writes these registers to:

Initialize the GPU on startup. This involves configuring clocks, resetting units, loading firmware, and setting up memory mappings.

Configure execution. Pipeline state, shader programs, resource bindings are all set through registers.

Submit work. Doorbell registers trigger command processing.

Query status. The driver reads registers to check GPU busy status, query completed work, and detect errors.

Handle interrupts. Interrupt enable and status registers control how the GPU signals events to the CPU.

Manage power. Clock and voltage registers control power states.

Each register has a specific address within a BAR. The BAR range that contains registers is typically small compared to the total GPU address space. The driver maps this range at initialization time and accesses registers through the mapped virtual address using readl and writel functions.

Register layouts are hardware-specific. The driver must know the exact offset of each register and the meaning of each bit field. This information comes from the GPU specification. A typical register definition in driver code looks like:

```
#define GPU_STATUS_OFFSET        0x0000
#define GPU_STATUS_BUSY          BIT(0)
#define GPU_STATUS_IDLE          BIT(1)
#define GPU_STATUS_ERROR         BIT(2)

#define GPU_DOORBELL_OFFSET      0x1000

#define GPU_COMMAND_PTR_LOW_OFFSET  0x2000
#define GPU_COMMAND_PTR_HIGH_OFFSET 0x2004
```

These definitions are the foundation of the hardware-specific layer of your driver.

## What Comes Next

GPU architecture is diverse. NVIDIA, AMD, Intel, ARM, and Qualcomm all have different architectures with different instruction sets, memory layouts, and programming models. But the fundamental concepts are consistent: command processors interpret command streams, execution units perform work in parallel, memory hierarchies provide data, and display engines output pixels.

The next chapter traces the path from an application's graphics call all the way through the Linux stack to the GPU hardware. Understanding this stack is essential for knowing what your driver needs to support and how userspace will interact with it.

# Chapter 3: The Linux Graphics Stack

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## A Brief History: From Framebuffer to DRM

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## The Modern Stack: Layer by Layer

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## The Complete Path: A Drawing Call End to End

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Userspace vs Kernel Responsibilities

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Render Nodes vs Card Nodes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Mesa and the Kernel: A Partnership

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## What Comes Next

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

# Chapter 4: Direct Rendering Manager

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## What DRM Is and Why It Exists

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## The drm_device Structure

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## The drm_driver Structure

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Device Registration Flow

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Device Management with devm

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## DRM File Operations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## DRM Minor Devices

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## DRM Capabilities

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Debugging Support

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## What Comes Next

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

# Chapter 5: Kernel Mode Setting

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## From UMS to KMS

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Display Pipeline Objects

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## How Objects Are Connected

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Display Modes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## EDID and DDC

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Atomic Mode Setting

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## The ww_mutex Locking Scheme

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Vblank and Timing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Standard Properties

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## What Comes Next

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

# Chapter 6: Graphics Execution Manager

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## What GEM Is and What It Solves

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## The drm_gem_object Structure

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## GEM Object Operations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## GEM Initialization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Creating GEM Objects

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## GEM Handles

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## GEM mmap

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## GEM Prime and Buffer Sharing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## GEM CMA Helper

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## What Comes Next

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

# Chapter 7: Translation Table Maps

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## What TTM Is and Why It Exists

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## TTM Memory Types

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## The ttm_bo Object

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## TTM Driver Interface

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## TTM Initialization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## TTM and GEM Integration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## When to Use TTM vs GEM

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## What Comes Next

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

# Chapter 8: PCI Integration and Device Discovery

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## PCI Enumeration: How Devices Are Found

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## The PCI Device Driver Model

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## PCI Configuration Space

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## PCI BARs: Memory-Mapped Registers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## DMA and DMA Masks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## PCI Interrupts: MSI and MSI-X

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Device Tree for Embedded GPUs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## What Comes Next

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

# Chapter 9: Building the Minimal DRM Driver

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Directory Structure and Kernel Module Layout

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Building the Driver

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Loading and Verifying

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Unloading the Module

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Troubleshooting

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

# Chapter 10: Device Registration and Driver Lifecycle

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Module Init and Exit

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Probe: Discovering and Initializing Hardware

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Probe Error Handling

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Remove: Cleaning Up

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Runtime Power Management

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## System Sleep: Suspend and Resume

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Debugging Registration Failures

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

# Chapter 11: GPU Memory Management

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Allocating GEM Objects from Kernel Space

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## DMA Allocation: Coherent vs Streaming

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Userspace GEM Handles and Ioctls

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Mapping GEM Objects

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## GEM Object Lifecycle and Reference Counting

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Mapping GEM Objects into GPU Address Space

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Implementing a Minimal GEM Memory Manager

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Testing the Memory Manager

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

# Chapter 12: Command Submission

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Command Streams: Structure and Semantics

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Ring Buffers and Circular Queues

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## The DRM Command Submission Ioctl

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Parsing and Validating Command Buffers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Hardware Submission: Writing to GPU Registers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Error Paths: Invalid Commands and Buffer Overruns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

