Operating System (CSC 3150)

Tutorial 10

KAI SHEN

SCHOOL OF SCIENCE AND ENGINEERING

E-MAIL: 118010254@LINK.CUHK.EDU.CN

Target

In this tutorial, we will discuss how to make a device in Linux and Assignment 5 related functions.

- Create file system node
- Allocate device number
- Initialize and remove a CDEV
- File operations
- Test device
- Work Routine
- Blocking and Non-blocking I/O
- DMA buffer

Device Driver(Textbook Chap.18.8)

- Linux Device Drivers include character devices (such as printers, terminals, and mice), block devices (including all disk drives), and network interface devices.

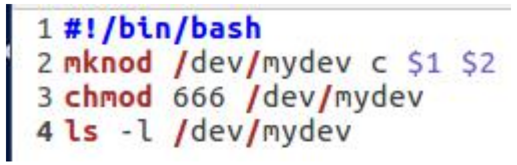
- Block devices include all devices that allow random access to completely independent, fixed-sized blocks of data, including hard disks and floppy disks, CD-ROMs and Blu-ray discs, and flash memory. Block devices are typically used to store file systems, but direct access to a block device is also allowed so that programs can create and repair the file system that the device contains. Applications can also access these block devices directly if they wish. For example, a database application may prefer to perform its own fine-tuned layout of data onto a disk rather than using the general-purpose file system.

- Character devices include most other devices, such as mice and keyboards. The fundamental difference between block and character devices is random access—block devices are accessed randomly, while character devices are accessed serially byte by byte. For example, seeking to a certain position in a file might be supported for a DVD but makes no sense for a pointing device such as a mouse.

- Network devices are dealt with differently from block and character devices. Users cannot directly transfer data to network devices. Instead, they must communicate indirectly by opening a connection to the kernel’s networking subsystem.

Create file system node

- command "mknod"
  - makes a directory entry and corresponding i-node for a special file.
- mknod Name b/c Major Minor
  - Name: device name
  - b: indicates the special file is a block-oriented device (disk, diskette, or tape)
  - c: indicates the special file is a character-oriented device (other devices)
  - Major: major device, which helps the operating system find the device driver code.
  - Minor: minor device, that is the unit drive or line number
- A script to use mknod is provided (mknod.sh)
  - Make a device named "mydev" under /dev

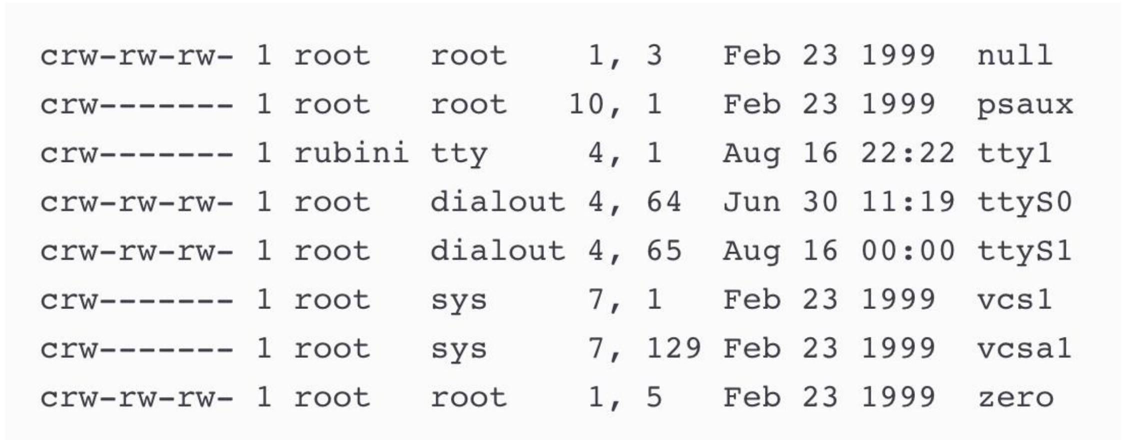


```
#!/bin/bash
mknod /dev/mydev c $1 $2
chmod 666 /dev/mydev
ls -l /dev/mydev
```

Create file system node

- major numbers: 1, 4, 7, 10,
- minors numbers: 1, 3, 5, 64, 65, 129.
- The major number identifies the driver associated with the device.
- The minor number is used only by the driver specified by the major number, which provides a way for the driver to differentiate among them.

