9.4 字符设备驱动程序

西安邮电大学

字符设备基础

字符设备是指只能一个字节一个字节进行读写操作的设备，不能随机读取设备中的某一数据、读取数据要按照先后数据。字符设备是面向流的设备，常见的字符设备有鼠标、键盘、串口、控制台和LED等。

一般每个字符设备或者块设备都会在/dev目录下对应一个设备文件。Linux用户层程序通过设备文件来使用驱动程序操作字符设备或块设备。

如何描述字符设备

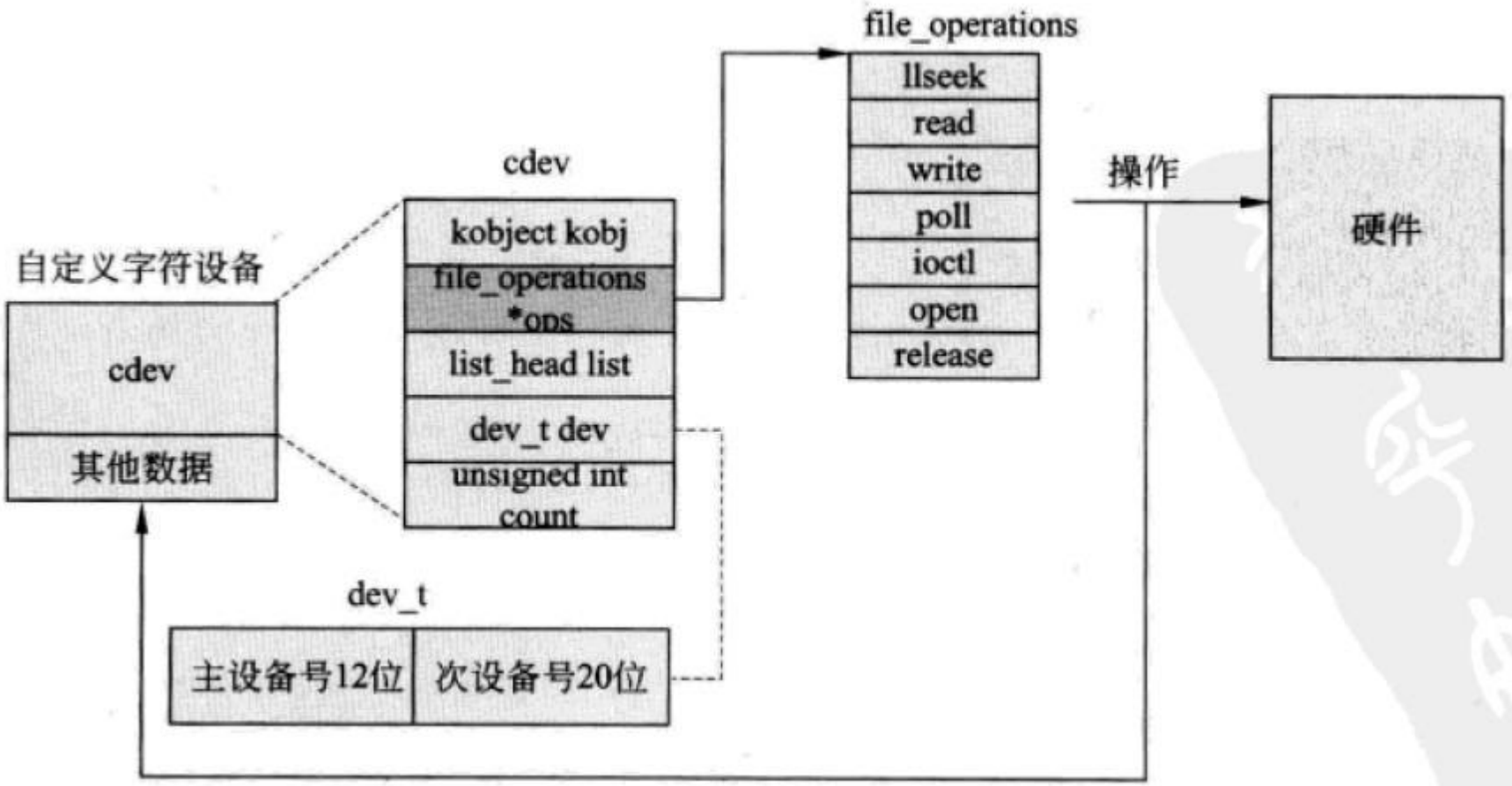
Linux内核中使用struct cdev来表示一个字符设备:其中，最关键的是file_operations结构，它是实现字符设备的操作集。

```
<include/linux/cdev.h>

struct cdev {
    struct kobject kobj;                    //内嵌的内核对象.
    struct module *owner;                   //该字符设备所在的内核模块（所有者）的对象指针，一般为THIS_MODULE主要用于模块计数
    const struct file_operations *ops;      //该结构描述了字符设备所能实现的操作集（打开、关闭、读/写、...），是极为关键的一个结构体
    struct list_head list;                  //用来将已经向内核注册的所有字符设备形成链表
    dev_t dev;                              //字符设备的设备号，由主设备号和次设备号构成（如果是一次申请多个设备号，此设备号为第一个）
    unsigned int count;                     //隶属于同一主设备号的次设备号的个数
    ...
```

