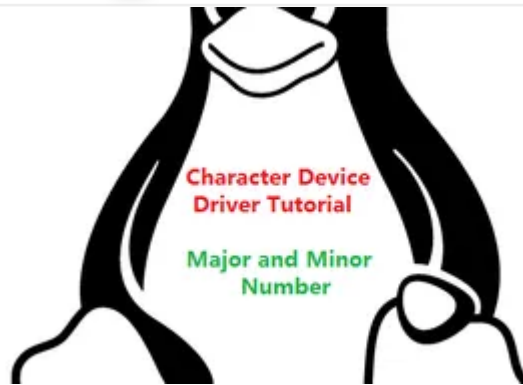


[Sidebar](#) ▼

[Home](#) → [Tutorials](#) → [Linux](#) → [Device Drivers](#) → **Linux Device Driver Tutorial Part 4 - Character Device Driver**

Device Drivers



Linux Device Driver Tutorial Part 4 - Character Device Driver

This is the [Series on Linux Device Driver](#). The aim of this series is to provide the easy and practical examples of Linux Device Drivers that anyone can understand easily. Now we are going to see Linux Device Driver Tutorial Part 4 - Character Device Driver Major Number and Minor Number.

5

Apple Music Turns 5 as It Continues Rivalry With Spotify

Post Contents [\[hide\]](#)

- 1 Introduction
- 2 How Applications will communicate Hardware devices?
- 3 Character Device Driver Major Number and Minor Number
- 4 Major Number and Minor Number
 - 4.1 Major number
 - 4.2 Minor Number
- 5 Allocating Major and Minor Number
 - 5.1 Statically allocating
 - 5.2 Dynamically Allocating
 - 5.3 Difference between static and dynamic method
- 6 Unregister the Major and Minor Number
- 7 Program for Statically Allocating Major Number
- 8 Program for Dynamically Allocating Major Number
 - 8.0.1 Share this:
 - 8.0.2 Like this:
 - 8.0.3 Related

5

Introduction

We already know what drivers are, and why we need them. What is so special about character drivers? If we write drivers for byte-oriented operations then we refer to them as character drivers. Since the majority of devices are byte-oriented, the majority of device drivers are character

device drivers. Take, for example, serial drivers, audio drivers, video drivers, camera drivers, and basic I/O drivers. In fact, all device drivers that are neither storage nor network device drivers are some type of character driver.

How Applications will communicate Hardware devices?

This below diagram will show the full path of communication.

5

- Initially, Application will open the device file. This device file is created by the device driver).
- Then this device file will find the corresponding device driver using

major and minor numbers.

- Then that Device driver will talk to Hardware device.

Character Device Driver Major Number and Minor Number

One of the basic features of the Linux kernel is that it abstracts the handling of devices. All hardware devices look like regular files; they can be opened, closed, read, and written using the same, standard, system calls that are used to manipulate files. To Linux, everything is a file. To write to the hard disk, you write to a file. To read from the keyboard is to read from a file. To store backups on a tape device is to write to a file. Even to read from memory is to read from a file. If the file from which you are trying to read or to which you are trying to write is a “normal” file, the process is fairly easy to understand: the file is opened and you read or write data. So device driver also likes the file. The driver will create a special file for every hardware device. We can communicate with the hardware using those special files (device files).

If you want to create a special file, we should know about the major number and minor number in the device driver. In this tutorial, we will learn that major and minor number.

Major Number and Minor Number

The Linux kernel represents character and block devices as pairs of numbers `<major>:<minor>`.

Major number

Traditionally, the major number identifies the driver associated with the device. A major number can also be shared by multiple device drivers. See `/proc/devices` to find out how major numbers are assigned on a running Linux instance.

```
linux@embetronicx-VirtualBox:~/project/devicedriver/devicefile$ cat /proc/devices
```

```
Character devices:
```

```
1 mem
4 /dev/vc/0
4 tty
```

```
Block devices:
```

```
1 ramdisk
```

259 blkext

These numbers are major numbers.

Minor Number

The major number is to identify the corresponding driver. Many devices may use the same major number. So we need to assign the number to each device which is using the same major number. So this is the minor number. In other words, The device driver uses the minor number **<minor>** to distinguish individual physical or logical devices.

Allocating Major and Minor Number

We can allocate the major and minor numbers in two ways.

