第 16 章　驱动开发之字符设备驱动程序框架

本章目标

- 掌握编写一个字符设备驱动程序的框架、必须的步骤

16.1　字符设备驱动程序框架简介

我们在学习 C 语言的时候，知道每个应用程序的入口函数，即第一个被执行的函数是 main 函数，那么，我们自己的驱动程序，哪个函数是入口函数呢？

在写驱动程序的时候，如果函数的名字可以任意取，常常为 xxxx_init()，当实现好这个 xxxx_init()函数以后，内核其实并不知道这个就是我们驱动的入口函数，因此我们要想办法告诉内核，我们的入口函数是哪个？我们通过 module_init()函数来告诉内核，具体如下。

```
module_init(f403tech_drv_init);
```

通过上面的修饰以后，f403tech_drv_init()这个函数就变成了我们的驱动程序的入口函数了。当然，有入口函数，自然还需要一个出口函数，我们通过 module_exit()函数来告诉内核，具体如下。

```
module_exit(f403tech_drv_exit);
```

从上一章中，我们知道，应用程序是通过 open、read、write ...函数来和我们的驱动程序进行交互的，那么我们的驱动程序是怎么给应用程序提供这些接口的呢？

我们在写驱动程序的时候，我们首先需要定义出一个 file_operations 结构体，该结构体便是驱动和应用程序交互的接口。具体定义如下。

```
struct file_operations {
    struct module *owner;
    loff_t (*llseek) (struct file *, loff_t, int);
    ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
    ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
    ssize_t (*aio_read) (struct kiocb *, const struct iovec *, unsigned long,
loff_t);
    ssize_t (*aio_write) (struct kiocb *, const struct iovec *, unsigned long,
loff_t);
    int (*readdir) (struct file *, void *, filldir_t);
```

```
    unsigned int (*poll) (struct file *, struct poll_table_struct *);
    long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
    long (*compat_ioctl) (struct file *, unsigned int, unsigned long);
    int (*mmap) (struct file *, struct vm_area_struct *);
    int (*open) (struct inode *, struct file *);
    int (*flush) (struct file *, fl_owner_t id);
    int (*release) (struct inode *, struct file *);
    int (*fsync) (struct file *, loff_t, loff_t, int datasync);
    int (*aio_fsync) (struct kiocb *, int datasync);
    int (*fasync) (int, struct file *, int);
    int (*lock) (struct file *, int, struct file_lock *);
    ssize_t (*sendpage) (struct file *, struct page *, int, size_t, loff_t *, int);
    unsigned long (*get_unmapped_area)(struct file *, unsigned long, unsigned long,
unsigned long, unsigned long);
    int (*check_flags)(int);
    int (*flock) (struct file *, int, struct file_lock *);
    ssize_t (*splice_write)(struct pipe_inode_info *, struct file *, loff_t *,
size_t, unsigned int);
    ssize_t (*splice_read)(struct file *, loff_t *, struct pipe_inode_info *, size_t,
unsigned int);
    int (*setlease)(struct file *, long, struct file_lock **);
    long (*fallocate)(struct file *file, int mode, loff_t offset,
              loff_t len);
};
```

我们的驱动程序要给应用程序提供哪些接口，就只需要填充相应的成员即可。比如我们想提供 open、read、write 这三个接口，就应该如下定义。

```
static struct file_operations f403tech_drv_fops = {
    .owner  = THIS_MODULE,      /* 这是一个宏，推向编译模块时自动创建的
__this_module 变量 */
    .open   = f403tech_drv_open,
    .read   = f403tech_drv_read,
    .write  = f403tech_drv_write,
};
```

