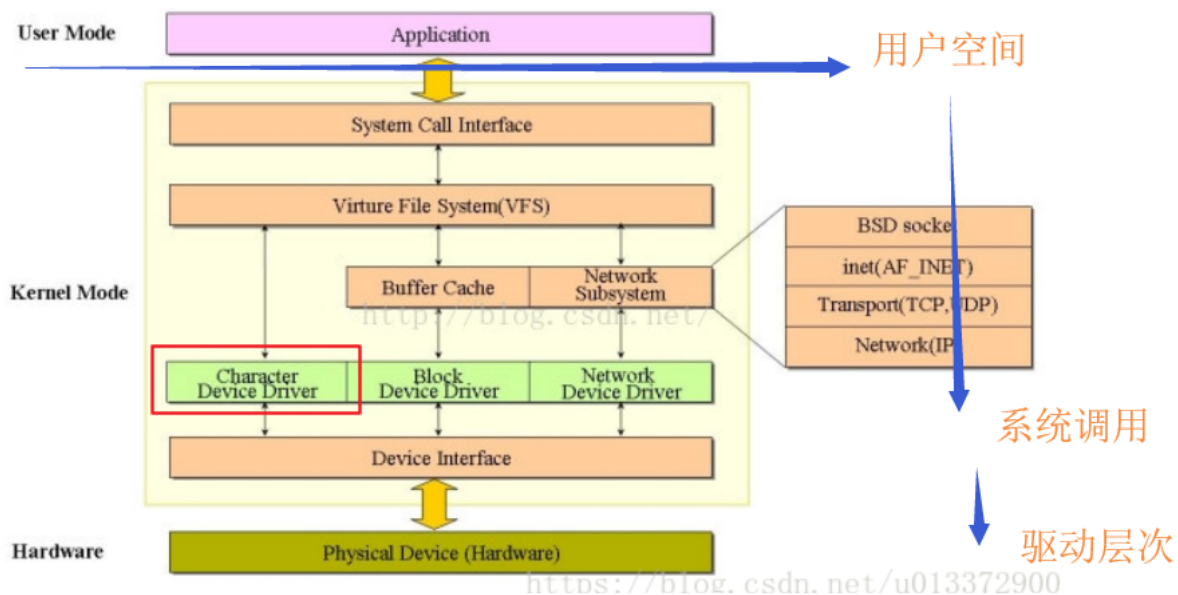


第一课:一大致的对应关系



第二课:

主要内容 :

- 1.定义file_operations , 并实现open write 等相关函数
- 2.把file_operations , 告诉内核(采用: register_chrdev)
- 3.谁来调用 , 来保证注册结构体

module_init(scx200_gpio_init);

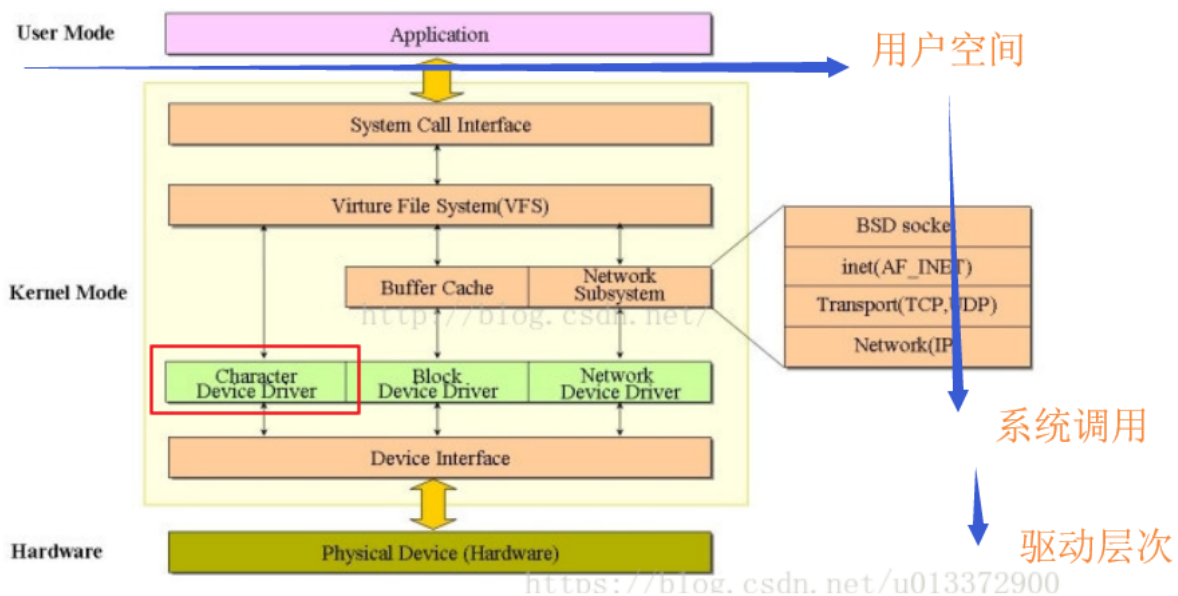
```
module_exit(scx200_gpio_cleanup);
```

