```c
#include <linux/init.h>
#include <linux/module.h>
#include <linux/fs.h>
#include <linux/cdev.h>
#include <linux/device.h>
#include <linux/slab.h>     // kmalloc(), kfree()
#include <asm/uaccess.h>    // copy_to_user()

MODULE_AUTHOR("Jakub Werner");
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("A dummy driver");
MODULE_SUPPORTED_DEVICE("none");

#define MAJORNUM 116
#define NUMDEVICES 2
#define DEVNAME "t12hello"

static struct cdev *cdev = NULL;

static int is_open = 0;

static struct class *dev_class;
static struct device *device;

static atomic_t v;

static ssize_t driver_read(struct file *instanz, char *buf, size_t num,
                loff_t * off);
static ssize_t driver_open(struct inode *inode, struct file *file);
static ssize_t driver_close(struct inode *inode, struct file *file);

static struct file_operations fops = {
    .owner = THIS_MODULE,
    .read = driver_read,
    .open = driver_open,
    .release = driver_close
};

static int __init mod_init(void)
{
    dev_t major_nummer = MKDEV(MAJORNUM, 0);

    atomic_set(&v, -1);
    printk(KERN_ALERT "Hello, world\n");

    if (register_chrdev_region(MKDEV(MAJORNUM, 0), NUMDEVICES, DEVNAME)) {
        pr_warn("Device number 0x%x not available ...\n",
            MKDEV(MAJORNUM, 0));
        return -EIO;
    }

    pr_info("Device number 0x%x created\n", MKDEV(MAJORNUM, 0));

    cdev = cdev_alloc();
    if (cdev == NULL) {
        pr_warn("cdev_alloc failed!\n");
        goto free_devnum;
    }

    kobject_set_name(&cdev->kobj, DEVNAME);
    cdev->owner = THIS_MODULE;
    cdev_init(cdev, &fops);

    if (cdev_add(cdev, MKDEV(MAJORNUM, 0), NUMDEVICES)) {
        pr_warn("cdev_add failed!\n");
        goto free_cdev;
    }

    device = device;

    dev_class = class_create(THIS_MODULE, DEVNAME);
    device = device_create(dev_class, NULL, major_nummer, NULL, DEVNAME);
```

```
    return 0;

 free_cdev:
    kobject_put(&cdev->kobj);
    cdev = NULL;
 free_devnum:
    unregister_chrdev_region(MKDEV(MAJORNUM, 0), NUMDEVICES);
    return -1;
}

static ssize_t driver_read(struct file *instanz, char *userbuf, size_t count,
               loff_t * off)
{

    char *string = "Hello World!!!\n";

    ssize_t num = 0;
    ssize_t len = strlen(string);
    num = len - copy_to_user(userbuf, string, len);
    pr_debug("Module fops : sent %d bytes to user space \n", num);
    return num;

}

static ssize_t driver_open(struct inode *inode, struct file *file)
{
    if (is_open)
        return -EBUSY;

    is_open++;

    try_module_get(THIS_MODULE);
    pr_debug
        ("Module fops:device %s was opened from device with minor no %d \n",
         DEVNAME, iminor(inode));
    return 0;

}

static ssize_t driver_close(struct inode *inode, struct file *file)
{
    is_open--;

    module_put(THIS_MODULE);
    pr_debug("Module fops:device %s was closed \n", DEVNAME);
    return 0;

}

static void __exit mod_exit(void)
{

    if (cdev) {
        cdev_del(cdev);
    }

    device_destroy(dev_class, MKDEV(MAJORNUM, 0));
    class_destroy(dev_class);

    unregister_chrdev_region(MKDEV(MAJORNUM, 0), NUMDEVICES);
    printk(KERN_ALERT "Goodbye, cruel world\n");
}

module_init(mod_init);
module_exit(mod_exit);
```

```c
#include <linux/init.h>
#include <linux/module.h>
#include <linux/fs.h>
#include <linux/cdev.h>
#include <linux/device.h>
#include <linux/slab.h>     // kmalloc(), kfree()
#include <asm/uaccess.h>    // copy_to_user()

