文档说明：
本流程图配套视频教程：《Linux内核编程》05期：字符设备驱动
作者博客：www.zhaixue.cc
淘宝店：https://wanglitao.taobao.com

Linux字符设备驱动架构

用户驱动：rtc字符驱动

```
drivers/rtc.c
static const struct file_operations rtc_fops = {
    .owner    = THIS_MODULE,
    .open     = rtc_open,
    .read     = rtc_read,
    .write    = rtc_write,
    .unlocked_ioctl = rtc_ioctl,
    .release = rtc_release,
};

static int __init rtc_init(void)
{
    int ret = 0;

    regs = (rtc_reg_t *)ioremap(RTC_BASE, sizeof(rtc_reg_t));
    printk("rtc_init\n");

    ret = register_chrdev(222, "rtc-demo", &rtc_fops);
    if (ret < 0) {
        printk("Register char module: rtc failed..\n");
        return 0;
    }
    else {
        printk("Register char module: rtc success!\n");
    }

    return 0;
}
```

register_chrdev注册流程

1. 申请设备号：__register_chrdev_region
2. 创建cdev对象：cdev_alloc
3. 初始化cdev对象：cdev->ops = fops
4. 将cdev添加到驱动模型中：cdev_add，kboj、kset
5. 将cdev和chrdevs数组中的char_device_struct建立关联：cd->cdev = cdev

```
include/linux/fs.h  fs/chr_dev.c
static inline int register_chrdev(unsigned int major, const char *name,
□□□□  const struct file_operations *fops)
{
□return __register_chrdev(major, 0, 256, name, fops);
}
int __register_chrdev(unsigned int major, unsigned int baseminor,
□□       unsigned int count, const char *name,
□□       const struct file_operations *fops)
{
□struct char_device_struct *cd;
□struct cdev *cdev;
□int err = -ENOMEM;

□cd = __register_chrdev_region(major, baseminor, count, name);
□cdev = cdev_alloc();
□cdev->owner = fops->owner;
□cdev->ops = fops;
□kobject_set_name(&cdev->kobj, "%s", name);
□cdev_add(cdev, MKDEV(cd->major, baseminor), count);
□cd->cdev = cdev;

□return major ? 0 : cd->major;
}
```

```
#define CHRDEV_MAJOR_HASH_SIZE 255
// 全局数组每个char_device_struct元素对应一个主设备号
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];
```

```
struct cdev {
    struct kobject kobj;
    struct module *owner;
    const struct file_operations *ops;
    struct list_head list;
    dev_t dev;
    unsigned int count;
}
```

主设备号12位：次设备号20位

open("dev/rtc-demo0", O_RDWR);