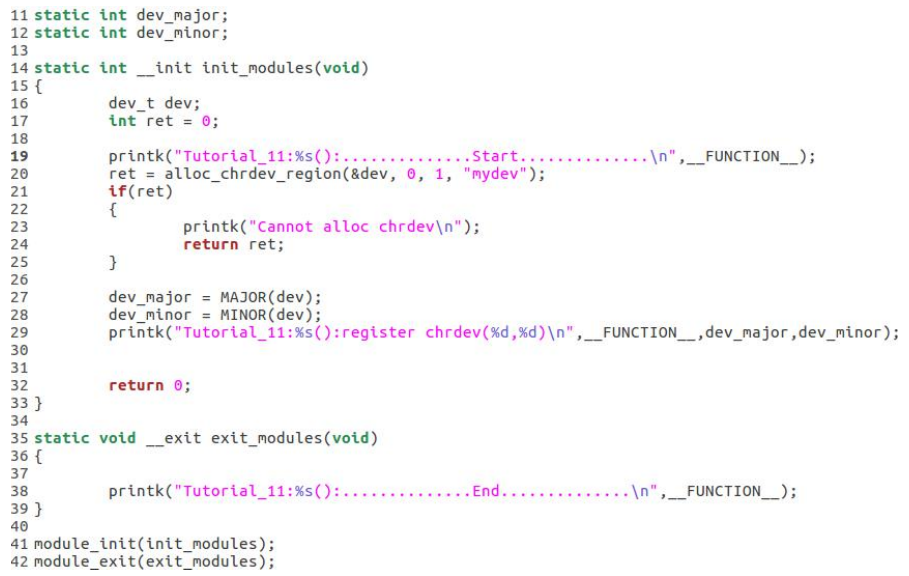
`ls –l output screenshot`



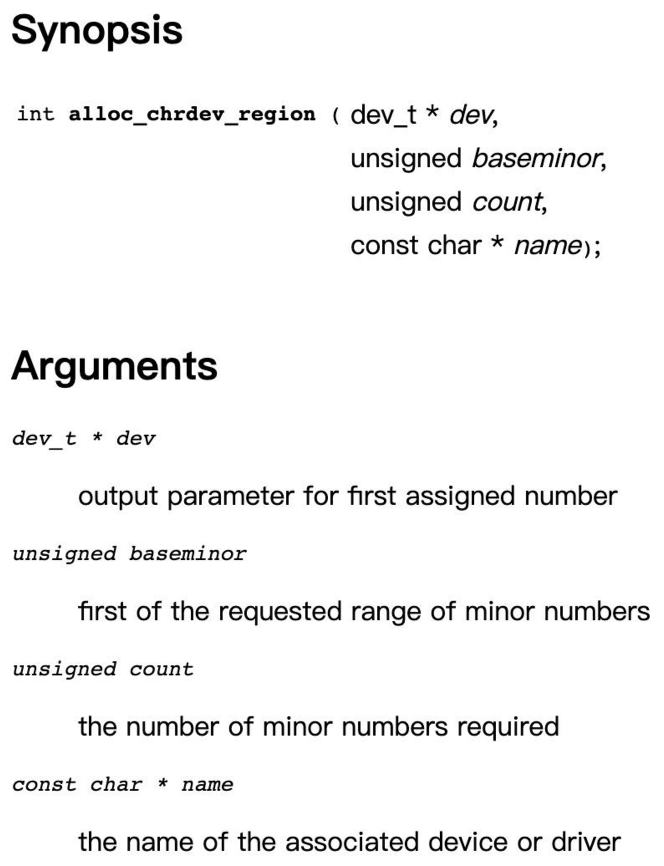
```
crw-rw-rw- 1 root    root     1, 3   Feb 23 1999  null
crw------- 1 root    root    10, 1   Feb 23 1999  psaux
crw------- 1 rubini  tty      4, 1   Aug 16 22:22 tty1
crw-rw-rw- 1 root    dialout  4, 64  Jun 30 11:19 ttyS0
crw-rw-rw- 1 root    dialout  4, 65  Aug 16 00:00 ttyS1
crw------- 1 root    sys      7, 1   Feb 23 1999  vcs1
crw------- 1 root    sys      7, 129 Feb 23 1999  vcsa1
crw-rw-rw- 1 root    root     1, 5   Feb 23 1999  zero
```