MODULE_AUTHOR("Jakub Werner");
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("A dummy driver");
MODULE_SUPPORTED_DEVICE("none");

#define MAJORNUM 115
#define NUMDEVICES 2
#define DEVNAME "t12null"

static struct cdev *cdev = NULL;

static int is_open = 0;
static atomic_t v;

static struct class *dev_class;
static struct device *device;

static ssize_t driver_write(struct file *instanz, const char __user * userbuf,
                size_t num, loff_t * off);
static ssize_t driver_open(struct inode *inode, struct file *file);
static ssize_t driver_close(struct inode *inode, struct file *file);

static struct file_operations fops = {
    .owner = THIS_MODULE,
    .write = driver_write,
    .open = driver_open,
    .release = driver_close
};

static int __init mod_init(void)
{
    dev_t major_nummer = MKDEV(MAJORNUM, 0);

    atomic_set(&v, -1);
    printk(KERN_ALERT "Hello, world\n");

    if (register_chrdev_region(MKDEV(MAJORNUM, 0), NUMDEVICES, DEVNAME)) {
        pr_warn("Device number 0x%x not available ...\n",
            MKDEV(MAJORNUM, 0));
        return -EIO;
    }

    pr_info("Device number 0x%x created\n", MKDEV(MAJORNUM, 0));

    cdev = cdev_alloc();
    if (cdev == NULL) {
        pr_warn("cdev_alloc failed!\n");
        goto free_devnum;
    }

    kobject_set_name(&cdev->kobj, DEVNAME);
    cdev->owner = THIS_MODULE;
    cdev_init(cdev, &fops);

    if (cdev_add(cdev, MKDEV(MAJORNUM, 0), NUMDEVICES)) {
        pr_warn("cdev_add failed!\n");
        goto free_cdev;
    }

    dev_class = class_create(THIS_MODULE, DEVNAME);
    device = device_create(dev_class, NULL, major_nummer, NULL, DEVNAME);

    return 0;

 free_cdev:
```

```
    kobject_put(&cdev->kobj);
    cdev = NULL;
 free_devnum:
    unregister_chrdev_region(MKDEV(MAJORNUM, 0), NUMDEVICES);
    return -1;
}

static ssize_t driver_write(struct file *instanz, const char __user * userbuf,
                size_t count, loff_t * off)
{

    pr_debug("writing count = %d", count);
    return count;
}

static ssize_t driver_open(struct inode *inode, struct file *file)
{
    if (is_open)
        return -EBUSY;

    is_open++;

    try_module_get(THIS_MODULE);
    pr_debug
        ("Module fops:device %s was opened from device with minor no %d \n",
         DEVNAME, iminor(inode));
    return 0;

}

static ssize_t driver_close(struct inode *inode, struct file *file)
{
    is_open--;

    module_put(THIS_MODULE);
    pr_debug("Module fops:device %s was closed \n", DEVNAME);
    return 0;

}

static void __exit mod_exit(void)
{

    if (cdev) {
        cdev_del(cdev);
    }

    device_destroy(dev_class, MKDEV(MAJORNUM, 0));
    class_destroy(dev_class);

    unregister_chrdev_region(MKDEV(MAJORNUM, 0), NUMDEVICES);
    printk(KERN_ALERT "Goodbye, cruel world\n");
}

module_init(mod_init);
module_exit(mod_exit);
```

```c
#include <linux/init.h>
#include <linux/module.h>
#include <linux/fs.h>       //struct file_operations
#include <linux/atomic.h>
#include <linux/cdev.h>
#include <linux/device.h>   // class_create, device_create

// Metainformation
MODULE_AUTHOR("Stefano Di Martno");
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("A dummy driver");
MODULE_SUPPORTED_DEVICE("none");

#define MAJORNUM 117
#define NUMDEVICES 1
#define DEVNAME "t12openclose"

static int open_count = 0;
static atomic_t v;
static struct cdev *cdev = NULL;
static struct class *dev_class;