cdev与file_operations的关系图

每个字符设备都有一个描述字符设备操作集的file_operations数据结构，它与cdev的关系图如图所示，

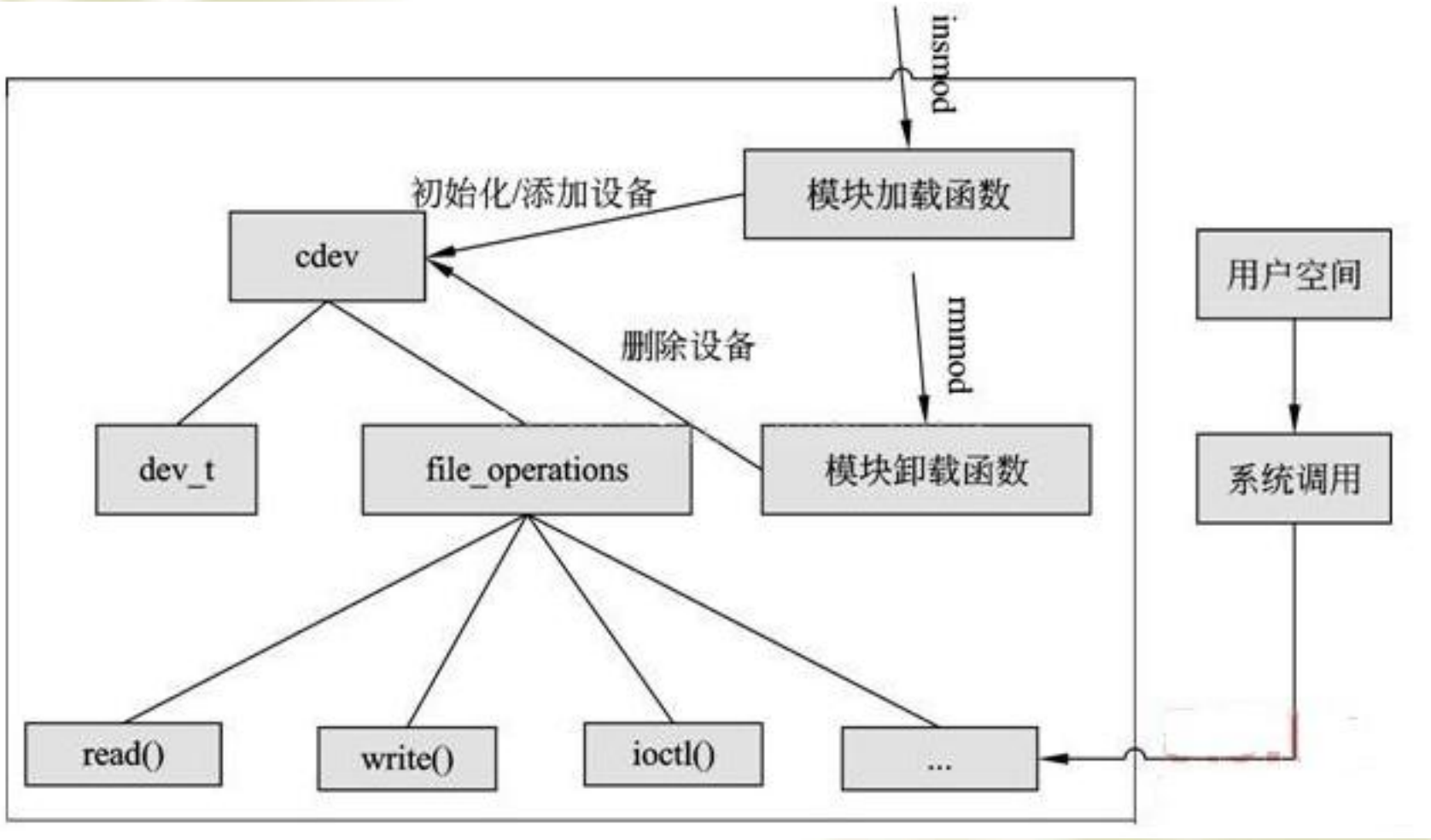


file_operations原型

file_operations数据结构位于fs.h中，其原型为：

```
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 (*read_iter) (struct kiocb *, struct iov_iter *);
        ssize_t (*write_iter) (struct kiocb *, struct iov_iter *);
        int (*iterate) (struct file *, struct dir_context *);
        int (*iterate_shared) (struct file *, struct dir_context *);
        __poll_t (*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 *);
        unsigned long mmap_supported_flags;
        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 (*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);
```

字符设备驱动框架


insmod
初始化/添加设备
模块加载函数
cdev
删除设备
rmmod
用户空间
dev_t
file_operations
模块卸载函数
系统调用
read()
write()
ioctl()
...

