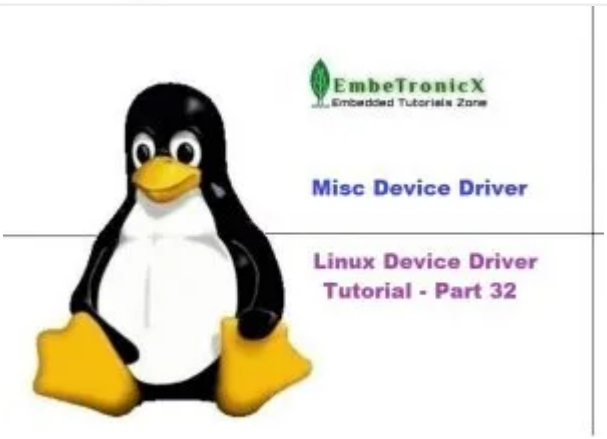
EmbeTronicX
Embedded Tutorials Zone

Sidebar

Device Drivers



Linux Device Driver Tutorial Part 32 – Misc Device Driver

This is the Series on Linux Device Driver. The aim of this series is to provide easy and practical examples that anyone can understand. This is the Linux Device Driver Tutorial Part 32 – Misc device driver in the Linux

X

Post Contents [hide]

reference. Please go through the below tutorial before we began.

- Dummy Linux Character device driver tutorial

Introduction – Misc Device Driver

Misc driver is the miscellaneous driver for miscellaneous devices. We can say that misc drivers are special and simple character drivers. You can write this misc driver when you cannot classify your peripheral. This means, if you don't want to use the major number, then you can write this misc driver. And also if you want to write a simple driver, then you can choose a misc driver instead of choosing a character driver.

So, you cannot choose the major number yourself when you write the misc driver. The default major number of all the misc drivers is **10**. But you can choose your minor numbers between 1 to 255. It has all the file operation calls like `open`, `read`, `write`, `close`, and `IOCTL`. This will create the device file under `/dev/{your_misc_file}`. It is almost like the character driver. Isn't it? Then why the hell we are using the misc

misc driver

- In misc driver, the major number will be 10 and the minor number is user convenient. Whereas, in character drivers, the user can select their own major and minor number if it is available.
- The device node or device file will be automatically generated in misc drivers. Whereas, in character drivers, the user has to create the device node or device file using **`cdev_init`**, **`cdev_add`**, **`class_create`**, and **`device_create`**.

Uses of Misc drivers

- If you write character drivers for simple devices, you have to create a major number as well. In such a case, the kernel has to keep that details in the static table. Which means, you end up wasting the RAM for simple devices. If you write multiple character drivers for multiple simple devices, then the RAM wastage also increases. To avoid this you can use the misc driver. Because, in the misc driver, you don't need to specify the major number since it has a fixed major number which is 10.
- If you use misc drivers, it will automatically create the device file compare to the character driver.

I hope you are okay to go to programming now.

Misc Device Driver API

In order to create misc drivers, we need to use **`miscdevice`** structure, file operations. After that, we need to register the device. Once we have done with the operation, then we can unregister the device. The following APIs are used to create and delete the misc device. You need to insert the header file **`include<linux/miscdevice.h>`**.

Misc device structure

Character drivers have **`cdev`**structure to know the driver's functions and other calls. Like that, the misc driver also has a separate structure to maintain the details. The structure as follows,

```
6 };
```

Where,

<**minor**>: You can assign your custom minor number to this. If you pass **MISC_DYNAMIC_MINOR** to this variable, then the misc driver will automatically generate the minor number and assign it to this variable. Every misc driver needs to have a different minor number since this is the only link between the device file and the driver. Recommended is a dynamic method to allocate the minor number instead of assigning the same minor number to the different misc devices. If you want to select your own minor number then check for the used minor number of all misc devices using **ls -l /dev/**, then you can hardcode your minor number if it is available.

<**name**>: you can assign the name of the misc driver. The device file will be created in this name and displayed under the /dev directory.

<**fops**>: This is the pointer to the file operations. This is the same file operation in character drivers.

<**next**> and <**prev**>: These are used to manage the circular linked list of misc drivers.

Once you loaded the misc driver, then you can see the major and minor number of your misc device driver using **ls -l /dev/{misc_driver_name}**. Please see the example below.

Register the misc device

This API is used to register the misc device with the kernel.

```
1 int misc_register(struct miscdevice * misc)
```

where,

X

failure.

You have to call this function in **__init** function. The structure passed is linked into the kernel and may not be destroyed until it has been unregistered.

This one function will avoid the below functions used in the character device driver while registering the device.

1. **alloc_chrdev_region();** – used for the major and minor number
2. **cdev_init();** – used to initialize **cdev**
3. **cdev_add();** – used to add the **cdev** structure to the device
4. **class_create();** – used to create a class
5. **device_create();** – used to create a device

That's why we are telling this is the simple method.

Example

```
static const struct file_operations fops = {
    .owner          = THIS_MODULE,
    .write          = etx_misc_write,
    .read           = etx_misc_read,
    .open           = etx_misc_open,
    .release        = etx_misc_close,
    .llseek         = no_llseek,
};

struct miscdevice etx_misc_device = {
    .minor = MISC_DYNAMIC_MINOR,
    .name = "simple_etx_misc",
    .fops = &fops,
};

static int __init misc_init(void)
{
    int error;

    error = misc_register(&etx_misc_device);
    if (error) {
        pr_err("misc_register failed!!!\n");
        return error;
    }

    pr_info("misc_register init done!!!\n");
```

X

Unregister the misc device

This API is used to un-register the misc device with the kernel.

```
misc_deregister(struct miscdevice * misc)
```

where,

<misc>: device structure to be un-registered

Unregister a miscellaneous device that was previously successfully registered with **misc_register()**. This has to be called in **__exit** function.

