# Linux驱动原始架构

- **设备号dev_t**
  - 概念：主设备号（高12位） + 次设备号（低20位）
  - 定义：
    - typedef u32 __kernel_dev_t;
      typedef __kernel_dev_t dev_t;
  - 相关宏
    - MAJOR(dev)
    - MINOR(dev)
    - MKDEV(ma,mi)
- **cdev结构体**
  - 概念：描述字符设备的
  - 主要字段
    - struct kobject kobj
      Llnux设备管理体系用到
    - struct module *owner
      所属的内核模块
    - **const struct file_operations *ops**
      **设备的操作手册**
    - struct list_head list
      设备间的联系关系（链表）
    - dev_t dev
      设备号
    - unsigned int count
      同类设备的数量
- **struct kobj_map cdev_map结构体**
  - 概念：用于管理所有的cdev结构体
  - 索引 = major % 255
- **设备节点**
  - 概念：表示硬件设备的文件，在/dev目录下
  - 用户通过这个文件，可以访问设备
  - 创建方式
    - 设备节点相关API
      - mknod 设备名 设备类型 主设备号 次设备号
      - 创建一个设备节点
        - struct device *device_create(struct class *class,struct device *parent,dev_t devt,void *drvdata,const char *fmt,...)
      - 删除一个设备节点
        - void device_destory(struct class *class,dev_t devt)
- **file结构体**
  - 概念：内核表示“每个打开文件”的数据结构
  - **后续用户操作的主题对象**
  - 主要字段
    - const struct file_operations *f_op
      与这个文件相关的所有操作函数指针
    - void *private_data
      私有数据指针
- **file_operations结构体**
  - 概念：定义应用程序如何和设备驱动程序交互
  - 主要字段
    - struct module *owner
      所属模块
    - open回调函数
      设备打开会调用，用于硬件初始化
    - release回调函数
      设备释放函数
    - llseek回调函数
      文件指针移动函数，改变读写位置
    - read函数
      从设备读取数据到用户程序
      - copy_to_user
        从内核空间拷贝数据到用户空间
      - copy_from_user
        从用户空间拷贝数据到内核空间
      - copy函数
    - write函数
      从用户程序写入数据到设备
- **inode结构体**
  - 概念：Linux文件系统的核心，管理文件系统最基本的单位，包含文件的所有基本信息
  - 一个文件只有一个inode，可以有多个file结构体（多次打开同一个文件）
  - 主要字段
    - dev_t i_rdev
      设备编号
    - struct cdev *i_cdev
      字符设备指针，指向对应的cdev结构体
- **代码框架（流程）一个设备**
  - init函数
    - 1、申请设备号
    - 2、cdev_init关联cdev和fops
    - 3、cdev_add添加cdev到cdev_map散列表
    - 4、注意失败的处理
  - exit函数
    - 1、注销申请的设备号
    - 2、cdev_del删除设备
  - file_operations结构体实现，即上面的fops
    - 1、创建结构体，绑定各个函数
    - 2、open函数（处理？？）
    - 3、release函数（处理？？）
    - 4、write函数
      ①判断文件读写位置和写入的字节数是否超过缓存BUFF_SIZE
      ②copy_from_user：拷贝用户空间数据到内核空间
      ③调整文件读写位置
    - 5、read函数
      ①判断是否超过缓存BUFF_SIZE
      ②copy_to_user：拷贝内核空间数据到用户空间
- **代码框架（多个设备）**
  - 方式一：每个字符设备对接其独立的数据缓冲区
    - 1、修改open函数，通过inode获取设备号
    - 2、根据设备号，调整每个file结构体的private_data
    - 3、private_data指向不同的BUFF
    - 4、修改read/write函数中的buf指向private_data
  - 方式二：自定义结构体，包含cdev和缓冲区
    - 1、根据结构体，创建多个字符设备
      //字符设备1
      static struct chr_dev vcdev1;
      // 字符设备2
      static struct chr_dev vcdev2;
    - 2、修改init函数，cdev_init和cdev_add独自执行
    - 3、修改exit函数，cdev_del独自指向
    - 4、修改open和release函数的private_data指向自定义的结构体
    - 5、修改write和read函数
      buf指向private_data中的缓冲成员变量
      - container_of函数
        - 参数
          - inode->i_cdev获取cdev
          - 自定义的结构体
          - 自定义结构体中的对应的成员名称
        - 返回
          - 具体自定义的机构提

- 设备号相关API
  - 静态注册一个或多个设备号
    - int register_chrdev_region(dev_t from,unsigned count,const char *name)
      name可以在/proc/devices中看到
  - 动态分配设备号
    - int alloc_chrdev_region(dev_t *dev,unsigned baseminor,unsigned count,const char *name)
      通过 cat /proc/devices 查询分配的主设备号
  - 释放设备号
    - void unregister_chrdev_region(dev_t from,unsigned count)
  - 静态或动态的分配设备号
    - static inline int register_chrdev(unsigned int major,const char *name,const struct file_operations *fops){
      return __register_chrdev(major,0,256,name,fops);
      }
      一次申请了256个，资源浪费
  - 对应的注销函数
    - static inline void unregister_chrdev(unsigned int major,const char *name){
      __unregister_chrdev(major,0,256,name);
      }

- 字符设备相关API
  - 定义字符设备
    - static struct cdev chrdev
    - struct cdev *cdev_alloc(void);
  - 移出字符设备
    - void cdev_del(struct cdev *p);
  - 初始化cdev（关联fops）
    - void cdev_init(struct cdev *cdev,const struct file_operations *fops)
  - 注册cdev设备
    - int cdev_add(struct cdev *p,dev_t dev,unsigned count);
  - 删除cdev设备
    - void cdev_del(struct cdev *p)