- 4.怎么通过主次设备号 , 如何找到最终的结构体

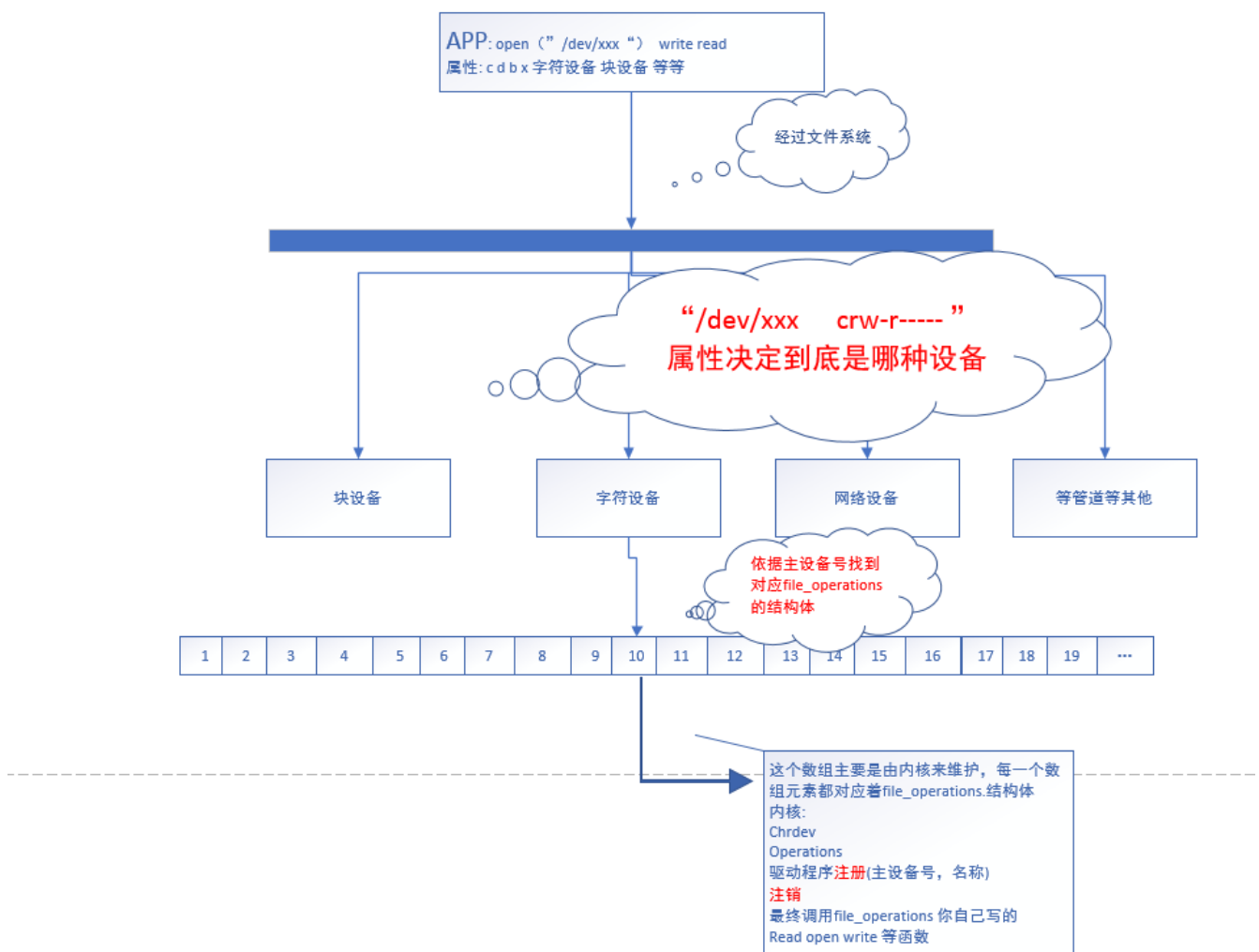
应用程序open("/dev/xxx") ----- 最终如何找到你注册的设备驱动

