ioctl.c

/\*

 \* ioctl.c

 \*/

#include <linux/cdev.h>

#include <linux/fs.h>

#include <linux/init.h>

#include <linux/ioctl.h>

#include <linux/module.h>

#include <linux/slab.h>

#include <linux/uaccess.h>

struct ioctl\_arg {

    unsigned int val;

};

/\* Documentation/ioctl/ioctl-number.txt \*/

#define IOC\_MAGIC '\x66'

#define IOCTL\_VALSET \_IOW(IOC\_MAGIC, 0, struct ioctl\_arg)

#define IOCTL\_VALGET \_IOR(IOC\_MAGIC, 1, struct ioctl\_arg)

#define IOCTL\_VALGET\_NUM \_IOR(IOC\_MAGIC, 2, int)

#define IOCTL\_VALSET\_NUM \_IOW(IOC\_MAGIC, 3, int)

#define IOCTL\_VAL\_MAXNR 3

#define DRIVER\_NAME "ioctltest"

static unsigned int test\_ioctl\_major = 0;

static unsigned int num\_of\_dev = 1;

static struct cdev test\_ioctl\_cdev;

static int ioctl\_num = 0;

struct test\_ioctl\_data {

    unsigned char val;

    rwlock\_t lock;

};

static long test\_ioctl\_ioctl(struct file \*filp, unsigned int cmd, unsigned long arg)

{

    struct test\_ioctl\_data \*ioctl\_data = filp->private\_data;

    int retval = 0;

    unsigned char val;

    struct ioctl\_arg data;

    memset(&data, 0, sizeof(data));

    switch (cmd) {

    case IOCTL\_VALSET:

        if (copy\_from\_user(&data, (int \_\_user \*)arg, sizeof(data))) {

            retval = -EFAULT;

            goto done;

        }

        pr\_alert("IOCTL set val:%x .\n", data.val);

        write\_lock(&ioctl\_data->lock);

        ioctl\_data->val = data.val;

        write\_unlock(&ioctl\_data->lock);

        break;

    case IOCTL\_VALGET:

        read\_lock(&ioctl\_data->lock);

        val = ioctl\_data->val;

        read\_unlock(&ioctl\_data->lock);

        data.val = val;

        if (copy\_to\_user((int \_\_user \*)arg, &data, sizeof(data))) {

            retval = -EFAULT;

            goto done;

        }

        break;

    case IOCTL\_VALGET\_NUM:

        retval = \_\_put\_user(ioctl\_num, (int \_\_user \*)arg);

        break;

    case IOCTL\_VALSET\_NUM:

        ioctl\_num = arg;

        break;

    default:

        retval = -ENOTTY;

    }

done:

    return retval;

}

static ssize\_t test\_ioctl\_read(struct file \*filp, char \_\_user \*buf, size\_t count, loff\_t \*f\_pos)

{

    struct test\_ioctl\_data \*ioctl\_data = filp->private\_data;

    unsigned char val;

    int retval;

    int i = 0;

    read\_lock(&ioctl\_data->lock);

    val = ioctl\_data->val;

    read\_unlock(&ioctl\_data->lock);

    for (; i < count; i++) {

        if (copy\_to\_user(&buf[i], &val, 1)) {

            retval = -EFAULT;

            goto out;

        }

    }

    retval = count;

out:

    return retval;

}

static int test\_ioctl\_close(struct inode \*inode, struct file \*filp)

{

    pr\_alert("%s call.\n", \_\_func\_\_);

    if (filp->private\_data) {

        kfree(filp->private\_data);

        filp->private\_data = NULL;

    }

    return 0;

}

static int test\_ioctl\_open(struct inode \*inode, struct file \*filp)

{

    struct test\_ioctl\_data \*ioctl\_data;

    pr\_alert("%s call.\n", \_\_func\_\_);

    ioctl\_data = kmalloc(sizeof(struct test\_ioctl\_data), GFP\_KERNEL);

    if (ioctl\_data == NULL)

        return -ENOMEM;

    rwlock\_init(&ioctl\_data->lock);

    ioctl\_data->val = 0xFF;

    filp->private\_data = ioctl\_data;

    return 0;

}

static struct file\_operations fops = {

    .owner = THIS\_MODULE,

    .open = test\_ioctl\_open,

    .release = test\_ioctl\_close,

    .read = test\_ioctl\_read,

    .unlocked\_ioctl = test\_ioctl\_ioctl,

};

static int ioctl\_init(void)

{

    dev\_t dev;

    int alloc\_ret = -1;

    int cdev\_ret = -1;

    alloc\_ret = alloc\_chrdev\_region(&dev, 0, num\_of\_dev, DRIVER\_NAME);

    if (alloc\_ret)

        goto error;

    test\_ioctl\_major = MAJOR(dev);

    cdev\_init(&test\_ioctl\_cdev, &fops);

    cdev\_ret = cdev\_add(&test\_ioctl\_cdev, dev, num\_of\_dev);

    if (cdev\_ret)

        goto error;

    pr\_alert("%s driver(major: %d) installed.\n", DRIVER\_NAME, test\_ioctl\_major);

    return 0;

error:

    if (cdev\_ret == 0)

        cdev\_del(&test\_ioctl\_cdev);

    if (alloc\_ret == 0)

        unregister\_chrdev\_region(dev, num\_of\_dev);

    return -1;

}

static void ioctl\_exit(void)

{

    dev\_t dev = MKDEV(test\_ioctl\_major, 0);

    cdev\_del(&test\_ioctl\_cdev);

    unregister\_chrdev\_region(dev, num\_of\_dev);

    pr\_alert("%s driver removed.\n", DRIVER\_NAME);

}

module\_init(ioctl\_init);

module\_exit(ioctl\_exit);

MODULE\_LICENSE("GPL");

MODULE\_DESCRIPTION("This is test\_ioctl module");

Result (dmesg command)