This one function will avoid the below functions used in the character device driver while un-registering the device.

1. **cdev_del();** – used to delete **cdev**
2. **unregister_chrdev_region();** – used to remove the major and minor number
3. **device_destroy();** – used to delete the device
4. **class_destroy();** – used to delete the class

Example

```
1 static void __exit misc_exit(void)
2 {
3     misc_deregister(&etx_misc_device);
4     pr_info("misc_register exit done1!!!\n");
5 }
```

Woohoo. That’s all. Just two APIs are needed to create the misc driver. Is it really simple compared to the character driver? Let’s jump to programing.

1

X

the code.

Driver Source Code

```
/***************************************************************************//**
*  \file       mis_driver.c
*
*  \details    Simple misc driver explanation
*
*  \author     EmbeTronicX
*
*  \Tested with kernel 5.3.0-42-generic
*
*******************************************************************************/
#include <linux/miscdevice.h>
#include <linux/fs.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/init.h>

static int etx_misc_open(struct inode *inode, struct file *file)
{
    pr_info("EtX misc device open\n");
    return 0;
}

static int etx_misc_close(struct inode *inodep, struct file *filp)
{
    pr_info("EtX misc device close\n");
    return 0;
}

static ssize_t etx_misc_write(struct file *file, const char __user *buf,
                size_t len, loff_t *ppos)
{
    pr_info("EtX misc device write\n");

    /* We are not doing anything with this data now */

    return len;
}


static ssize_t etx_misc_read(struct file *filp, char __user *buf,
                    size_t count, loff_t *f_pos)
{
    pr_info("EtX misc device read\n");

    return 0;
}

static const struct file_operations fops = {
    .owner          = THIS_MODULE,
    .write          = etx_misc_write
```

```
57 struct miscdevice etx_misc_device = {
58     .minor = MISC_DYNAMIC_MINOR,
59     .name = "simple_etx_misc",
60     .fops = &fops,
61 };
62
63 static int __init misc_init(void)
64 {
65     int error;
66
67     error = misc_register(&etx_misc_device);
68     if (error) {
69         pr_err("misc_register failed!!!\n");
70         return error;
71     }
72
73     pr_info("misc_register init done!!!\n");
74     return 0;
75 }
76
77 static void __exit misc_exit(void)
78 {
79     misc_deregister(&etx_misc_device);
80     pr_info("misc_register exit done!!!\n");
81 }
82
83 module_init(misc_init)
84 module_exit(misc_exit)
85
86 MODULE_LICENSE("GPL");
87 MODULE_AUTHOR("EmbeTronicX <embetronicx@gmail.com>");
88 MODULE_DESCRIPTION("A simple device driver - Misc Driver");
89 MODULE_VERSION("1.29");
```

Makefile

```
1 obj-m += misc_driver.o
2
3 KDIR = /lib/modules/$(shell uname -r)/build
4
5 all:
6         make -C $(KDIR)  M=$(shell pwd) modules
7
8 clean:
9         make -C $(KDIR)  M=$(shell pwd) clean
```

Execution

- Build the driver by using Makefile (**sudo make**)
- Load the driver using **sudo insmod misc_driver.ko**
- To check the major and minor number, use **ls -l**

X

- Here, 10 is the major number and 53 is the minor number.
- Now we will see write and read. Enter **sudo su** and enter your password if it asks to give permission.
- Do **echo 1 > /dev/simple_etx_misc**

Echo will open the driver and write 1 into the driver and finally close the driver. So if I do **echo** to our driver, it should call the open, write, and release (close) functions. Just check it out.

```
root@embetronicx:/home/slr/Desktop/LDD# echo 1 > /dev/simple_etx_misc
```

- Now Check using **dmesg**

```
root@embetronicx:/home/slr/Desktop/LDD# dmesg
[ 3947.029591] misc_register init done!!!
[ 4047.651860] EtX misc device open
[ 4047.651896] EtX misc device write
[ 4047.651901] EtX misc device close
```

- That’s cool. Now we will verify the read function. Do **cat > /dev/simple_etx_misc**

Cat command will open the driver, read the driver, and close the driver. So if I do **cat** to our driver, it should call the open, read, and release (close) functions. Just check.

```
root@embetronicx:/home/slr/Desktop/LDD# dmesg
[ 4400.914717] EtX misc device open
[ 4400.914734] EtX misc device read
[ 4400.914749] EtX misc device close
```

- Unload the driver using **`sudo rmmod misc_driver`**

Instead of doing **`echo`** and **`cat`** command in the terminal you can also use **`open(), read(), write(), close()`** system calls from user-space applications.

This is just a dummy misc driver. But you can implement your own logic in **`read`**, **`write`**, and **`IOCTL`** too like we did in our previous Linux device driver tutorials.

In our next tutorial, we will see the USB device drivers in Linux.

Character Device Driver
In "Device Drivers"

In "Device Drivers"

structure and File Operations
In "Device Drivers"

1

X

Join the discussion

This site uses Akismet to reduce spam. Learn how your comment data is processed.

1 COMMENT Oldest

Kathir

May 11, 2020 11:00 AM

Thanks for this tutorial. I was waiting for this for a long time. Thank you embetronicx.

Loading...

0 Reply

1

X