字符设备驱动框架

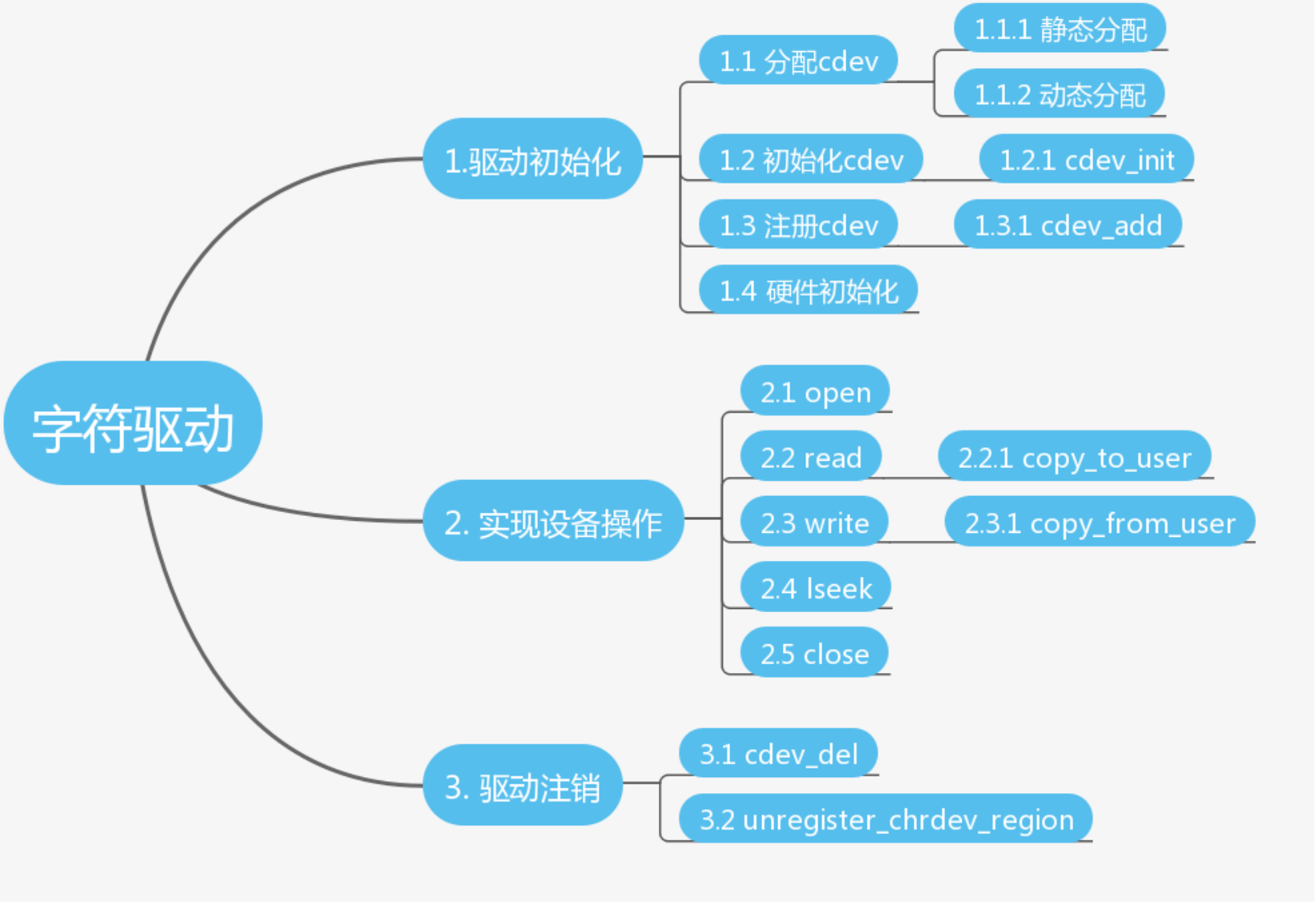
如图，在Linux内核代码中：

1. 使用struct cdev结构体来抽象一个字符设备；

2. 通过一个dev_t类型的设备号一确定字符设备唯一性；

3通过struct file_operations类型的操作方法集来定义字符设备提供给VFS的接口函数。

编写字符设备驱动的步骤



编写字符设备驱动的步骤

如图，编写字符设备驱动分为三大步骤：

1. 驱动的初始化
2. 实现设备的操作
3. 驱动的注销

其中调用的接口函数功能如下：

字符设备驱动接口函数

函数名	功能
cdev_alloc（）	动态申请（构造）cdev内存（设备对象）
cdev_init（）	初始化cdev的成员，并建立cdev和file_operations之间关联起来
cdev_add（）	注册cdev设备对象，也就是添加到系统字符设备列表中
cdev_del（）	将cdev对象从系统中移除（注销 ）
cdev_put（）	释放cdev内存

设备号的申请和释放

一个字符设备或块设备都有一个主设备号和一个次设备号。主设备号用来标识与设备文件相连的驱动程序，用来反映设备类型。次设备号被驱动程序用来辨别操作的是哪个设备，用来区分同类型的设备。

宏或者函数名	功能
MAJOR宏	从设备号中提取主设备号
MINOR宏	从设备号中提取次设备号
MKDEV宏	将主、次设备号拼凑为设备号
register_chrdev_region（）函数	静态申请设备号
alloc_chrdev_region（）	动态申请设备号
unregister_chrdev_region()	释放设备号

