**💡 Linux 字符设备驱动开发笔记总结（myled 实例）**

**一、整体开发流程（内核态 + 用户态）**

用户空间：

┌──────────────────────────────────────────┐

│ 应用层 test\_led.c │

│ ├─ open("/dev/myled", O\_RDWR) ──────────┐│

│ ├─ read(fd, buf, len) ││

│ ├─ write(fd, buf, len) ││

│ └─ close(fd) ││

└──────────────────────────────────────────┘│

│

内核空间（驱动） ▼

┌────────────────────────────────────────────┐

│ myled.ko │

│ ├─ 注册设备号（MKDEV, register\_chrdev） │

│ ├─ 初始化字符设备（cdev\_init + cdev\_add） │

│ ├─ file\_operations 结构体绑定实现函数 │

│ ├─ 创建 class → /sys/class/led\_class/ │

│ └─ 创建 device → /dev/myled │

└────────────────────────────────────────────┘

**二、用户空间应用程序（test\_led.c）**

**主要函数及说明：**

int fd = open("/dev/myled", O\_RDWR); // 打开设备文件

read(fd, read\_buf, 1); // 读取数据（会调用驱动里的 .read）

write(fd, write\_buf, 2); // 写入数据（会调用驱动里的 .write）

close(fd); // 关闭设备文件

**参数说明：**

* argc, argv：命令行参数，常用于传入设备节点路径
* ./test\_led /dev/myled
* argv[0] = "./test\_led"
* argv[1] = "/dev/myled"

**三、内核模块（myled.c）**

**1️⃣ 驱动结构：**

static const struct file\_operations led\_fops = {

.owner = THIS\_MODULE,

.open = led\_chrdev\_open,

.release = led\_chrdev\_close,

.read = led\_chrdev\_read,

.write = led\_chrdev\_write,

};

**2️⃣ 初始化函数：**

static int \_\_init myled\_init(void) {

// 1. 获取设备号

dev\_num = MKDEV(240, 0);

register\_chrdev\_region(dev\_num, 1, "myled");

// 2. 初始化并添加 cdev

cdev\_init(&led\_cdev, &led\_fops);

cdev\_add(&led\_cdev, dev\_num, 1);

// 3. 创建设备节点 /dev/myled

led\_class = class\_create(THIS\_MODULE, "led\_class");

led\_device = device\_create(led\_class, NULL, dev\_num, NULL, "myled");

}

**3️⃣ 退出函数：**

static void \_\_exit myled\_exit(void) {

device\_destroy(led\_class, dev\_num);

class\_destroy(led\_class);

cdev\_del(&led\_cdev)；

unregister\_chrdev\_region(dev\_num, 1);

}

**四、file\_operations 中各函数实现与含义**

| **函数名** | **作用说明** |
| --- | --- |
| .open | 设备打开时调用，一般用于初始化资源 |
| .release | 设备关闭时调用 |
| .read | 应用程序调用 read() 时触发，从内核向用户传数据 |
| .write | 应用程序调用 write() 时触发，从用户向内核传数据 |

**五、内核 API 函数详解**

| **函数** | **说明** |
| --- | --- |
| MKDEV(主, 次) | 合成设备号 |
| register\_chrdev\_region() | 向内核注册设备号 |
| cdev\_init() | 初始化 cdev 对象 |
| cdev\_add() | 将 cdev 添加到内核 |
| class\_create() | 创建一个设备类（生成 /sys/class 目录） |
| device\_create() | 在 /dev 下生成对应设备节点 |
| copy\_to\_user() | 从内核空间复制数据到用户空间 |
| copy\_from\_user() | 从用户空间复制数据到内核空间 |

**六、用户空间与内核空间数据传递机制**

* **用户 → 内核**：通过 .write() + copy\_from\_user()
* **内核 → 用户**：通过 .read() + copy\_to\_user()

// 内核读取数据到用户空间（to,from,size）

copy\_to\_user(user\_buf, kernel\_buf, size);

// 内核接收用户数据（to,from,size）

copy\_from\_user(kernel\_buf, user\_buf, size);

**七、错误处理机制与 return 值说明**

* copy\_to\_user() / copy\_from\_user() 返回值 >0 说明部分未成功复制，建议返回 -EFAULT
* .read() 和 .write() 的返回值应该是 **实际成功处理的字节数**

**八、常见路径生成关系**

| **路径** | **来源** |
| --- | --- |
| /dev/myled | device\_create() 创建设备节点 |
| /sys/class/led\_class | class\_create() 创建设备类 |
| /proc/devices | 注册后可查看设备号与名称 |

**九、总结关键点（助记）**

* **驱动注册核心三件套**：
  + register\_chrdev\_region() → 注册设备号
  + cdev\_add() → 注册字符设备
  + device\_create() → 创建设备节点
* **数据读写必须安全拷贝**：
  + copy\_to\_user() / copy\_from\_user() 是用户空间与内核空间通信桥梁
* **资源释放顺序要注意**：
  + 遵循「**先创建，后释放**」的逆序清理原则，避免内核资源泄漏或崩溃

**十、拓展理解：字符设备和驱动的本质**

* **字符设备**：以字符为单位进行读写的设备，例如 LED、串口、按键、GPIO 等。
* **驱动程序**：是内核中的一段程序，用于操作硬件；字符驱动是最常见的入门类型。

✅ **建议：**

* 用 dmesg 查看内核 log（例如 printk 打印信息）
* 多练习：改写驱动增加功能（比如操作实际 GPIO 控制 LED）
* 学会查看 /sys/class、/dev、/proc/devices 辅助调试设备注册情况