Writing Device Drivers in Linux: A Beginner's Guide

Device drivers act as the bridge between the hardware and the kernel. They allow user applications to interact with hardware devices without knowing the intricate details of how they operate. Linux, being an open-source OS, provides a rich environment for writing, testing, and deploying custom device drivers.

🧩 Anatomy of a Simple Character Device Driver

Here's what's typically needed for a character device:

1. **Register the device** with the kernel
2. **Define file operations** (open, read, write, release)
3. **Handle data exchange**
4. **Unregister on exit**

🧪 Testing the Driver

1. **Compile** with a Makefile
2. **Insert** using `insmod mydriver.ko`
3. **Create device node**:

`bash mknod /dev/mychardev c <major> 0` 4. **Interact**:

`bash echo "hello" > /dev/mychardev cat /dev/mychardev`

📚 Key Concepts

- **Device number (major/minor)**: Identifies the driver and the device.

- **File operations struct**: Maps system calls (`read`, `write`, etc.) to your driver's functions.
- **Copy_to_user / copy_from_user**: Used for safely transferring data between kernel and user space.

🚧 Precautions

- Always test drivers on virtual machines or non-critical systems.
- Use logging (`printk`) generously.
- Be cautious with memory access and pointer dereferencing.

📘 References

- Linux Device Drivers, 3rd Edition
- `man 9` pages: `man 9 register_chrdev`, `man 9 file_operations`
- kernel.org documentation