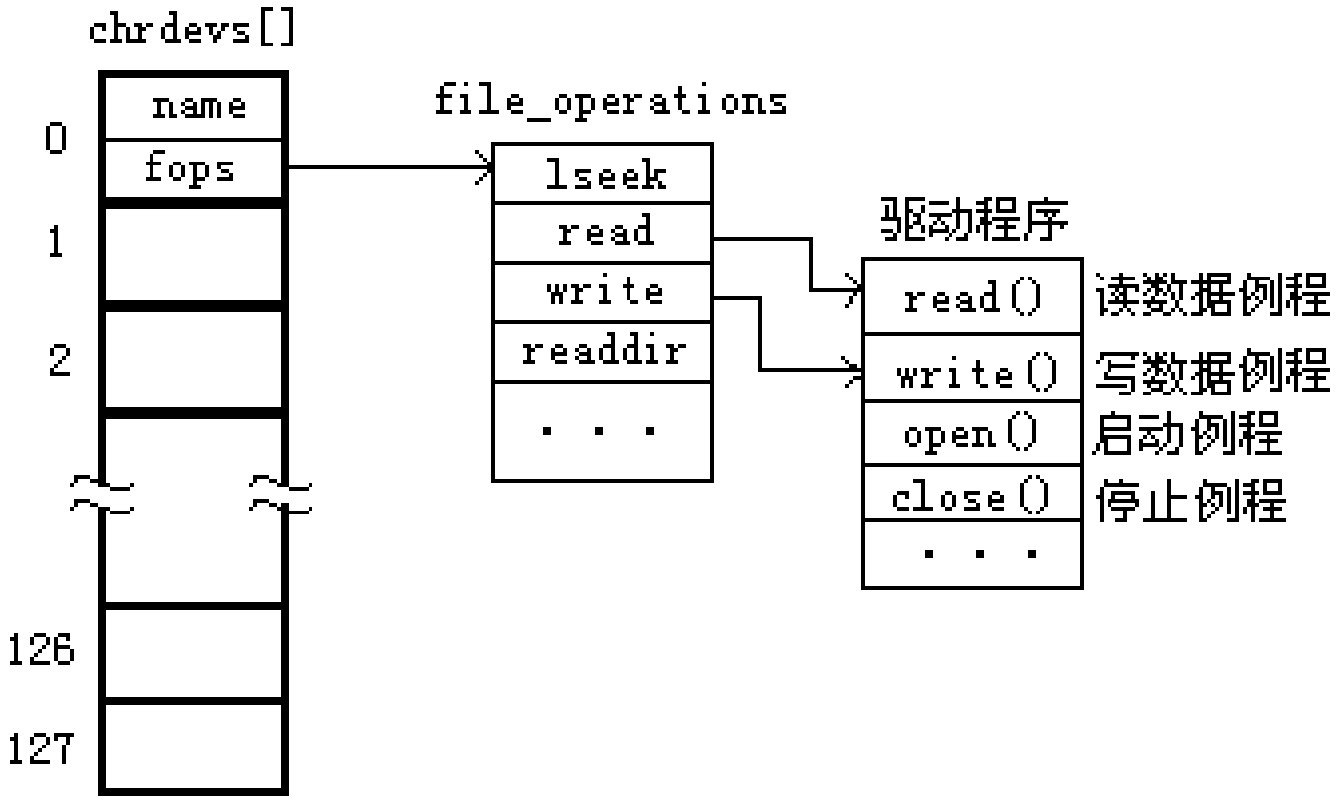
字符设备结构

在Linux中，字符设备是用一个叫做字符设备结构的数据结构char_device_struct来描述的。为了管理上的方便，系统维护了一个数组chrdevs[]，该数组的每一项都代表一个字符设备。

在文件linux/fs/char_dev.c中定义的char_device_struct的数据结构及数组chrdevs []