Linux 2.26对字符设备的管理

注册设备 /dev/xxx :



app-----字符驱动:相关的结构体



总结:

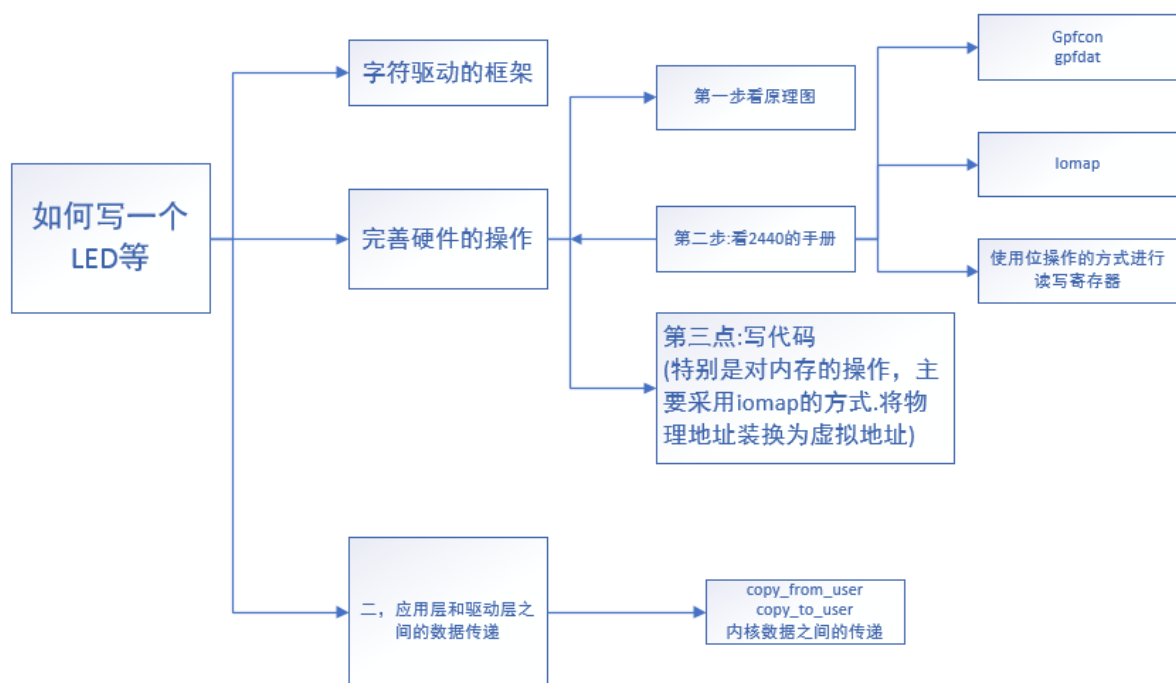
file_operations register_chrdev unregister_chrdev 函数

相关的实验：

实现一个没有硬件的led驱动

第三课:

整体知识体系:



主次设备号:设计一些重要结构体和/proc/devies的文件

主次设备号的管理:

主设备和次设备共同管理,主设备对应了某一类的led设备，次设备对应了某一个设备 led1 led2 led3

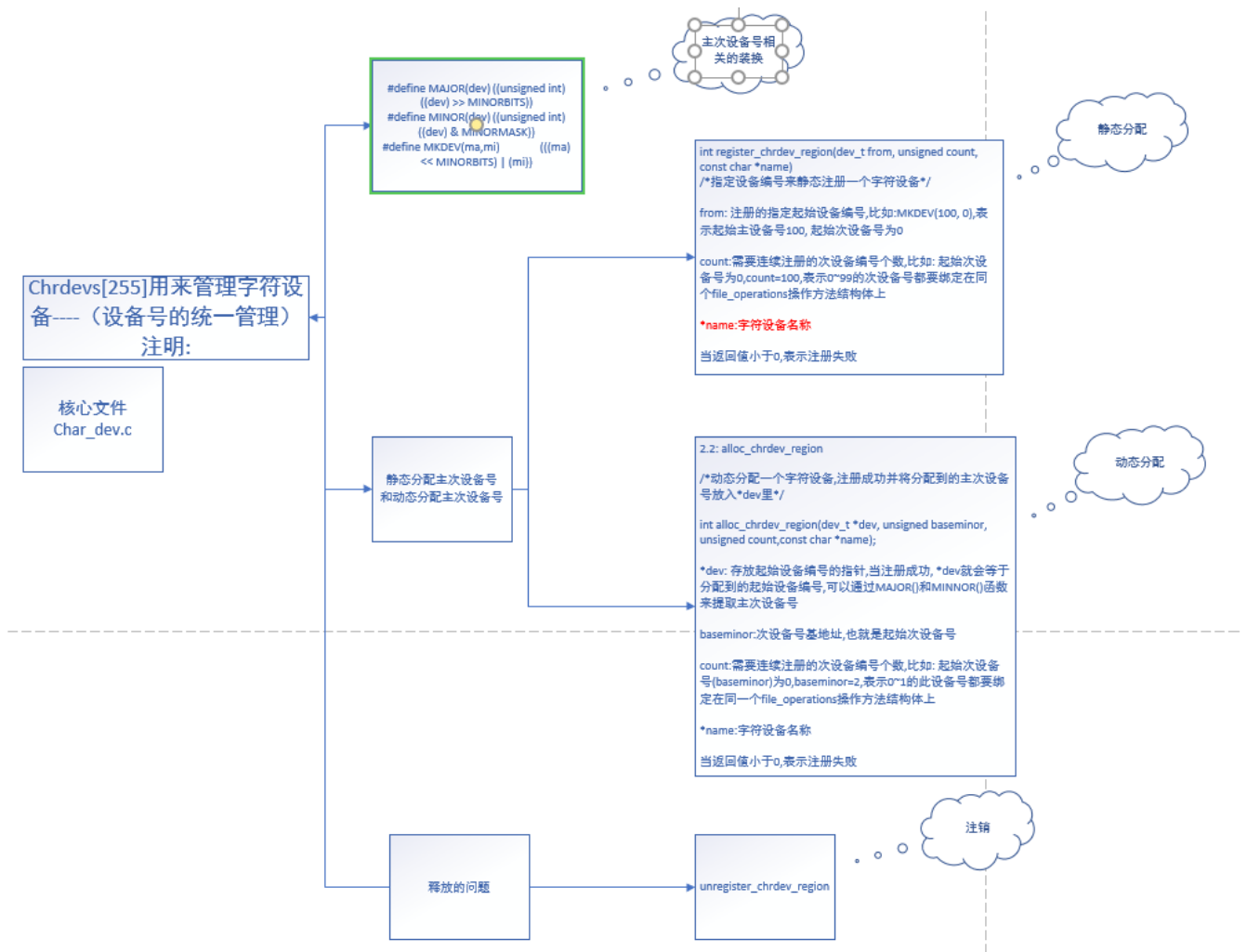
主设备通常对应了某一个**驱动程序**

在Linux 2.6内核管理体系中, `*chrdevs[CHRDEV_MAJOR_HASH_SIZE];`, 因此我们在申请，设备号的时候，实际上内核就是申请了一个`chrdevs[CHRDEV_MAJOR_HASH_SIZE];`一个数组，分配内存。