Create file system node

- You can get available major and minor number by:
  - Create a kernel module
  - Use alloc_chrdev_region() in module_init() function
  - Call MAJOR() and MINOR().



```c
static int dev_major;
static int dev_minor;

static int __init init_modules(void)
{
        dev_t dev;
        int ret = 0;

        printk("Tutorial_11:%s():..............Start..............\n",__FUNCTION__);
        ret = alloc_chrdev_region(&dev, 0, 1, "mydev");
        if(ret)
        {
                printk("Cannot alloc chrdev\n");
                return ret;
        }

        dev_major = MAJOR(dev);
        dev_minor = MINOR(dev);
        printk("Tutorial_11:%s():register chrdev(%d,%d)\n",__FUNCTION__,dev_major,dev_minor);


        return 0;
}

static void __exit exit_modules(void)
{

        printk("Tutorial_11:%s():..............End..............\n",__FUNCTION__);
}

module_init(init_modules);
module_exit(exit_modules);
```



Synopsis

```
int alloc_chrdev_region ( dev_t * dev,
                          unsigned baseminor,
                          unsigned count,
                          const char * name);
```

Arguments

*dev_t * dev*

output parameter for first assigned number

unsigned baseminor

first of the requested range of minor numbers

unsigned count

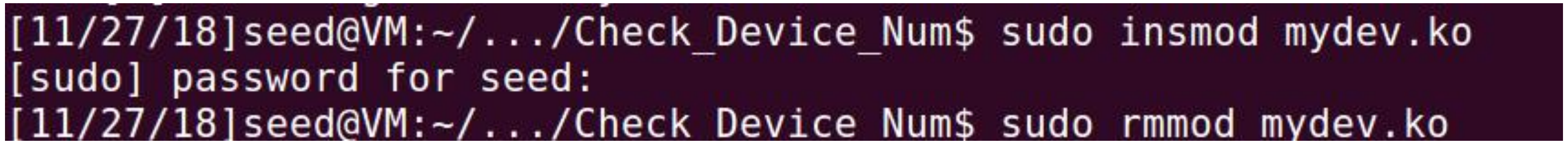
the number of minor numbers required

*const char * name*

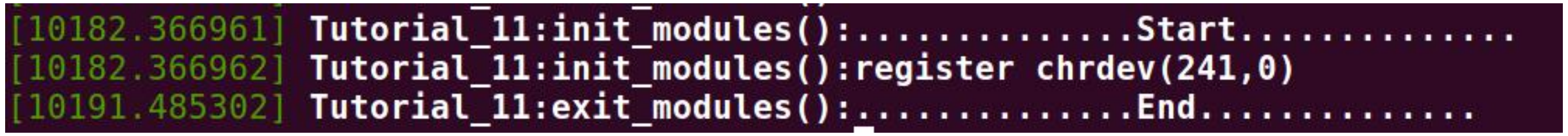
the name of the associated device or driver

Create file system node

- Insmod mydev module to check available major and minor number.

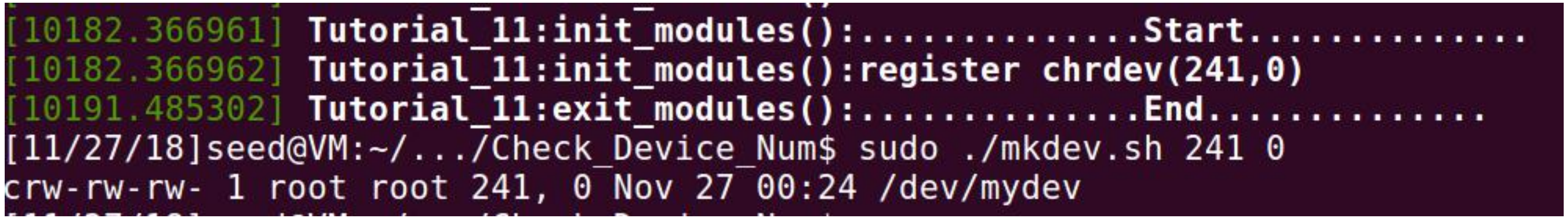


```
[11/27/18]seed@VM:~/.../Check_Device_Num$ sudo insmod mydev.ko
[sudo] password for seed:
[11/27/18]seed@VM:~/.../Check_Device_Num$ sudo rmmod mydev.ko
```



```
[10182.366961] Tutorial_11:init_modules():..............Start..............
[10182.366962] Tutorial_11:init_modules():register chrdev(241,0)
[10191.485302] Tutorial_11:exit_modules():..............End..............
```