1. Statically allocating
2. Dynamically Allocating

Statically allocating

If you want to set a particular major number for your driver, you can use this method. This method will allocate that major number if it is available. Otherwise, it won't.

```
int register_chrdev_region(dev_t first, unsigned int count,  
                           char *name);
```

Here, **first** is the beginning device number of the range you would like to allocate.

count is the total number of contiguous device numbers you are requesting. Note that, if the count is large, the range you request could spill over to the next major number; but everything will still work properly as long as the number range you request is available.

name is the name of the device that should be associated with this number range; it will appear in /proc/devices and sysfs.

The return value from **register_chrdev_region** will be 0 if the allocation was successfully performed. In case of error, a negative error code will

be returned, and you will not have access to the requested region.

The `dev_t` type (defined in `<linux/types.h>`) is used to hold device numbers—both the major and minor parts. `dev_t` is a 32-bit quantity with 12 bits set aside for the major number and 20 for the minor number.

If you want to create the `dev_t` structure variable for your major and minor number, please use the below function.

```
MKDEV(int major, int minor);
```

If you want to get your major number and minor number from `dev_t`, use the below method.

```
MAJOR(dev_t dev);
```

```
MINOR(dev_t dev);
```

If you pass the `dev_t` structure to this MAJOR or MINOR function, it will return that major/minor number of your driver.

Example,

```
1 dev_t dev = MKDEV(235, 0);  
2  
3 register_chrdev_region(dev, 1, "Embetronicx_Dev");
```

Dynamically Allocating

If we don't want fixed major and minor number please use this method. This method will allocate the major number dynamically to your driver which is available.

5

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

`dev` is an output-only parameter that will, on successful completion, hold the first number in your allocated range.

firstminor should be the requested first minor number to use; it is usually 0.

count is the total number of contiguous device numbers you are requesting.

name is the name of the device that should be associated with this number range; it will appear in */proc/devices* and *sysfs*.

Difference between static and dynamic method

A static method is only really useful if you know in advance which major number you want to start with. With the Static method, you are telling the kernel that what device numbers you want (the start major/minor number and count) and it either gives them to you or not (depending on availability).

With Dynamic method, you are telling the kernel that how many device numbers you need (the starting minor number and count) and it will find a starting major number for you, if one is available, of course.

Partially to avoid conflict with other device drivers, it's considered preferable to use the Dynamic method function, which will dynamically allocate the device numbers for you.

The disadvantage of dynamic assignment is that you can't create the device nodes in advance, because the major number assigned to your module will vary. For normal use of the driver, this is hardly a problem, because once the number has been assigned, you can read it from */proc/devices*.

Unregister the Major and Minor Number

Regardless of how you allocate your device numbers, you should free them when they are no longer in use. Device numbers are freed with:

```
void unregister_chrdev_region(dev_t first, unsigned int
count);
```

The usual place to call ***unregister_chrdev_region*** would be in your module's cleanup function (Exit Function).

Program for Statically Allocating Major Number

In this program, I'm assigning 235 as a major number.

```
1 #include<linux/kernel.h>
2 #include<linux/init.h>
3 #include<linux/module.h>
4 #include <linux/fs.h>
5
6 dev_t dev = MKDEV(235, 0);
7 static int __init hello_world_init(void)
8 {
9     register_chrdev_region(dev, 1, "Embetrionicx_Dev");
10    printk(KERN_INFO "Major = %d Minor = %d \n", MAJOR(dev), MINOR(dev));
11    printk(KERN_INFO "Kernel Module Inserted Successfully...\n");
12    return 0;
13 }
14
15 void __exit hello_world_exit(void)
16 {
17    unregister_chrdev_region(dev, 1);
18    printk(KERN_INFO "Kernel Module Removed Successfully...\n");
19 }
20
21 module_init(hello_world_init);
22 module_exit(hello_world_exit);
23
24 MODULE_LICENSE("GPL");
25 MODULE_AUTHOR("EmbeTronicX <embetronicx@gmail.com or admin@embetronicx.com>");
26 MODULE_DESCRIPTION("A simple hello world driver");
27 MODULE_VERSION("1.0");
```

- Build the driver by using Makefile (***sudo make***)
- Load the driver using ***sudo insmod***
- Check the major number using ***cat /proc/devices***

5

```
linux@embetronicx-VirtualBox:~/home/driver/driver$ cat /proc/devices | grep "Embetrionicx
235 Embetrionicx_Dev
```

