Linux Device Drivers (LDD) – Notes & Examples

What is Linux?

Linux is a free and open-source operating system (OS) based on UNIX, created in 1991 by Linus Torvalds.

- Allows users to modify, customize, and redistribute the source code.
- Various distributions (distros) exist for desktops, servers, and embedded devices.

What is a Driver?

A driver is a specialized piece of software (usually a kernel module) that allows the Linux kernel to communicate with:

- Hardware devices
- Kernel subsystems
  in an abstracted and standardized way.

Types of Linux Drivers

1. Hardware Drivers – Directly interact with hardware
    - Examples: character drivers, block drivers, network drivers
2. Software Subsystem Drivers – Provide interfaces for kernel subsystems
    - Examples: file systems, protocols (IPv4/IPv6, TCP, UDP), security layers
3. Virtual/Kernel Service Drivers – Simulate devices or expose kernel features to user space

---

Steps to Write a Linux Device Driver

Every driver has at least two essential functions:
Init function – Runs when the module is inserted

```
static int __init my_driver_init(void) {
   // Register driver with subsystem
   return 0;
}
```

1.

Exit function – Runs when the module is removed

```
static void __exit my_driver_exit(void) {
   // Unregister driver
}
```

2.

Register the init & exit functions

```
module_init(my_driver_init);
module_exit(my_driver_exit);
```

3.

📊 Driver Types & Kernel Interfaces

Driver Type	Init API	Exit API	Interface Struct	Key Structures

Character Driver	alloc_chrdev_region()	unregister_chrdev_region()	struct file_operations	struct cdev, struct file, struct inode
Block Driver	register_blkdev()	unregister_blkdev()	struct block_device_operations	struct gendisk, struct request_queue
Network Driver	register_netdev()	unregister_netdev()	struct net_device_ops	struct net_device
Platform Driver	platform_driver_register()	platform_driver_unregister()	struct platform_driver	struct platform_device, Device Tree
PCI Driver	pci_register_driver()	pci_unregister_driver()	struct pci_driver	struct pci_dev, struct pci_device_id
USB Driver	usb_register()	usb_deregister()	struct usb_driver	struct usb_device, usb_interface
I2C Driver	i2c_add_driver()	i2c_del_driver()	struct i2c_driver	struct i2c_client, struct i2c_adapter
SPI Driver	spi_register_driver()	spi_unregister_driver()	struct spi_driver	struct spi_device, struct spi_master
Input Driver	input_register_device()	input_unregister_device()	struct input_dev	input_event() APIs
Regulator Driver	regulator_register()	Handled by devm_*	struct regulator_ops	struct regulator_dev, struct regulator_desc

---

Example: Character Device Driver

Headers

```
#include <linux/module.h>
#include <linux/init.h>
#include <linux/fs.h>
#include <linux/cdev.h>
#include <linux/device.h>
#include <linux/slab.h>
#include <linux/uaccess.h>
#include <linux/ioctl.h>
```

Minimal Character Driver Example

```
#define DEVICE_NAME "pseudo_char_dev"
#define CLASS_NAME  "pseudo_class"
#define BUFFER_SIZE 1024
```

```
static struct file_operations pseudo_fops = {
    .owner = THIS_MODULE,
    .open = pseudo_open,
    .release = pseudo_release,
    .read = pseudo_read,
    .write = pseudo_write,
    .unlocked_ioctl = pseudo_ioctl,
};

static int __init pseudo_init(void) {
    alloc_chrdev_region(&dev_num, 0, 1, DEVICE_NAME);
    kmalloc(BUFFER_SIZE, GFP_KERNEL);
    cdev_init(&pseudo_cdev, &pseudo_fops);
    cdev_add(&pseudo_cdev, dev_num, 1);
    pseudo_class = class_create(THIS_MODULE, CLASS_NAME);
    device_create(pseudo_class, NULL, dev_num, NULL, DEVICE_NAME);
    return 0;
}

static void __exit pseudo_exit(void) {
    device_destroy(pseudo_class, dev_num);
    class_destroy(pseudo_class);
    cdev_del(&pseudo_cdev);
    unregister_chrdev_region(dev_num, 1);
}

module_init(pseudo_init);
module_exit(pseudo_exit);
MODULE_LICENSE("GPL");
```

Advanced Example: Multi-Device Character Driver

This driver:

- Creates multiple /dev/mycharX devices
- Uses per-device buffers
- Supports open, read, write, release
- Uses mutexes to protect concurrent access

✅ Key Takeaways

- Every driver needs init & exit functions.
- Kernel provides registration APIs depending on driver type.
- struct file_operations is the heart of character drivers.
- Device files in /dev are created using udev + class_create() + device_create().
- Use synchronization primitives (mutex, spinlock) to handle concurrency.