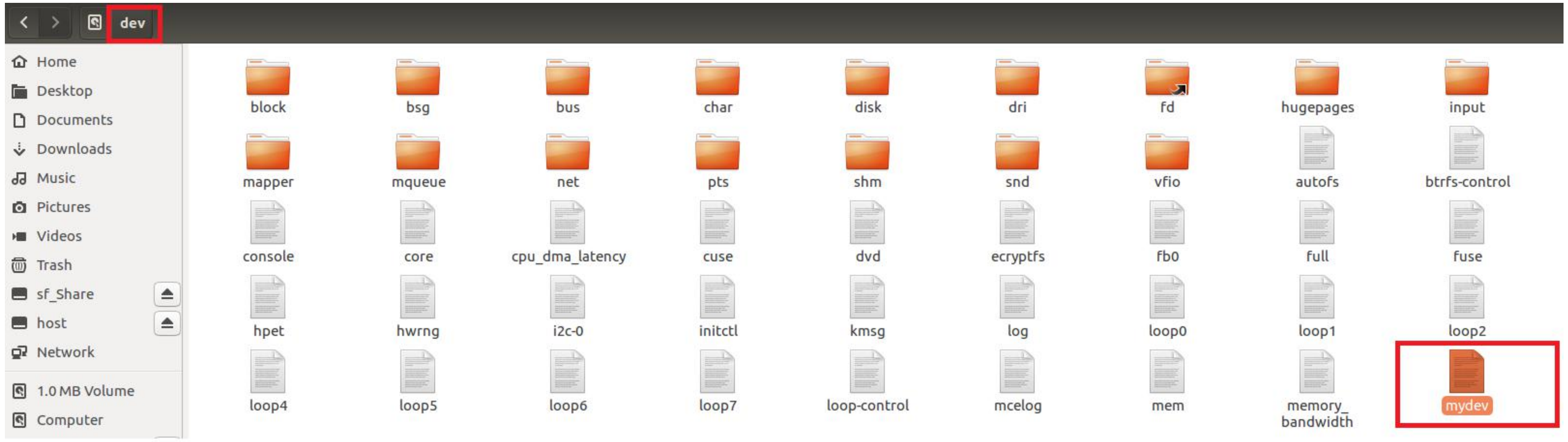
- Run “mkdev.sh” to create directory for mydev



```
[10182.366961] Tutorial_11:init_modules():..............Start..............
[10182.366962] Tutorial_11:init_modules():register chrdev(241,0)
[10191.485302] Tutorial_11:exit_modules():..............End..............
[11/27/18]seed@VM:~/.../Check_Device_Num$ sudo ./mkdev.sh 241 0
crw-rw-rw- 1 root root 241, 0 Nov 27 00:24 /dev/mydev
```