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

Program for Dynamically Allocating Major Number

This program will allocate a major number dynamically.

```
1  #include<linux/kernel.h>
2  #include<linux/init.h>
3  #include<linux/module.h>
4  #include <linux/kdev_t.h>
5  #include <linux/fs.h>
6
7  dev_t dev = 0;
8
9  static int __init hello_world_init(void)
10 {
11     /*Allocating Major number*/
12     if((alloc_chrdev_region(&dev, 0, 1, "Embetrionicx_Dev")) <0){
13         printk(KERN_INFO "Cannot allocate major number for device 1\n");
14         return -1;
15     }
16     printk(KERN_INFO "Major = %d Minor = %d \n",MAJOR(dev), MINOR(dev));
17     printk(KERN_INFO "Kernel Module Inserted Successfully...\n");
18     return 0;
19 }
20
21 void __exit hello_world_exit(void)
22 {
23     unregister_chrdev_region(dev, 1);
24     printk(KERN_INFO "Kernel Module Removed Successfully...\n");
25 }
26
27 module_init(hello_world_init);
28 module_exit(hello_world_exit);
29
30 MODULE_LICENSE("GPL");
31 MODULE_AUTHOR("EmbeTronicX <embetronicx@gmail.com or admin@embetronicx.com>");
32 MODULE_DESCRIPTION("A simple hello world driver");
33 MODULE_VERSION("1.1");
```

- Build the driver by using Makefile (***sudo make***)
- Load the driver using ***sudo insmod***
- Check the major number using ***cat /proc/devices***

```
linux@embetronicx-VirtualBox::/home/driver/driver$ cat /proc/devices | grep "Embetronicx"
243 Embetronicx_Dev
```

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

This function allocates a major number of 243 for this driver.

Before unloading the driver just check the files in ***/dev*** directory using ***ls /dev/***. You won't find our driver file. Because we haven't created yet. In our [next tutorial](#), we will see that device file.

I hope it helped you. If you have any doubts about this please comment below.

4.5

Article Rating

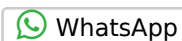


Share this:



Post

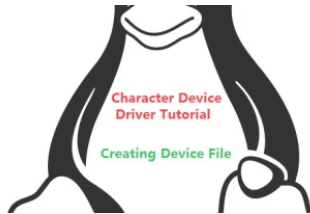
Tweet



Share

Like this:

Loading...

Related

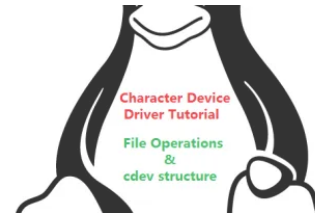
[Linux Device Driver
Tutorial Part 5 - Device
File Creation](#)

In "Device Drivers"



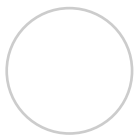
[Linux Device Driver
Tutorial Part 32 - Misc
Device Driver](#)

In "Device Drivers"



[Linux Device Driver
Tutorial Part 6 - Cdev
structure and File
Operations](#)

In "Device Drivers"

☒ Subscribe ▼Connect with | [Login](#)*Join the discussion***B** *I* U 

This site uses Akismet to reduce spam. [Learn how your comment data is processed](#).

5 COMMENTS

Oldest ▼

**saiteja**

October 10, 2017 6:07 AM

nice tutorial

5

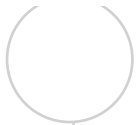
Loading...



0



Reply



Himanshu Gupta

November 9, 2017 10:58 PM

Clearly explained. Very well done. It will be gr8 for us if you can make tutorials on PCI driver as well.

Loading...

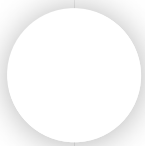


0



Reply

5



KOTHAPALLY AJAY KUMAR

May 11, 2020 6:38 PM

I am getting following error by adding MODULE_AUTHOR and MODULE_LICENSE.

```
make -C /lib/modules/4.4.0-144-generic/build M=/home/ajay/lsp/ldd modules
```

```
make[1]: Entering directory /usr/src/linux-headers-4.4.0-144-generic'
```

```
CC [M] /home/ajay/lsp/ldd/statically.o
```

```
In file included from include/linux/module.h:18:0,
from /home/ajay/lsp/ldd/statically.c:2:
include/linux/moduleparam.h:21:1: error: expected ',' or ';' before 'static'
```

```
static const char __UNIQUE_ID(name)[] \
^
```

```
include/linux/module.h:171:32: note: in expansion of macro '__MODULE_INFO'
```

```
#define MODULE_INFO(tag, info) __MODULE_INFO(tag, tag, info)
^~~~~~
```

```
include/linux/module.h:218:42: note: in expansion of macro 'MODULE_INFO'
```

```
#define MODULE_DESCRIPTION(description)
MODULE_INFO(description, _description)
^~~~~~
```

```
/home/ajay/lsp/ldd/statically.c:23:1: note: in expansion of macro 'MODULE_DESCRIPTION'
```

```
MODULE_DESCRIPTION("Module inserted statically");
^~~~~~
```

```
make[2]: *** [/home/ajay/lsp/ldd/statically.o] Error 1
```

```
make[1]: *** [_module/home/ajay/lsp/ldd] Error 2
```

```
make[1]: Leaving directory /usr/src/linux-headers-4.4.0-144-generic'
```

```
make: *** [all] Error 2
```

Loading...