static void __exit mod_exit(void)
{
    printk(KERN_ALERT "Goodbye, cruel world\n");
    device_destroy(dev_class, MKDEV(MAJORNUM, 0));
    class_destroy(dev_class);

    if (cdev) {
        cdev_del(cdev);
    }

    unregister_chrdev_region(MKDEV(MAJORNUM, 0), NUMDEVICES);
}

static int driver_open(struct inode *inode, struct file *instance)
{
    printk("open() called!\n");

    if (open_count == 0) {
        open_count++;
        pr_debug("open_count: device is locked by process!\n");
    } else {
        pr_debug
            ("open_count: device already locked by another process!\n");
        pr_debug("open_count: %d process is accessing this file\n",
             open_count);
    }

    if (atomic_inc_and_test(&v)) {
        pr_debug("atomic_inc_and_test: device is locked by process!\n");
    } else {
        pr_debug
            ("atomic_inc_and_test: device already locked by another process!\n");
        pr_debug
            ("atomic_inc_and_test: %d process is accessing this file\n",
             open_count);
        atomic_dec_and_test(&v);

        return -EBUSY;
    }

    return 0;
}

static ssize_t driver_write(struct file *instanz, const char __user * userbuf,
                size_t count, loff_t * off)
{

    pr_debug("writing count = %d\n", count);
    return count;
}
```

```
static ssize_t driver_read(struct file *file, char *user, size_t count,
                loff_t * offset)
{
    return 0;       //EOF
}

static int driver_close(struct inode *inode, struct file *instance)
{
    printk("close() called\n");

    open_count--;

    pr_debug("open_count: %d pending processes\n", open_count);

    if (atomic_dec_and_test(&v)) {
        pr_debug("atomic_dec_and_test: %d pending processes\n",
             open_count);
    }

    return 0;
}

static struct file_operations fops = {
    .owner = THIS_MODULE,
    .read = driver_read,
    .write = driver_write,
    .open = driver_open,
    .release = driver_close,
};

static int __init mod_init(void)
{
    dev_t major_nummer = MKDEV(MAJORNUM, 0);

    printk(KERN_ALERT "Hello, world\n");

    atomic_set(&v, -1);

    if (register_chrdev_region(major_nummer, NUMDEVICES, DEVNAME)) {
        pr_warn("Device number 0x%x not available ...\n",
            MKDEV(MAJORNUM, 0));
        return -EIO;
    }

    pr_info("Device number 0x%x created\n", MKDEV(MAJORNUM, 0));

    cdev = cdev_alloc();
    if (cdev == NULL) {
        pr_warn("cdev_alloc failed!\n");
        goto free_devnum;
    }

    kobject_set_name(&cdev->kobj, DEVNAME);
    cdev->owner = THIS_MODULE;
    cdev_init(cdev, &fops);

    if (cdev_add(cdev, MKDEV(MAJORNUM, 0), NUMDEVICES)) {
        pr_warn("cdev_add failed!\n");
        goto free_cdev;
    }

    dev_class = class_create(THIS_MODULE, DEVNAME);
    device_create(dev_class, NULL, major_nummer, NULL, DEVNAME);

    return 0;

 free_cdev:
    kobject_put(&cdev->kobj);
    cdev = NULL;
 free_devnum:
    unregister_chrdev_region(MKDEV(MAJORNUM, 0), NUMDEVICES);
    return -1;
}
```

```
module_init(mod_init);
module_exit(mod_exit);
```

```c
#include <linux/init.h>
#include <linux/module.h>
#include <linux/fs.h>
#include <linux/cdev.h>
#include <linux/slab.h>     // kmalloc(), kfree()
#include <asm/uaccess.h>    // copy_to_user()
#include <linux/device.h>   // class_create, device_create

// Metainformation
MODULE_AUTHOR("Stefano Di Martno");
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("A dummy driver");