Create file system node

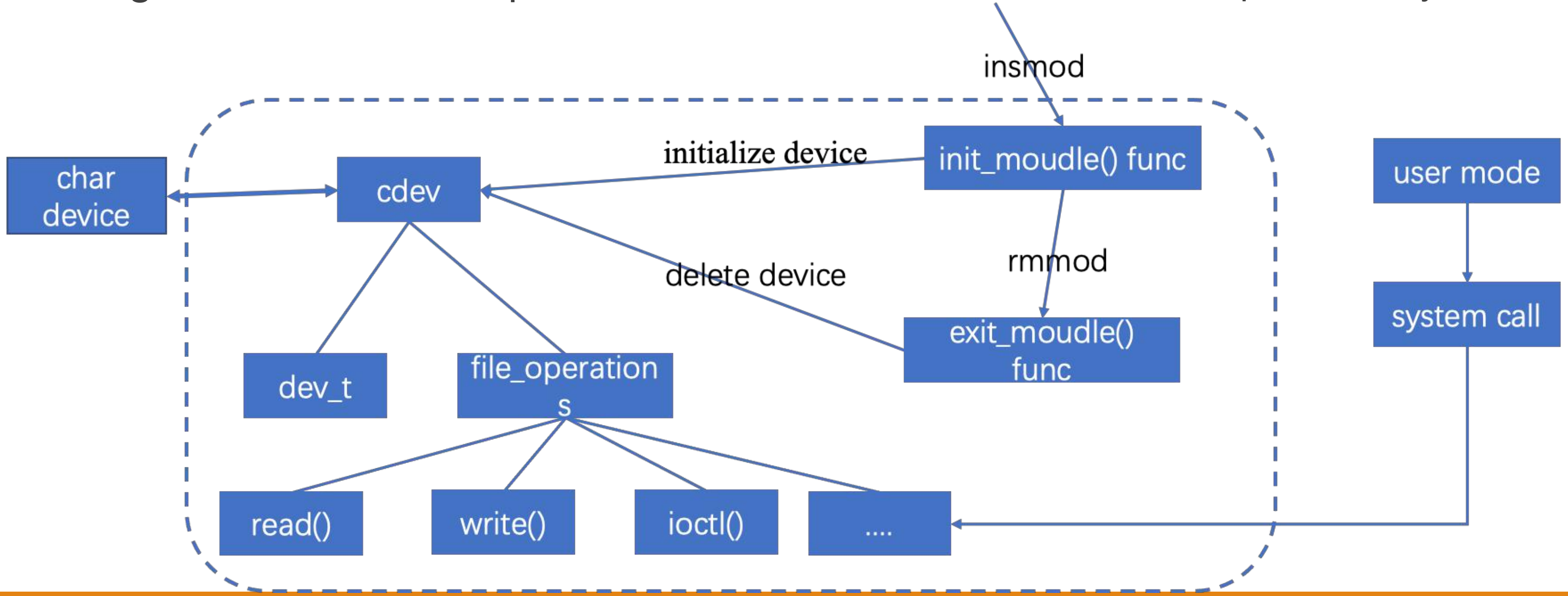
- Under “/dev”, you will find “mydev”



```
block    bsg      bus              char     disk          dri       fd     hugepages         input
mapper   mqueue   net              pts      shm           snd       vfio   autofs            btrfs-control
console  core     cpu_dma_latency  cuse     dvd           ecryptfs  fb0    full              fuse
hpet     hwrng    i2c-0            initctl  kmsg          log       loop0  loop1             loop2
loop4    loop5    loop6            loop7    loop-control  mcelog    mem    memory_bandwidth  mydev
```

Before we continue

- Essentially, writing the Char Driver is to define the cdev structure:
- Through its member dev_t to define the device number (divided into major and minor device numbers) to determine the uniqueness of the character device
- Through its member file_operations to define the interface functions provided by the character device



Register range of device number

- alloc_chrdev_region(dev_t * dev,
  unsigned baseminor,
  unsigned count,
  const char * name)
  - Allocates a range of char device numbers.
  - The major number will be chosen dynamically.
  - It returns (along with the first minor number) in dev. Returns zero or a negative error code.
  - dev: output parameter for first assigned number
  - baseminor: first of the requested range of minor numbers
  - count: the number of minor numbers required
  - name: the name of the associated device or driver

Unregister range of device number

- unregister_chrdev_region (dev_t from,
  unsigned count)
  - This function will unregister a range of count device numbers, starting with from.
  - The caller should normally be the one who allocated those numbers in the first place.
  - from: the first in the range of numbers to unregister
  - count: the number of device numbers to unregister

Initialize and remove a CDEV

- cdev_alloc()
  - Allocate a cdev structure
  - Allocates and returns a cdev structure, or NULL on failure
  - cdev structure is defined as: struct cdev *dev_cdevp;