5

Admin

 Reply to [KOTHAPALLY AJAY KUMAR](#)

May 11, 2020 9:42 PM

Hi Ajay,

Did you follow the module info like below?

```
MODULE_LICENSE("GPL");  
MODULE_AUTHOR("EmbeTronicX <embetronicx@gmail.com  
or admin@embetronicx.com>");  
MODULE_DESCRIPTION("A simple hello world driver");  
MODULE_VERSION("1.1");
```

You have to use a semicolon (;) for each MODULE_* macros.

Seems you are missing those. If you don't miss those then attach your code. We will have a look at your code.

Loading...

 0   Reply

Admin


 Reply to [KOTHAPALLY AJAY KUMAR](#)

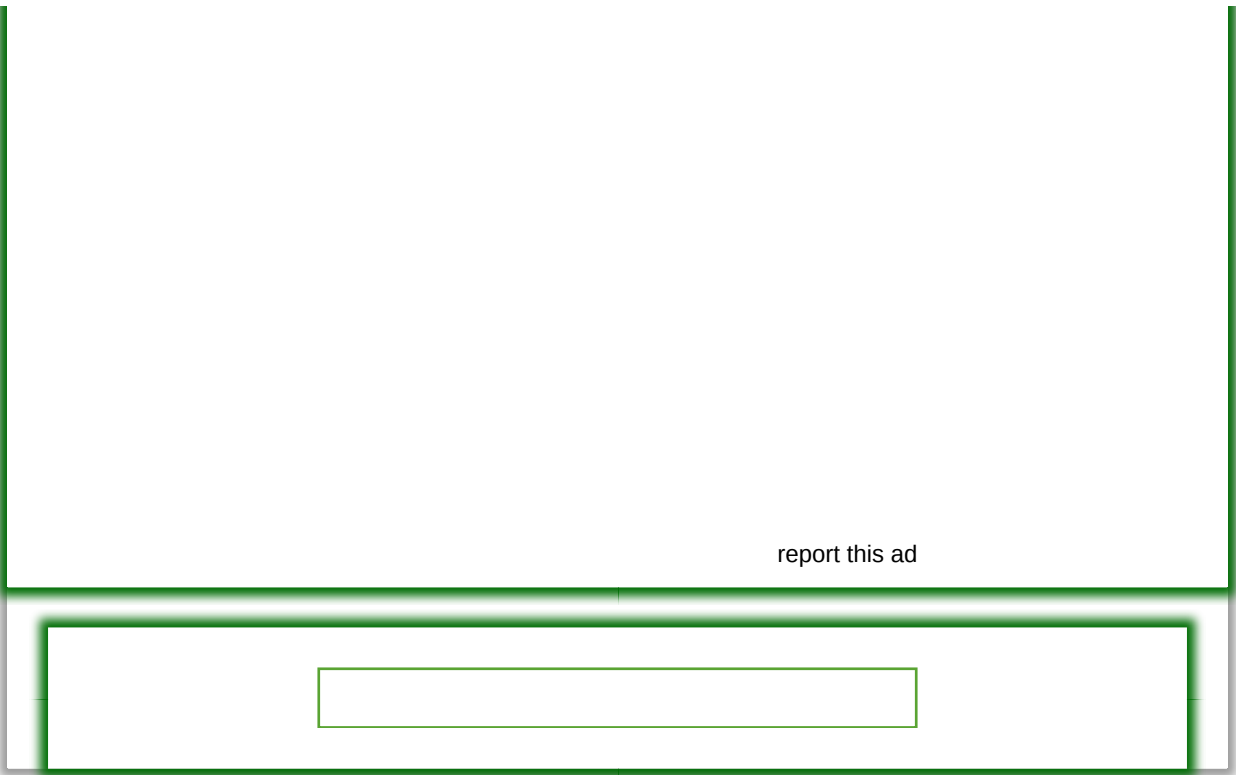
May 12, 2020 3:15 PM

Hi,

I have gone through your code which you sent by mail. As I mentioned earlier, you have missed the semicolon in line number 22 which is MODULE_AUTHOR. Please add the semicolon. It should fix the problem.

Loading...

 0   Reply



5