当 file_operations 结构体定义、设置好以后，我们只需要通过 register_chrdev()函数将该机构图注册进内核即可。

16.2 字符设备驱动程序框架实现

经过前面部分的讲解，相信大家一定对如何写一个自己的驱动程序，有所感悟了。接下来，给大家一个简单的驱动程序的例子，可以用于作为框架模板，以后的驱动都可以基于该驱动进行修改。

```
/*
** 包含一些头文件，这是我们写驱动程序所必须的。
** 问：我们在写驱动的时候，应该包含哪些头文件呢？
** 答：在写驱动程序的时候，我们并不用刻意的去记需要包含哪些头文件，
**     我们只需要参考其他的驱动程序的头文件即可，说通俗一点，就是
**     先把内核中其他驱动程序的头文件先复制过来用着，然后在编译的
**     时候，再更加提示信息来添加或者修改头文件。
*/
#include <linux/mm.h>
#include <linux/miscdevice.h>
#include <linux/slab.h>
#include <linux/vmalloc.h>
#include <linux/mman.h>
#include <linux/random.h>
#include <linux/init.h>
#include <linux/raw.h>
#include <linux/tty.h>
#include <linux/capability.h>
#include <linux/ptrace.h>
#include <linux/device.h>
#include <linux/highmem.h>
#include <linux/crash_dump.h>
#include <linux/backing-dev.h>
#include <linux/bootmem.h>
#include <linux/splice.h>
#include <linux/pfn.h>
#include <linux/export.h>
#include <linux/io.h>
#include <linux/aio.h>
#include <linux/kernel.h>
#include <linux/module.h>

#include <asm/uaccess.h>
```

```
#define DEVICE_NAME     "f403tech"  /* 加载模式后，执行”cat /proc/devices”命令看到的设备名称 */
#define F403TECH_MAJOR       0       /* 主设备号 */

static struct class *f403tech_drv_class;

static int f403tech_drv_open(struct inode *inode, struct file *file)
{
    printk("%s:Hello F403Tech\n", __FUNCTION__); // printk 用于驱动中添加打印，用法和应用程序中的 printf 一样

    return 0;
}

static ssize_t f403tech_drv_read(struct file *file, char __user *buf, size_t size, loff_t *ppos)
{
    printk("%s:Hello F403Tech\n", __FUNCTION__); // printk 用于驱动中添加打印，用法和应用程序中的 printf 一样

    return 0;
}

static ssize_t f403tech_drv_write(struct file *file, const char __user *buf, size_t size, loff_t *ppos)
{
    printk("%s:Hello F403Tech\n", __FUNCTION__); // printk 用于驱动中添加打印，用法和应用程序中的 printf 一样

    return 0;
}

/* 这个结构是字符设备驱动程序的核心
** 当应用程序操作设备文件时所调用的 open、read、write 等函数，
** 最终会调用这个结构中指定的对应函数
**/
static struct file_operations f403tech_drv_fops = {
    .owner  = THIS_MODULE,      /* 这是一个宏，推向编译模块时自动创建的
```

```
__this_module 变量 */
    .open    = f403tech_drv_open,
    .read    = f403tech_drv_read,
    .write   = f403tech_drv_write,
};

int major;
/*
** 执行 insmod 命令时就会调用这个函数
*/
static int __init f403tech_drv_init(void)
{
    /* 注册字符设备
    ** 这步是写字符设备驱动程序所必须的
    ** 参数为主设备号、设备名字、file_operations 结构;
    ** 这样，主设备号就和具体的 file_operations 结构联系起来了，
    ** 操作主设备为 F403TECH_MAJOR 的设备文件时，就会调用 f403tech_drv_fops 中的相
关成员函数
    ** F403TECH_MAJOR 可以设为 0，表示由内核自动分配主设备号
    */
    major = register_chrdev(F403TECH_MAJOR, DEVICE_NAME, &f403tech_drv_fops);
    if (major < 0)
    {
        printk(DEVICE_NAME " can't register major number\n");
        return major;
    }

    /*
    ** 以下两行代码用于创建设备节点，是必须的。
    ** 当然，如果不写这两行，那么就必须在 linux 系统命令行中通过 mknod 这个命令来
创建设备节点
    */
    /* 创建类 */
    f403tech_drv_class = class_create(THIS_MODULE, "f403tech");
    /* 类下面创建设备节点 */
    device_create(f403tech_drv_class, NULL, MKDEV(major, 0), NULL, "f403tech");
        // /dev/f403tech

    /*
```

```
    ** 打印一个调试信息
    */
    printk("%s:Hello F403Tech\n", __FUNCTION__);// printk 用于驱动中添加打印，用法和应用程序中的 printf 一样

    return 0;
}

/*
 * 执行 rmmod 命令时就会调用这个函数
 */
static void __exit f403tech_drv_exit(void)
{
    unregister_chrdev(major, "f403tech");        // 与入口函数的 register_chrdev 函数配对使用
    device_destroy(f403tech_drv_class, MKDEV(major, 0));// 与入口函数的 device_create 函数配对使用
    class_destroy(f403tech_drv_class);  // 与入口函数的 class_create 函数配对使用

    printk("%s:Hello F403Tech\n", __FUNCTION__);// printk 用于驱动中添加打印，用法和应用程序中的 printf 一样
}

/* 这两行指定驱动程序的初始化函数和卸载函数 */
module_init(f403tech_drv_init);
module_exit(f403tech_drv_exit);

/* 描述驱动程序的一些信息，不是必须的 */
MODULE_AUTHOR("http://www.f403tech.com");
MODULE_VERSION("0.1.0");
MODULE_DESCRIPTION("RT5350 FIRST Driver");
MODULE_LICENSE("GPL");
```

注意：

1). 该教程为我司(https://wy-wulian.taobao.com/)原创教程，版权所有；

2). 该教程会不断更新、不断深入，详情请咨询我司客服；

3). 针对该教程，我们还有 QQ 群和论坛，专门负责技术答疑，详情请咨询我司客服。