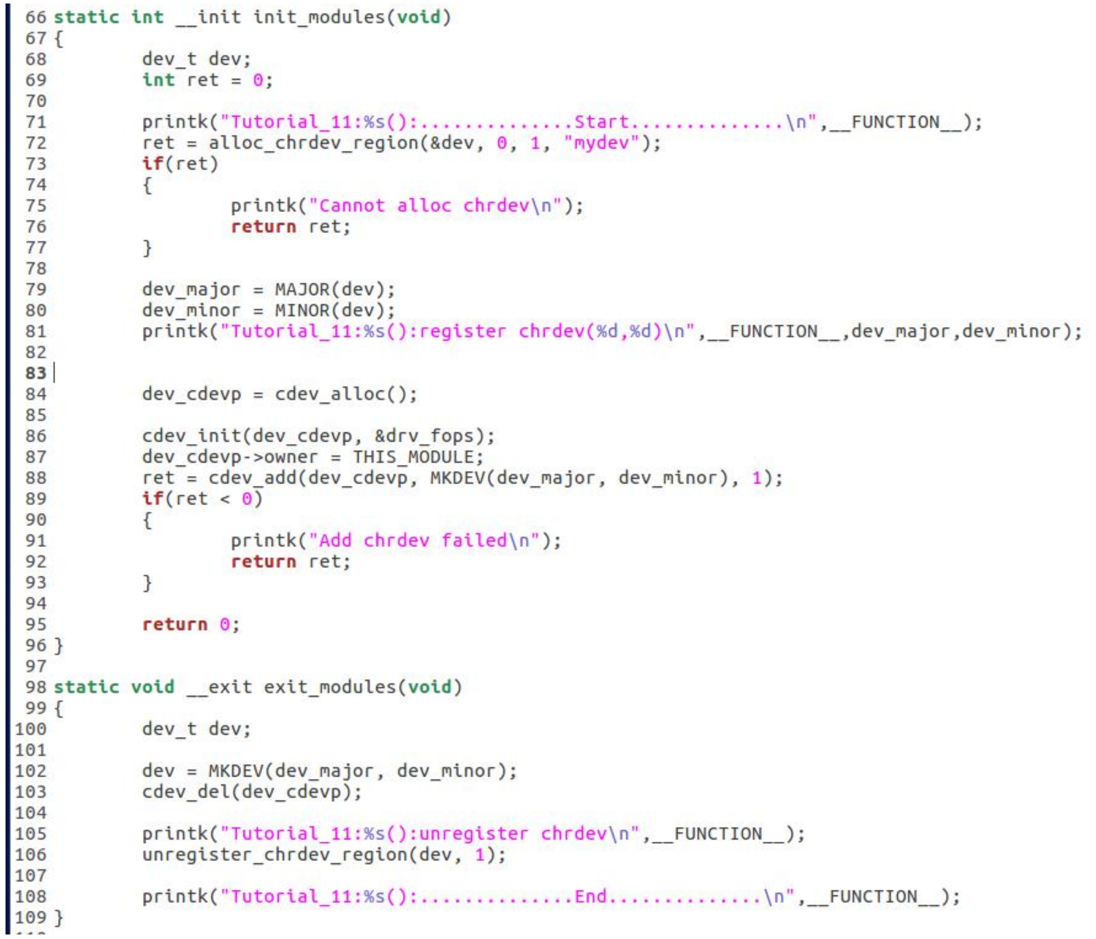
- cdev_init(struct cdev * cdev, const struct file_operations * fops)
  - Initializes cdev, remembering fops, making it ready to add to the system with cdev_add
  - cdev: the structure to initialize
  - fops: the file_operations for this device

Initialize and remove a CDEV

- cdev_add(struct cdev * p, dev_t dev, unsigned count)
  - adds the device represented by p to the system, making it live immediately.
  - A negative error code is returned on failure.
  - p: the cdev structure for the device.
  - dev: the first device number for which this device is responsible.
  - count: the number of consecutive minor numbers corresponding to this device.

- cdev_del(struct cdev * p)
  - removes p from the system, possibly freeing the structure itself
  - p: the cdev structure to be removed

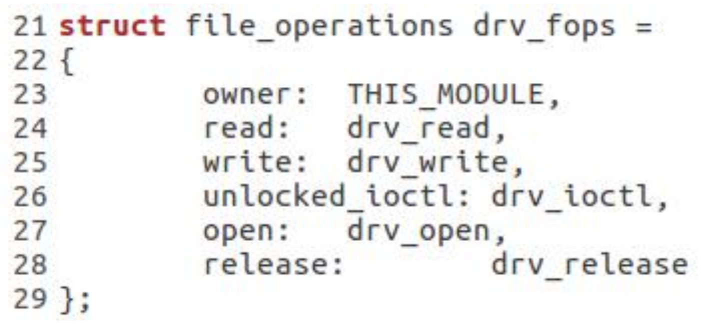
Initialize and remove a CDEV



```
static int __init init_modules(void)
{
        dev_t dev;
        int ret = 0;

        printk("Tutorial_11:%s():..............Start..............\n",__FUNCTION__);
        ret = alloc_chrdev_region(&dev, 0, 1, "mydev");
        if(ret)
        {
                printk("Cannot alloc chrdev\n");
                return ret;
        }

        dev_major = MAJOR(dev);
        dev_minor = MINOR(dev);
        printk("Tutorial_11:%s():register chrdev(%d,%d)\n",__FUNCTION__,dev_major,dev_minor);


        dev_cdevp = cdev_alloc();

        cdev_init(dev_cdevp, &drv_fops);
        dev_cdevp->owner = THIS_MODULE;
        ret = cdev_add(dev_cdevp, MKDEV(dev_major, dev_minor), 1);
        if(ret < 0)
        {
                printk("Add chrdev failed\n");
                return ret;
        }

        return 0;
}

static void __exit exit_modules(void)
{
        dev_t dev;

        dev = MKDEV(dev_major, dev_minor);
        cdev_del(dev_cdevp);

        printk("Tutorial_11:%s():unregister chrdev\n",__FUNCTION__);
        unregister_chrdev_region(dev, 1);

        printk("Tutorial_11:%s():..............End..............\n",__FUNCTION__);
}
```