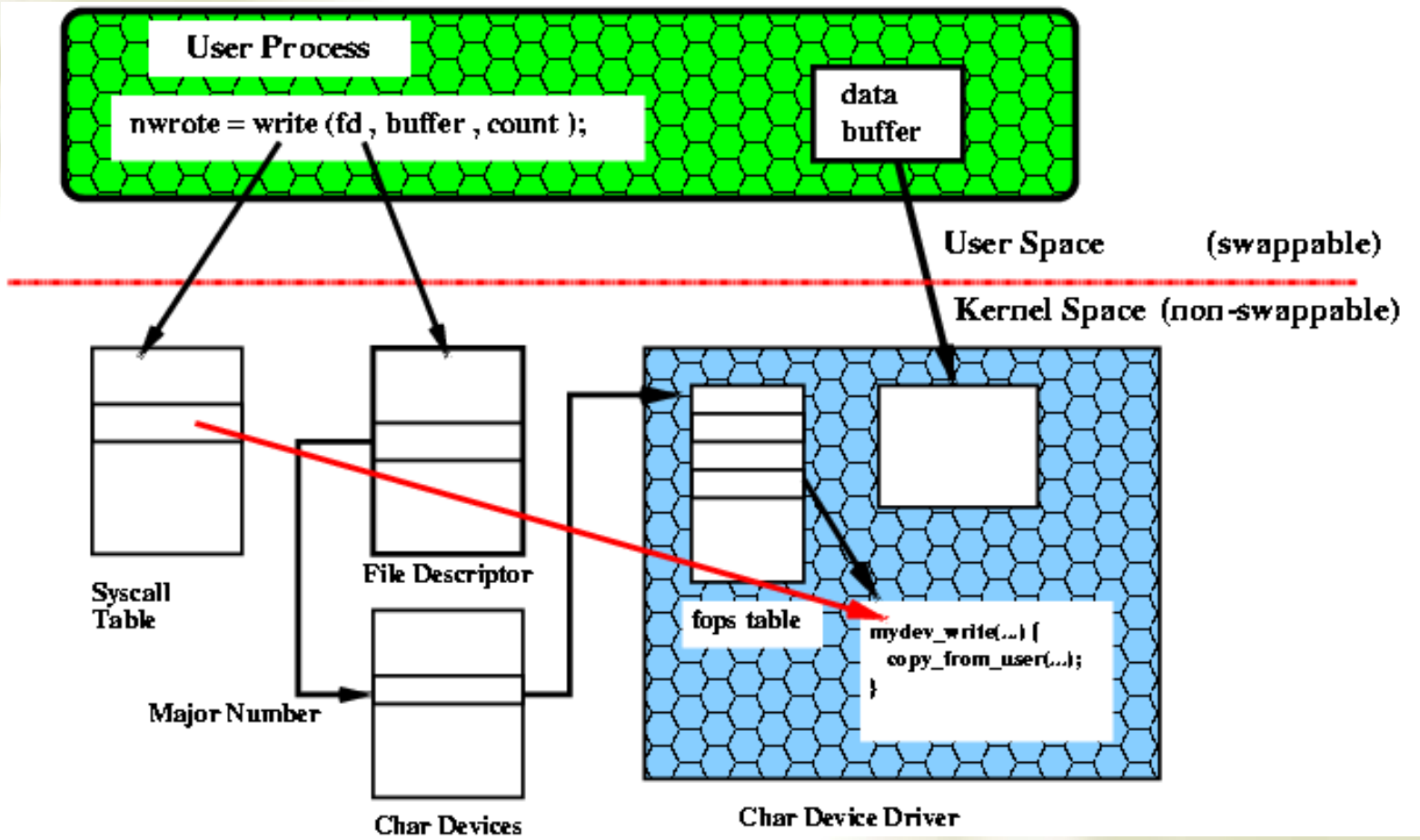
```
static struct char_device_struct {
        struct char_device_struct *next;
        unsigned int major;
        unsigned int baseminor;
        int minorct;
        char name[64];
        struct cdev *cdev;              /* will die */
} *chrdevs[CHRDEV_MAJOR_HASH_SIZE];
```