MODULE_SUPPORTED_DEVICE("none");

#define MAJORNUM 113
#define NUMDEVICES 2
#define DEVNAME_0 "t12zero_0"
#define DEVNAME_1 "t12zero_1"

static struct cdev *cdev = NULL;

static char hello_world[] = "Hello World\n";
struct class *dev_class[2];

static void __exit mod_exit(void)
{
    printk(KERN_ALERT "Goodbye, cruel world\n");

    device_destroy(dev_class[0], MKDEV(MAJORNUM, 0));
    class_destroy(dev_class[0]);

    device_destroy(dev_class[1], MKDEV(MAJORNUM, 1));
    class_destroy(dev_class[1]);

    if (cdev) {
        cdev_del(cdev);
    }

    unregister_chrdev_region(MKDEV(MAJORNUM, 0), NUMDEVICES);
}

static int driver_open(struct inode *inode, struct file *instance)
{
    int minor = iminor(inode);

    instance->private_data = kmalloc(sizeof(int), GFP_KERNEL);

    if (instance->private_data == NULL) {
        pr_alert("Could not allocate memory!\n");
        return -1;
    }
    *((int *)(instance->private_data)) = minor;

    return 0;
}

static ssize_t driver_read(struct file *instance, char *user, size_t count,
               loff_t * offset)
{
    long not_copied, to_copy, minor, sent, copied;
    char *data;

    minor = *((int *)(instance->private_data));

    if (minor == 0) {
        to_copy = 1;
        data = "0";
    } else if (minor == 1) {
        to_copy = strlen(hello_world);
        data = hello_world;
    } else
        return -1;  // TODO richtige Fehlermeldung!
```

```
    if (to_copy > count)
        to_copy = count;

    not_copied = copy_to_user(user, data, to_copy);

    sent = to_copy - not_copied;
    copied = to_copy - not_copied;

    pr_debug
        ("Module  zero: sent %ld bytes to user space \nNot copied: %ld bytes\n",
         sent, not_copied);

    pr_debug("Copied: %ld\n", copied);

    return copied;
}

static int driver_close(struct inode *inode, struct file *instance)
{
    kfree(instance->private_data);
    printk("close() called\n");

    return 0;
}

static struct file_operations fops = {
    .owner = THIS_MODULE,
    .read = driver_read,
    .open = driver_open,
    .release = driver_close,
};

static void create_device(dev_t dev_number, char *dev_name, int index)
{
    dev_class[index] = class_create(THIS_MODULE, dev_name);
    device_create(dev_class[index], NULL, dev_number, NULL, dev_name);
}

static int __init mod_init(void)
{
    dev_t number_min_0 = MKDEV(MAJORNUM, 0);
    dev_t number_min_1 = MKDEV(MAJORNUM, 1);

    printk(KERN_ALERT "Hello, world\n");

    if (register_chrdev_region(number_min_0, NUMDEVICES, DEVNAME_0)) {
        pr_warn("Device number 0x%x not available ...\n",
            MKDEV(MAJORNUM, 0));
        return -EIO;
    }

    pr_info("Device number 0x%x created\n", MKDEV(MAJORNUM, 0));

    cdev = cdev_alloc();
    if (cdev == NULL) {
        pr_warn("cdev_alloc failed!\n");
        goto free_devnum;
    }

    kobject_set_name(&cdev->kobj, DEVNAME_0);
    cdev->owner = THIS_MODULE;
    cdev_init(cdev, &fops);

    if (cdev_add(cdev, MKDEV(MAJORNUM, 0), NUMDEVICES)) {
        pr_warn("cdev_add failed!\n");
        goto free_cdev;
    }

    create_device(number_min_0, DEVNAME_0, 0);
    create_device(number_min_1, DEVNAME_1, 1);

    return 0;
```

```
 free_cdev:
    kobject_put(&cdev->kobj);
    cdev = NULL;
 free_devnum:
    unregister_chrdev_region(MKDEV(MAJORNUM, 0), NUMDEVICES);
    return -1;
}

module_init(mod_init);
module_exit(mod_exit);
```