申请了必须释放掉

主设备和次设备Linux API

#####



字符设备管理struct cdev 结构体的管理

struct cdev *cdev , const struct file_operations *fops 和主次设备号相结合

#####

对字符的设备的管理相关的数据结构好API

主要是对 static struct kobj_map *cdev_map;全局变量，进行相关的分配内存，设置数据，注销内存

Fs/Char_dev.c 文件系统中字符设备管理相关的API

static struct kobj_map *cdev_map;
核心数据结构来管理

struct
cdev *cdev_alloc(void)
分配了一个结构体
void cdev_init
将file_operations挂钩在
struct cdev上

int cdev_add(struct cdev
*p, dev_t dev, unsigned
count)
cdev_map 分配一个空间
并且进行相关的注册

cdev_del
释放字符设备

exact_match, exact_lock

1. 字符设备自己提供锁和匹配函数。
2. 块设备和其他的一些设备也会提供自己的匹配函数和锁

Drivers\base\map.c 用来管理驱动的基础,提供一组API将数据字符, 块设备等抽象为一个数据结构

kobj_map
这个函数, cdev_map中
probe分配空间, 进行参
数判定

kobj_unmap
释放相关的空间

Fs/Char_dev.c 文件系统块设备的注册方式

static struct kobj_map
*bdev_map;
核心的数据结构来管理

.....

.....

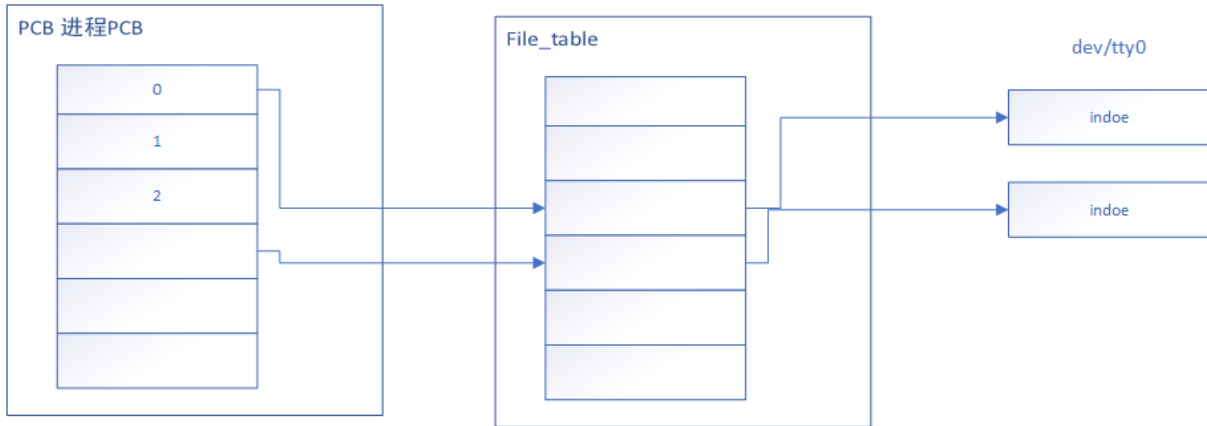
.....