# Chapter 13: GPU Virtual Memory

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Why GPUs Use Virtual Memory

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## GPU Address Spaces and Contexts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Page Table Structures

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## The DRM GPUVA Manager

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Mapping and Unmapping Buffers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Handling GPU Page Faults

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

# Chapter 14: Synchronization and Fences

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Why Synchronization Is Harder with GPUs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## DRM Fences: The Fundamental Synchronization Object

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Software Fences vs Hardware Fences

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Fence Contexts and Timelines

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## DMA Fences and Cross-Device Synchronization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Implementing Fence Signaling from Interrupts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Userspace Fence Waiting

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

# Chapter 15: Interrupts and GPU Completion

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## GPU Interrupts: Types and Sources

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## MSI vs Legacy INTx Interrupts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Requesting and Sharing IRQs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## The Interrupt Handler Structure

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Signaling Fences and Wakeups from Interrupt Context

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Debugging Interrupt Issues

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

# Chapter 16: Display Pipeline and Framebuffers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Framebuffer Registration and Pixel Formats

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Implementing CRTC Enable/Disable and Mode Set

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Plane Composition and Layering

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Connector Detection and EDID Reading

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Atomic Commit for the Display Pipeline

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Scanout and Vblank Timing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

# Chapter 17: The Userspace Interface

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## DRM Ioctls: Standard vs Driver-Private

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Designing a Clean Ioctl Interface

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Versioning and Backward Compatibility

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## The DRM File Private Data Structure

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Userspace Workflow: Open, Get Capabilities, Allocate, Submit, Wait

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## libdrm Integration and uAPI Stability

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Mesa Driver Hooks: Gallium, VK, and Driver Layers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

# Chapter 18: Power Management

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Runtime Power Management: Suspend and Resume

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## System Sleep: Freeze, Thaw, Poweroff, Restore

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## GPU Power States and DVFS

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Clock Gating and Power Gating

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Idle Detection and Autosuspend

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Resuming GPU State After Sleep

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Power Debugging and Measurement

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

# Chapter 19: Debugging GPU Drivers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Kernel Log Analysis: dmesg, printk Levels

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Dynamic Debug: Enabling Per-File Logging

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## ftrace and Function Graph Tracing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## DRM-Specific Debugfs Entries

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## DRM Tracepoints for GPU Operations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Using QEMU and Virtual Hardware for Testing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## GDB with Kernel Modules

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Crash Dumps and Stack Traces

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

# Chapter 20: Concurrency and Safety

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Locking Primitives: Mutex, Spinlock, Rwlock

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## DRM's Reservation Locking for Shared Objects

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Lock Ordering and Deadlocks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Reference Counting: Kref, Drm_Ref, Custom

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Memory Ordering: Barriers and Ordering Guarantees

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Race Conditions Specific to GPU Drivers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## RCU for Read-Mostly Data Structures

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

# Chapter 21: GPU Hangs and Error Recovery

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## What a GPU Hang Is and What Causes It

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Detecting Hangs: Watchdogs and Timeouts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## The DRM Reset Infrastructure

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Reset Sequence: Stopping Submission, Resetting, Resuming

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Recovering GPU State: Contexts, Page Tables, Buffers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Handling Stale Fences and Waiting Processes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## GPU Reset and Userspace Notification

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Fault Injection for Testing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

# Chapter 22: Performance Optimization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Profiling GPU Drivers: Where Time Is Spent

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Reducing CPU Overhead in the Fast Path

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Batching and Coalescing Command Submissions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Prefetching and Cache-Friendly Data Structures

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Avoiding Page Faults in GPU Memory

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Interrupt Coalescing and Rate Limiting

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Measuring and Comparing Performance

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

# Chapter 23: Security Considerations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## The GPU Driver as an Attack Surface

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## IOMMU and DMA Protection

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Command Stream Validation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## GPU Sandboxing and VM Isolation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Privilege Escalation via the Driver

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Information Disclosure through GPU Memory

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Secure Boot and Signed Drivers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Best Practices for Security-Hardened Drivers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

# Chapter 24: Reading Upstream DRM Drivers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Choosing Drivers to Study: Simple vs Complex Examples

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Anatomy of a Modern DRM Driver: File Layout

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Understanding DRM Core Abstractions in Practice

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## How to Read and Understand Complex Initialization Sequences

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Identifying Patterns: Common Idioms Across Drivers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Using Coccinelle and Other Tools to Explore Code

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

# Chapter 25: Upstream Development

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## The Linux Graphics Mailing Lists and Maintainers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Finding a Maintainer for Your Driver or Subsystem

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Patch Series Structure and Cover Letters

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Writing Commit Messages for Graphics Drivers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Code Review Expectations and Common Feedback

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## The Review Cycle: What to Expect

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Handling Objections and Iteration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Tools: B4, Git, Patchwork

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

# Chapter 26: Conclusion

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## The Complete Driver: From Nothing to Functional

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## What We Learned and Why It Matters

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Emerging Trends: Vulkan Drivers, GPU Compute, Async Scheduling

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Continuing to Learn: Where to Go Next

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

## Final Encouragement

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.

# References

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/linuxgpudriversfromscratch.