File operations

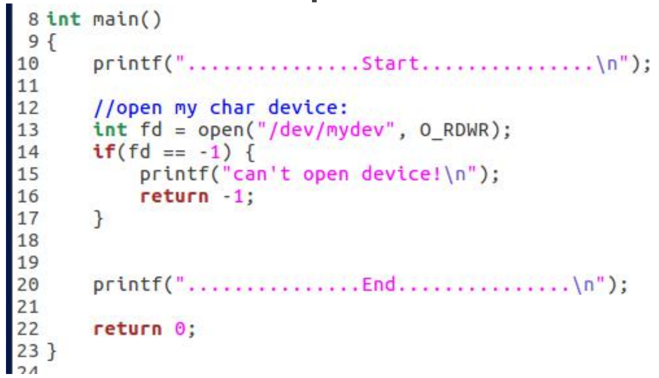
- In Linux, control device just likes R/W file.
- You should write a struct file_ operation to map the operations to functions in this module.
- And use cdev_init() at module init to bind cdev and file_ operations.
- File operations are defined in fs.h
    - https://elixir.bootlin.com/linux/latest/source/include/linux/fs.h#L1730



```
struct file_operations drv_fops =
{
        owner:  THIS_MODULE,
        read:   drv_read,
        write:  drv_write,
        unlocked_ioctl: drv_ioctl,
        open:   drv_open,
        release:        drv_release
};
```

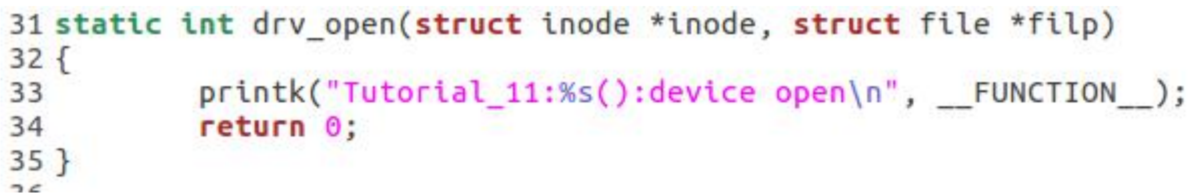
File operations

- In user mode, open a file



```
int main()
{
    printf("...............Start...............\n");

    //open my char device:
    int fd = open("/dev/mydev", O_RDWR);
    if(fd == -1) {
        printf("can't open device!\n");
        return -1;
    }


    printf("...............End...............\n");

    return 0;
}
```

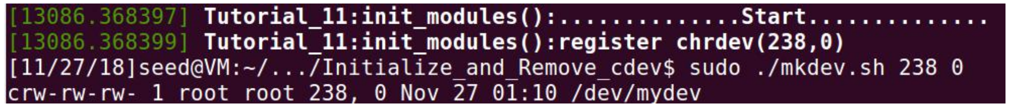
- In kernel device, it maps to “drv_open”



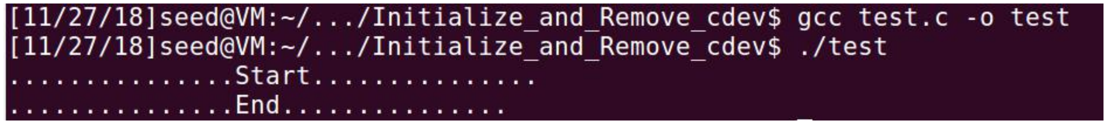
```
static int drv_open(struct inode *inode, struct file *filp)
{
        printk("Tutorial_11:%s():device open\n", __FUNCTION__);
        return 0;
}
```

