ONE_Tech

Next Mile is My Destination 博客新家: sketch2sky.com , 欢迎交流

Linux Platform驱动模型(三) _platform+cdev

平台总线是一种实现设备信息与驱动方法相分离的方法，利用这种方法，我们可以写出一个更像样一点的字符设备驱动，即使用cdev作为接口，平台总线作为分离方式:

```
xjkeydrv_init()：模块加载函数
└──platform_driver_register()将驱动对象模块注册到平台总线
    └──platform_driver.probe()探测函数，提取相应的信息
        └──xjkey_init()：初始化cdev对象，创建设备文件等关于cdev接口创建的工作
            └──cdev_init():将cdev结构与fops绑定到一起，在fops实现操作接口与控制硬件的逻辑
```

```
static struct class *cls = NULL;

static int major = 0;
static int minor = 0;
const  int count = 1;

#define DEVNAME "xjkey"

static struct cdev *xjkeyp = NULL;

static unsigned long irqflags;
static int irq;
```

```
static atomic_t tv;

#if 0
/{
        key@26{
                compatible = "xj4412,key";
                interrupt-parent = <&gpx1>;
                interrupts = <2 2>;
        };
};
#endif

static irqreturn_t handler_t(int irq, void *dev_id)
{
        printk(KERN_INFO "%s : %s : %d\n", __FILE__, __func__, __LINE__);
        return IRQ_HANDLED;
}

//打开设备
static int xjkey_open(struct inode *inode, struct file *filp)
{
        //get major and minor from inode
        printk(KERN_INFO "(major=%d, minor=%d), %s : %s : %d\n",
                imajor(inode), iminor(inode), __FILE__, __func__,
__LINE__);

        if(!atomic_dec_and_test(&tv)){
                atomic_inc(&tv);
                return -EBUSY;
        }

        return request_irq(irq, handler_t, irqflags, DEVNAME, NULL);
}

//关闭设备
```

4. Linux i2c子系统(一) _动手写一个i2c设备驱动(23320)
5. Linux设备管理（一）_kobject, kset,ktype分析(19570)
6. Linux块设备IO子系统(一) _驱动模型(17375)
7. Linux设备管理（二）_从cdev_add说起(16572)
8. 从0移植uboot (一) _配置分析(14214)
9. Linux字符设备驱动框架(13786)
10. 从0移植uboot (二) _uboot启动流程分析(13482)

评论排行榜

1. Linux tcp黏包解决方案(7)
2. Linux设备管理（一）_kobject, kset,ktype分析(6)
3. Linux usb子系统(一) _写一个usb鼠标驱动(2)
4. 跟着内核学框架-从misc子系统到3+2+1设备识别驱动框架(2)
5. Linux驱动技术(八) _并发控制技术(2)
6. Linux驱动技术(四) _异步通知技术(2)
7. Linux设备文件三大结构：inode,file,file_operations(2)
8. Linux 多线程信号量同步(2)
9. 从0移植uboot(六) _实现网络功能(1)
10. Linux input子系统编程、分析与模板(1)