字符设备驱动程序的注册



char_device_struct结构中的域cdev中的fops是指向文件操作函数集结构的指针。每个注册的驱动的程序在chrdevs表中都有一项。

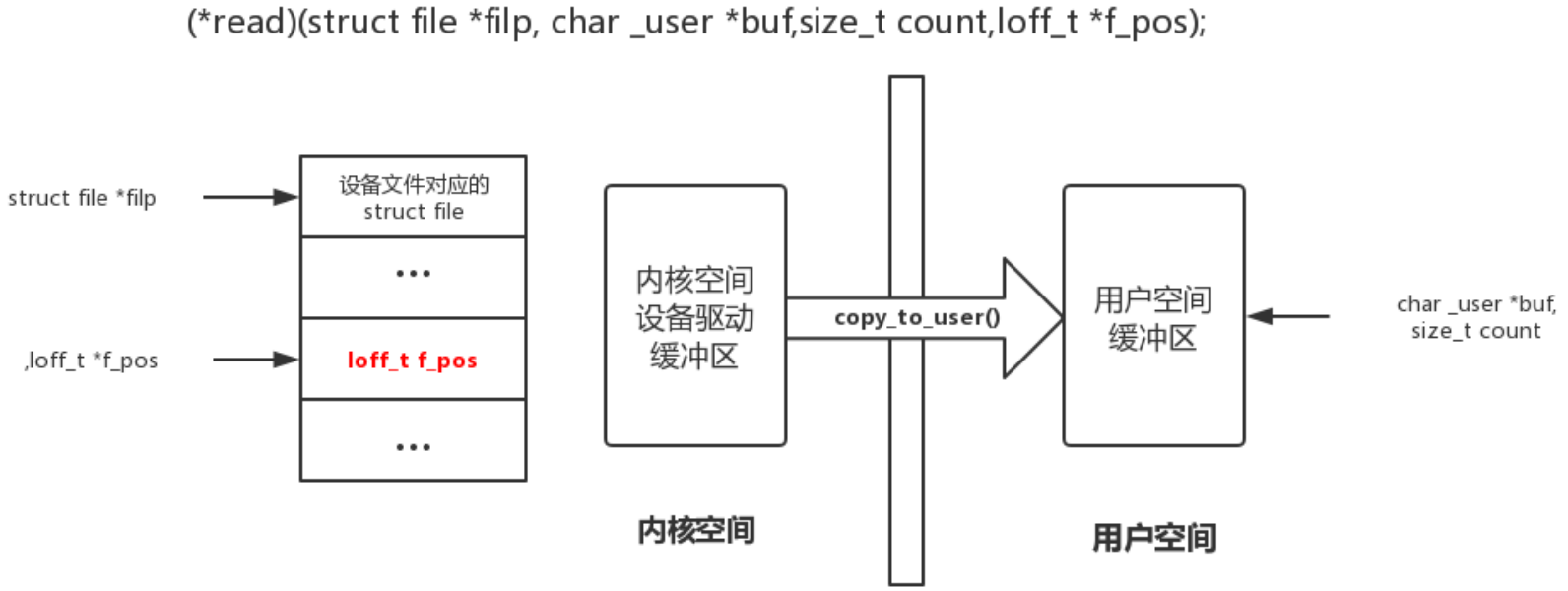
从系统调用到驱动程序


User Process
nwrote = write (fd , buffer , count);
data
buffer
User Space (swappable)
Kernel Space (non-swappable)
Syscall
Table
File Descriptor
Major Number
Char Devices
fops table
mydev_write(...) {
copy_from_user(...);
}
Char Device Driver

从系统调用到驱动程序

当用户进程在调用write系统调用时，则陷入内核，首先查系统调用表，找到write系统调用的服务例程总入口，那么如何找到你自己写的驱动程序mydev_write呢。当我们打开文件时，open()的第一个参数是设备文件名，文件描述符fd就与这个设备文件关联起来了，因此，通过主设备号，在字符设备表中就可以找到对应的驱动程序的入口函数了。