Test device

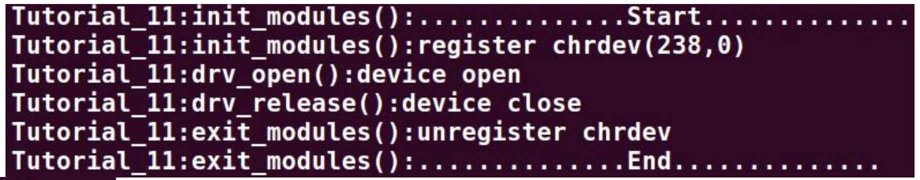
- Test the file operations in your device (./Initialize_and_Remove_cdev)
  - Run make file to build mydev.ko: make
  - Insert your device module: sudo insmod mydev.ko
  - Check the major and minor number: dmesg
  - Run mknod script to create file system node : sudo sh ./mkdev.sh Major Minor
  - Build test executable file: gcc test.c -o test
  - Run test: ./test
  - Remove your device module: sudo rmmod mydev.ko
  - Check messages in kernel log: dmesg
  - Remove filesystem node: sudo sh ./rmdev.sh



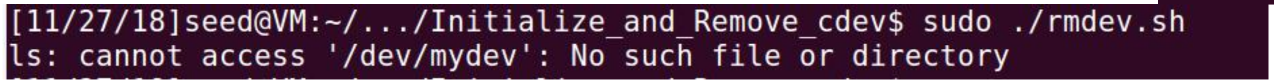
```
[13086.368397] Tutorial_11:init_modules():.............Start.............
[13086.368399] Tutorial_11:init_modules():register chrdev(238,0)
[11/27/18]seed@VM:~/.../Initialize_and_Remove_cdev$ sudo ./mkdev.sh 238 0
crw-rw-rw- 1 root root 238, 0 Nov 27 01:10 /dev/mydev
```



```
[11/27/18]seed@VM:~/.../Initialize_and_Remove_cdev$ gcc test.c -o test
[11/27/18]seed@VM:~/.../Initialize_and_Remove_cdev$ ./test
...............Start..............
...............End..............
```



```
Tutorial_11:init_modules():.............Start.............
Tutorial_11:init_modules():register chrdev(238,0)
Tutorial_11:drv_open():device open
Tutorial_11:drv_release():device close
Tutorial_11:exit_modules():unregister chrdev
Tutorial_11:exit_modules():.............End.............
```



```
[11/27/18]seed@VM:~/.../Initialize_and_Remove_cdev$ sudo ./rmdev.sh
ls: cannot access '/dev/mydev': No such file or directory
```

Work Routine

- In Linux, the work could be written in a work routine function in module.
- Use INIT_WORK() and schedule_work() to queue work to system queue.
- work: struct work_struct *work
- INIT_WORK(work, func): initialize work (from an allocated buffer).
- schedule_work(work): put work task in global workqueue
- flush_scheduled_work(): flush work on work queue. It is to flush the kernel-global work queue. Forces execution of the kernel-global workqueue and blocks until its completion.

Blocking and Non-blocking I/O

- Your write function in module can be blocking or non-blocking.
- Blocking write need to wait computation completed.
- Non-blocking write just return after queueing work.
- Use work routine to control blocking and non-blocking write.

DMA buffer

- Direct Memory Access
- To simulate register and memory on device, you could kmalloc a dma buffer.
- This buffer is as I/O port mapping in main memory.
- What device do is written in work routine function . This function get data from this buffer.
- You need to implement in & out function to access dma buffer just like physical device.
- out function is used to output data to dma buffer.
- in function is used to input data from dma buffer.

DMA buffer

- kzalloc(size_t size, gfp_t flags)
  - allocate memory. The memory is set to zero.
  - size: how many bytes of memory are required.
  - flag: the type of memory to allocate (GFP_KERNEL - Allocate normal kernel ram.)
  - For flags, more details could be checked on: https://www.kernel.org/doc/htmldocs/kernel-api/API-kmalloc.html

- kfree(const void * objp)
  - free previously allocated memory
  - objp: pointer returned by kmalloc / kzalloc
  - If objp is NULL, no operation is performed.

Reference

- Char devices
  - https://www.kernel.org/doc/htmldocs/kernel-api/chrdev.html
  - https://www.oreilly.com/library/view/linux-device-drivers/0596000081/ch03s02.html
  - https://www.programmersought.com/article/10174645171/
  - Operating-System-Concepts Chapter 18.8

- Work queues
  - https://www.ibm.com/developerworks/library/l-tasklets/index.html
  - https://linuxtv.org/downloads/v4l-dvb-internals/device-drivers/ch01s06.html

Thank you