推荐排行榜

1. Linux Platform驱动模型(二) _驱动方法(11)
2. Linux设备树语法详解(11)
3. Linux i2c子系统(一) _动手写一个i2c设备驱动(6)
4. Linux Platform驱动模型(一) _设备信息(6)
5. Linux usb子系统(二) _usb-skeleton.c精析(5)

```
static int xjkey_release(struct inode *inode, struct file *filp)
{
        //get major and minor from irqflagsinode
        printk(KERN_INFO "(major=%d, minor=%d), %s : %s : %d\n",
                imajor(inode), iminor(inode), __FILE__, __func__,
__LINE__);

        free_irq(irq, NULL);

        atomic_inc(&tv);
        return 0;
}

static struct file_operations fops = {
        .owner  = THIS_MODULE,
        .open   = xjkey_open,
        .release= xjkey_release,
};

static int xjkey_init(void)
{
        dev_t devnum;
        int ret, i;

        struct device *devp = NULL;

        //get command and pid
        printk(KERN_INFO "(%s:pid=%d), %s : %s : %d\n",
                current->comm, current->pid, __FILE__, __func__, __LINE__);

        //1. alloc cdev objxjkey_init
        xjkeyp = cdev_alloc();
        if(NULL == xjkeyp){
                return -ENOMEM;
        }
```

```
        //2. init cdev obj
        cdev_init(xjkeyp, &fops);

        ret = alloc_chrdev_region(&devnum, minor, count, DEVNAME);
        if(ret){
                goto ERR_STEP;
        }
        major = MAJOR(devnum);

        //3. register cdev obj
        ret = cdev_add(xjkeyp, devnum, count);
        if(ret){
                goto ERR_STEP1;
        }

        cls = class_create(THIS_MODULE, DEVNAME);
        if(IS_ERR(cls)){
                ret = PTR_ERR(cls);
                goto ERR_STEP1;
        }

        for(i = minor; i < (count+minor); i++){
                devp = device_create(cls, NULL, MKDEV(major, i), NULL,
"%s%d", DEVNAME, i);
                if(IS_ERR(devp)){
                        ret = PTR_ERR(devp);
                        goto ERR_STEP2;
                }
        }

        // init atomic_t
        atomic_set(&tv, 1);

        //get command and pid
        printk(KERN_INFO "(%s:pid=%d), %s : %s : %d - ok.\n",
                current->comm, current->pid, __FILE__, __func__, __LINE__);
```

```
        return 0;

ERR_STEP2:
        for(--i; i >= minor; i--){
                device_destroy(cls, MKDEV(major, i));
        }
        class_destroy(cls);

ERR_STEP1:
        unregister_chrdev_region(devnum, count);

ERR_STEP:
        cdev_del(xjkeyp);

        //get command and pid
        printk(KERN_INFO "(%s:pid=%d), %s : %s : %d - fail.\n",
                current->comm, current->pid, __FILE__, __func__, __LINE__);
        return ret;
}

static void xjkey_exit(void)
{
        int i;
        //get command and pid
        printk(KERN_INFO "(%s:pid=%d), %s : %s : %d - leave.\n",
                current->comm, current->pid, __FILE__, __func__, __LINE__);

        for(i=minor; i < (count+minor); i++){
                device_destroy(cls, MKDEV(major, i));
        }
        class_destroy(cls);

        unregister_chrdev_region(MKDEV(major, minor), count);
        cdev_del(xjkeyp);
}
```

```
static int xjkey_probe(struct platform_device *pdev)
{
        struct resource *irq_res;
        irq_res = platform_get_resource(pdev, IORESOURCE_IRQ, 0);
        if(irq_res){
                irq = irq_res->start;
                irqflags = irq_res->flags & IRQF_TRIGGER_MASK;
        }else{
                printk(KERN_INFO "No 0 irq\n");
                return -EINVAL;
        }

        return xjkey_init();
}

static int xjkey_remove(struct platform_device *pdev)
{
        printk(KERN_INFO "%s : %s : %d - leave.\n", __FILE__, __func__,
__LINE__);
        xjkey_exit();
        return 0;
}

struct of_device_id of_tbl[] = {
        {.compatible = "xj4412,key",},
        {},
};
MODULE_DEVICE_TABLE(of, of_tbl);


//1. alloc obj
static struct platform_driver xjkeydrv = {
        .probe  = xjkey_probe,
        .remove = xjkey_remove,

        .driver = {
```

```
                .name = DEVNAME,
                .of_match_table = of_tbl,
        },
};

//3. register obj
module_platform_driver(xjkeydrv);

MODULE_LICENSE("GPL");
```

分类: Linux驱动开发

标签: 平台总线, 字符设备

Abnor
关注 - 0
粉丝 - 294
+加关注

1 0

« 上一篇: Linux Platform驱动模型(二)_驱动方法
» 下一篇: Linux内核 设备树操作常用API

posted @ 2017-02-06 08:58 Abnor 阅读(2752) 评论(0) 编辑 收藏 举报