1. 通过系统调用，调用sys_open，根据设备名对应的inode信息i_flags找到设备类型，创建file
2. 根据设备号(inode->i_rdev)查找cdev：kboj_lookup
3. 保存找到的cdev：inode->i_cdev = cdev
4. 将inode添加到cdev->list链表中
5. 用cdev中的ops初始化file：file->f_op = cdev->ops
6. 调用open回调函数：file->f_op->open(inode, filp)
7. open执行完成，通过VFS返回给用户空间一个文件描述符：fd
   fd和内核中的file相对应，上层通过fd就可以找到file，找到对应的read、write函数

```
fs/inode.c:
void init_special_inode(struct inode *inode, umode_t mode, dev_t rdev)
{
□inode->i_mode = mode;
□if (S_ISCHR(mode)) {
□□inode->i_fop = &def_chr_fops;
□□inode->i_rdev = rdev;
□} else if (S_ISBLK(mode)) {
□□inode->i_fop = &def_blk_fops;
□□inode->i_rdev = rdev;
□} else if (S_ISFIFO(mode))
□□inode->i_fop = &pipefifo_fops;
□else if (S_ISSOCK(mode))
□□;□/* leave it no_open_fops */
□else
□□printk("...\n");
}
```

```
fs/chr_dev.c:
const struct file_operations def_chr_fops = {
□.open = chrdev_open,
□.llseek = noop_llseek,
};
static int chrdev_open(struct inode *inode, struct file *filp)
{
□const struct file_operations *fops;
□struct cdev *p, *new;
□p = inode->i_cdev;
□□struct kobject *kobj;
□□int idx;
□□kobj = kobj_lookup(cdev_map, inode->i_rdev, &idx);
□□new = container_of(kobj, struct cdev, kobj);
□□inode->i_cdev = p = new;
□□list_add(&inode->i_devices, &p->list);
□□new = NULL;
□fops = fops_get(p->ops);
□replace_fops(filp, fops);
□filp->f_op->open(inode, filp);
□return 0;
}
```

```
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 *);
    int (*iterate) (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);
};
```

```
struct fdtable {
□unsigned int max_fds;
□struct file __rcu **fd; /* current fd array */
□unsigned long *close_on_exec;
□unsigned long *open_fds;
□unsigned long *full_fds_bits;
□struct rcu_head rcu;
};
```

```
struct fd {
□struct file *file;
□unsigned int flags;
};
```

```
SYSCALL_DEFINE3(open, const char __user *, filename, int,
flags, umode_t, mode)
{
    if (force_o_largefile())
□flags |= O_LARGEFILE;
    return do_sys_open(AT_FDCWD, filename, flags, mode);
}

long do_sys_open(int dfd, const char __user *filename, int
flags, umode_t mode)
{
    struct open_how how = build_open_how(flags, mode);
    return do_sys_openat2(dfd, filename, &how);
}

do_sys_openat2
    -> do_filp_open->path_openat->do_open->vfs_open
        ->do_dentry_open: file->f_op=inode->i_fop
                          file->f_op->open
    -> fd_install(fd,file)->__fd_install:fdt->fd[fd]
```

```
fs/read_write.c
SYSCALL_DEFINE3(read, unsigned int, fd, char __user *, buf, size_t, count)
{
□return ksys_read(fd, buf, count);
}
ssize_t ksys_read(unsigned int fd, char __user *buf, size_t count)
{
□struct fd f = fdget_pos(fd); //通过描述符fd找到对应的fd结构和file对象
□ssize_t ret = -EBADF;

□if (f.file) {
□□loff_t pos, *ppos = file_ppos(f.file); //获取文件读写位置
□□if (ppos) {
□□□pos = *ppos;
□□□ppos = &pos;
□□}
□□ret = vfs_read(f.file, buf, count, ppos);
□□if (ret >= 0 && ppos)
□□□f.file->f_pos = pos;
□□fdput_pos(f);
□}
□return ret;
}

ssize_t vfs_read(struct file *file, char __user *buf, size_t count, loff_t *pos)
{
□ssize_t ret;

□ret = rw_verify_area(READ, file, pos, count);
□if (count > MAX_RW_COUNT)
□□count =  MAX_RW_COUNT;

□if (file->f_op->read)
□ret = file->f_op->read(file, buf, count, pos);
□else if (file->f_op->read_iter)
□□ret = new_sync_read(file, buf, count, pos);
□else
□□ret = -EINVAL;
□inc_syscr(current);
□return ret;
}
```

mknod：每创建一个文件，文件系统就会分配一个inode，文件与inode一一对应

```
struct inode {    //内核中的所有文件都使用inode来表示
    umode_t□□i_mode;
    unsigned short□□i_opflags;
    kuid_t□□i_uid; // inode拥有者ID
    kgid_t□□i_gid; // inode所属群组ID
    unsigned int□□i_flags          ; //文件类型

    const struct inode_operations□*i_op;
    struct super_block□*i_sb;
    struct address_space□*i_mapping;
    unsigned long□□i_ino;
    union {
□     const unsigned int i_nlink;
□     unsigned int __i_nlink;
□};
    dev_t□□□i_rdev;              // 若是设备文件，表示设备号
    loff_t□□i_size;
    unsigned short          □i_bytes;
    u8□□□i_blkbits;
    u8□□□i_write_hint;
    blkcnt_t□□i_blocks;

    unsigned long□□dirtied_when;□
    unsigned long□□dirtied_time_when;
    union {
□     struct hlist_head□i_dentry;
□     struct rcu_head□i_rcu;
□};
    atomic64_t□□i_version;
    atomic64_t□□i_sequence;
    atomic_t□□i_count;
    atomic_t□□i_dio_count;
    atomic_t□□i_writecount;

    union {
□     const struct file_operations   *i_fop;□
□     void (*free_inode)(struct inode  *);
□};
    struct address_space□i_data;
    struct list_head□i_devices;
    union {
□     struct pipe_inode_info  *i_pipe;
□     struct block_device     *i_bdev;
 struct cdev□       *i_cdev;      //指向对应的cdev
□     char□□       *i_link;
□     unsigned□□       i_dir_seq;
□};
    __u32□□□i_generation;
    void□□□*i_private;
};
```

