platform驱动样例与测试

1.代码

```
#include <linux/module.h>
#include <linux/types.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/init.h>
#include <linux/cdev.h>
#include <linux/slab.h>
#include <linux/uaccess.h>
#include <linux/platform_device.h>


#define GLOBALMEM_SIZE 0x1000       /* 4kb */
#define MEM_CLEAR 0x1
#define GLOBALMEM_MAJOR 230

/* types */
struct globalmem_dev {
    struct cdev cdev;
    unsigned char mem[GLOBALMEM_SIZE];
};

/* datas */
static int globalmem_major = GLOBALMEM_MAJOR;
struct globalmem_dev *globalmem_devp;



/* fops functions definition */
static int globalmem_open(struct inode *inode, struct file *filp)
{
    filp->private_data = globalmem_devp;
    return 0;
}

static ssize_t globalmem_read(struct file *filp, char __user *buf,
                                size_t size, loff_t *ppos)
{
    unsigned long p = *ppos;
    unsigned int count = size;
    int ret = 0;

    struct globalmem_dev *dev = filp->private_data;

    if(p >= GLOBALMEM_SIZE)
        return 0;
    if(count > GLOBALMEM_SIZE - p)
        count = GLOBALMEM_SIZE - p;

    if(copy_to_user(buf,dev->mem + p,count)){
        ret = - EFAULT;
    }else{
        *ppos += count;
        ret = count;
        printk(KERN_INFO "read %u byte(s) from %lu\n", count, p);
    }

    return ret;
}

static ssize_t globalmem_write(struct file *filp, const char __user *buf,
                                size_t size, loff_t *ppos)
{
    unsigned long p = *ppos;
    unsigned int count = size;
    int ret = 0;

    struct globalmem_dev *dev = filp->private_data;

    if(p >= GLOBALMEM_SIZE)
        return 0;

    if(count > GLOBALMEM_SIZE - p)
        count = GLOBALMEM_SIZE - p;

    if(copy_from_user(dev->mem + p, buf, count)){
        ret = - EFAULT;
    }else{
        *ppos += count;
        ret = count;
        printk(KERN_INFO "written %u byte(s) to %lu\n", count,p);
    }

    return ret;
}

static loff_t globalmem_llseek(struct file *filp, loff_t offset, int orig)
{
    loff_t ret;
    switch(orig){
    case 0:     /* from begin of file */
        if(offset < 0) {
            ret = -EINVAL;
            break;
        }
        if((unsigned int)offset >= GLOBALMEM_SIZE){
            ret = -EINVAL;
            break;
        }
        filp->f_pos = offset;
        ret = filp->f_pos;
        break;
    case 1:     /* from the current point */
        if((filp->f_pos + offset) < 0) {
            ret = -EINVAL;
            break;
        }
        if((filp->f_pos + offset) >= GLOBALMEM_SIZE){
            ret = -EINVAL;
            break;
        }
        filp->f_pos += offset;
        ret = filp->f_pos;
        break;
    default:
        ret = -EINVAL;
    }

    return ret;
}

static long globalmem_ioctl(struct file *filp, unsigned int cmd, unsigned long arg)
{
    struct globalmem_dev *dev = filp->private_data;

    switch(cmd){
    case MEM_CLEAR:
        memset(dev->mem,0,GLOBALMEM_SIZE);
        printk(KERN_INFO "globalmem is set to zero\n");
        break;
    default:
        return -EINVAL;
    }

    return 0;
}

static int globalmem_release(struct inode *inode, struct file *filp)
{
    return 0;
}



/* fill in the file_operation structure */
static const struct file_operations globalmem_fops = {
    .owner = THIS_MODULE,
    .llseek = globalmem_llseek,
    .read = globalmem_read,
    .write = globalmem_write,
    .unlocked_ioctl = globalmem_ioctl,
    .open = globalmem_open,
    .release = globalmem_release,
};


static void globalmem_setup_cdev(struct globalmem_dev *dev,int index)
{
    int err;
    dev_t devno = MKDEV(globalmem_major,index);

    cdev_init(&dev->cdev,&globalmem_fops);
    dev->cdev.owner = THIS_MODULE;

    err = cdev_add(&dev->cdev, devno, 1);

    if(err)
        printk(KERN_NOTICE "Error %d adding globalmem %d\n", err, index);
}


/* 当总线匹配了platform_device中的name和platform_driver.driver中的name后调用 */
static int globalmem_probe(struct platform_device *pdev)
{
    printk("insmod platform_1.ko\n");
    int ret = 0;
    dev_t devno = MKDEV(globalmem_major, 0);

    if (globalmem_major)                                            /* 分配设备号 */
        ret = register_chrdev_region(devno, 1, "globalmem");        /* 该名称显示在
/proc/devices */
    else {
        ret = alloc_chrdev_region(&devno, 0, 1, "globalmem");
        globalmem_major = MAJOR(devno);
    }
    if (ret < 0)
        return ret;

    globalmem_devp = kzalloc(sizeof(*globalmem_devp),GFP_KERNEL);
    if (!globalmem_devp) {
        ret = -ENOMEM;
        goto fail_malloc;
    }

    globalmem_setup_cdev(globalmem_devp, 0);

    return 0;

fail_malloc:
    unregister_chrdev_region(devno, 1);
    return ret;
}

/* 在设备或驱动注销时调用 */
```

```c
static int globalmem_remove(struct platform_device *pdev)
{
    printk("rmmod platform_1.ko\n");
    cdev_del(&globalmem_devp->cdev);
    kfree(globalmem_devp);
    unregister_chrdev_region(MKDEV(globalmem_major, 0), 1);
    return 0;
}

static struct platform_driver globalmem_driver = {
    .driver = {
        .name = "globalmem_yeshen",     /* 该名称显示在 /sys/bus/platform/drivers */
        .owner = THIS_MODULE,
    },
    .probe = globalmem_probe,
    .remove = globalmem_remove,
};


static struct platform_device globalmem_device = {
    .name = "globalmem_yeshen",     /* 该名称显示在 /sys/devices/platform 和
/sys/bus/platform/devices */
    .id = -1,
};

static struct platform_device *platform_devices = {
    &globalmem_device,
};



static int __init globalmem_init(void)
{
    int ret;
    ret = platform_driver_register(&globalmem_driver);      /*注册驱动*/
    if(ret){
        printk("platform driver register failed!\n");
        return ret;
    }

    ret = platform_add_devices(&platform_devices,1);        /*批量注册设备,内部调用了
platform_device_register()*/
    if(ret){
        printk("platform register devices failed!\n");
        return ret;
    }

    return ret;
}

static void __exit globalmem_exit(void)
{
    platform_device_unregister(&globalmem_device);      /*注销设备*/
    platform_driver_unregister(&globalmem_driver);      /*注销驱动*/

}

module_param(globalmem_major, int, S_IRUGO);

module_init(globalmem_init);
module_exit(globalmem_exit);

MODULE_AUTHOR("Yeshen 569242715@qq.com");
MODULE_LICENSE("GPL v2");
```

2.说明

这个paltform测试驱动基于字符驱动globalmem作些修改。在加载模块时调用platform_driver_register和platform_add_devices分别注册platform驱动和platform设备，然后当总线匹配了platform_device中的name和platform_driver.driver中的name后调用globalmem_probe，此时是platform中真正的具体设备的注册和初始化。当卸载模块时会注销platform设备和platform驱动，进而回调globalmem_remove作真正的具体设备的释放。