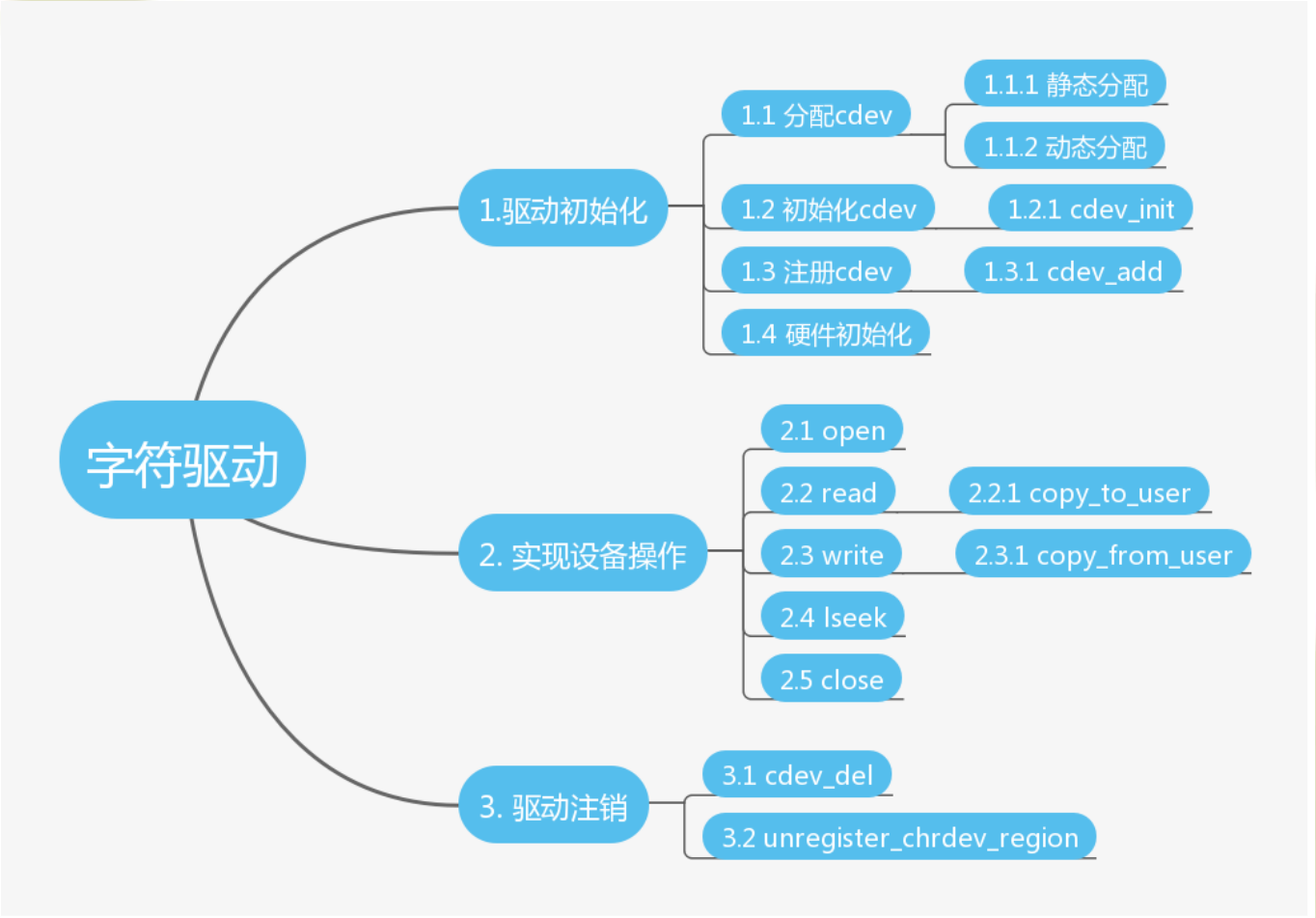
用户空间与内核空间数据的传送



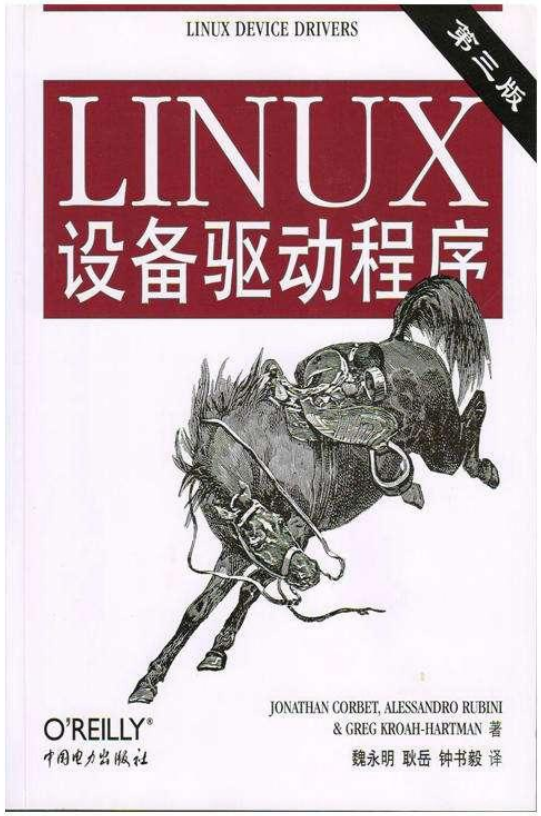
当调用读函数时，通过内核的copy_to_user()函数把内核空间缓冲区中的数据拷贝到用户空间的缓冲区。

编写简单的字符设备驱动程

在动手实践环节，将给出一个字符设备驱动程序的编写过程以及运行机制。



参考文献



1. 《Linux 驱动开发》

2. 网上有大量详尽的字符驱动开发资料，读者可自行查阅，推荐一篇 http://www.cnblogs.com/chen-farsight/p/6155518.html3.

3. 文中的大多数图片来自google搜索，版权归原作者所有

带着疑问上路

从原理上说明file_operations操作方法集，为什么定义了字符设备提供给VFS的接口函数？

谢谢大家！

THANK YOU