多个设备可以共用同一个驱动程序.通过字符设备的inode中的i_devices 和cdev中的list组成一个链表

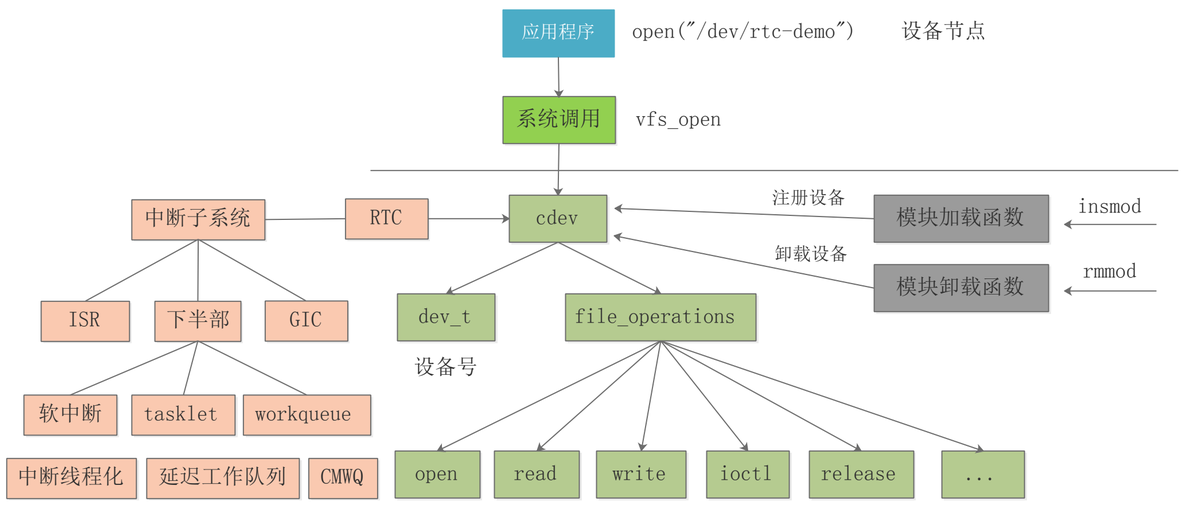
i_cdev为 NULL ，说明该设备文件没有被打开

```
struct file { //用来表示进程中打开的文件
□union {
□□struct llist_node□fu_llist;
□□struct rcu_head □fu_rcuhead;
□} f_u;
□struct path□□f_path;
□struct inode□□*f_inode;□
□const struct file_operations□*f_op;
□spinlock_t□□f_lock;
□enum rw_hint□□f_write_hint;
□atomic_long_t□□f_count;
□unsigned int □□f_flags;
□fmode_t□□□f_mode;
 void□□□*private_data                  ;
□errseq_t□□f_wb_err;
□errseq_t□□f_sb_err; /* for syncfs */
}
```

```
include/linux/fdtable.h
struct files_struct {   // 用户进程打开的文件列表
       atomic_t count;
       bool resize_in_progress;
       wait_queue_head_t resize_wait;

       struct fdtable __rcu *fdt;
       struct fdtable fdtab;
       spinlock_t file_lock ____cacheline_aligned_in_smp;
       unsigned int next_fd;
       unsigned long close_on_exec_init[1];
       unsigned long open_fds_init[1];
       unsigned long full_fds_bits_init[1];
 struct file __rcu * fd_array[NR_OPEN_DEFAULT];
};
```

File Descriptor：fd
fd = open("/dev/rtc-demo0", O_RDWR);



文档说明：
本流程图配套视频教程：《Linux内核编程》05期：字符设备驱动
公众号：宅学部落
作者博客：www.zhaixue.cc
淘宝店：https://wanglitao.taobao.com