应用程序实现open write read等函数的实质

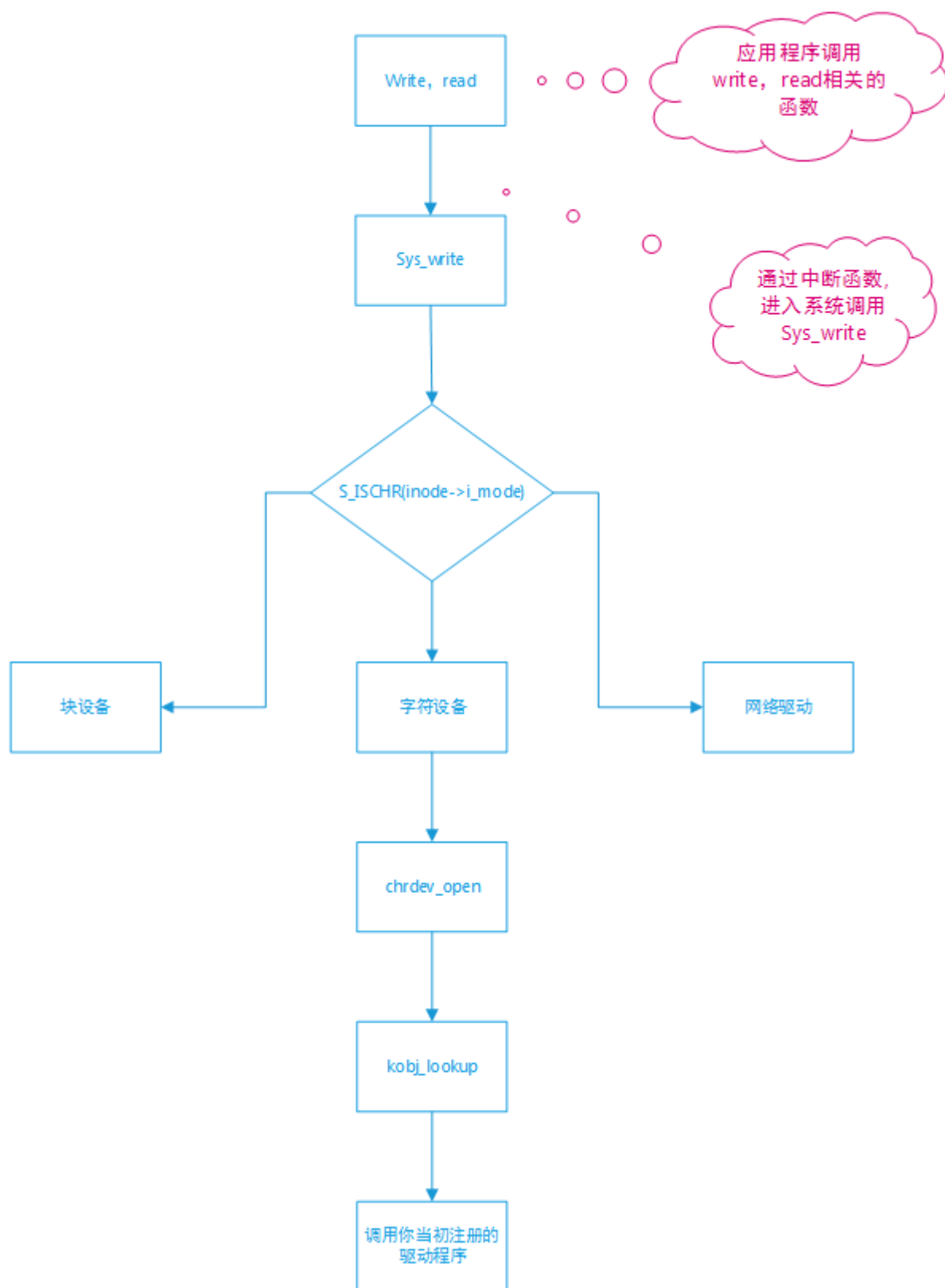
open函数的作用, 获取file_name的inode节点

应用层调用open函数

```
int open(const char *pathname, int flags);  
经过系统中断，进入系统函数  
Int sys_open(const char *pathname, int flag  
i= open_namei(filename, flag, &inode)  
.....  
F->f_inode = inode
```



获取inode节点之后的write和read



盲区:

kobject kobject kset kobj_type 管理的问题，和相关结构体具体表的含义(等待高手解答吧)

