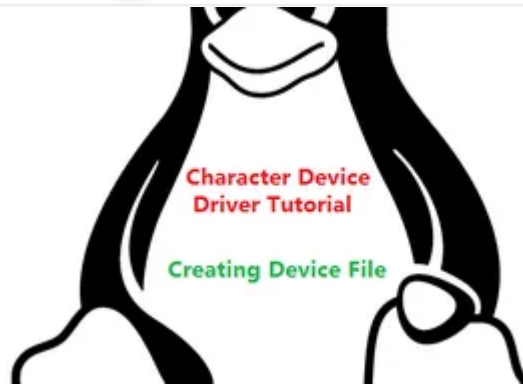




Sidebar▼

[Home](#) → [Tutorials](#) → [Linux](#) → [Device Drivers](#) → **Linux Device Driver**
Tutorial Part 5 - Device File Creation

📁 Device Drivers



Linux Device Driver Tutorial Part 5 - Device File Creation

This article is a continuation of the [Series on Linux Device Driver](#) and carries on the discussion on character drivers and their implementation. In this tutorial, we will discuss Device File Creation for Character Drivers.

5

Apple Music Turns 5 as It Continues Rivalry With Spotify

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

[1 Device File Creation for Character Drivers](#)

[2 Device Files](#)

[3 Creating Device File](#)

[3.1 Manually Creating Device File](#)

[3.1.1 Advantages](#)

[3.1.2 Programming](#)

[3.2 Automatically Creating Device File](#)

[3.2.1 Create the class](#)

[3.2.2 Create Device](#)

[3.2.3 Programming](#)

[3.2.4 Share this:](#)

[3.2.5 Like this:](#)

[3.2.6 Related](#)

Device File Creation for Character Drivers

In [5](#) our last [tutorial](#) we have seen how to assign a major and minor number. But if you see there it will create a major and minor numbers. But I did not create any device files in `/dev/` directory. The device file is important to communicate with the hardware. Let's start our tutorial.

Device Files

The device file allows transparent communication between user-space applications and hardware.

They are not normal “files”, but look like files from the program’s point of view: you can read from them, write to them, *mmap()* onto them, and so forth. When you access such a device “file,” the kernel recognizes the I/O request and passes it a device driver, which performs some operation, such as reading data from a serial port or sending data to hardware.

Device files (although inappropriately named, we will continue to use this term) provide a convenient way to access system resources without requiring the application programmer to know how the underlying device works. Under Linux, as with most Unix systems, device drivers themselves are part of the kernel.

All device files are stored in */dev* directory. Use *ls* command to browse the directory:

```
ls -l /dev/
```

Each device on the system should have a corresponding entry in */dev*. For example, */dev/ttyS0* corresponds to the first serial port, known as COM1 under MS-DOS; */dev/hda2* corresponds to the second partition on the first IDE drive. In fact, there should be entries in */dev* for devices you do not have. The device files are generally created during system installation and include every possible device driver. They don’t necessarily correspond to the actual hardware on your system. There are a number of pseudo-devices in */dev* that don’t correspond to any actual peripheral. For example, */dev/null* acts as a byte sink; any write request to */dev/null* will succeed, but the data are written will be ignored.

5

When using *ls -l* to list device files in */dev*, you’ll see something like the following:

```
crw--w---- 1 root tty 4, 0 Aug 15 10:40 tty0
brw-rw---- 1 root disk 1, 0 Aug 15 10:40 ram0
```

First of all, note that the first letter of the permissions field is denoted that driver type. Device files are denoted either by **b**, for block devices, or **c**, for character devices.

Also, note that the size field in the **ls -l** listing is replaced by two numbers, separated by a comma. The first value is the *major device number* and the second is the *minor device number*. This we have discussed in the previous [tutorial](#).

Creating Device File

We can create a device file in two ways.

1. Manually
2. Automatically

We will see one by one.

Manually Creating Device File

We can create the device file manually by using **mknod**.

```
mknod -m <permissions> <name> <device type> <major> <minor>
```

<name> - your device file name that should have a full path (**/dev/name**)

<device type> - Put **c** or **b**

c - Character Device

b - Block Device

<major> - major number of your driver

<minor> - minor number of your driver

-m <permissions> - optional argument that sets the permission bits of the new device file to *permissions*

Example:

```
sudo mknod -m 666 /dev/etx_device c 246 0
```

If you don't want to give permission, You can also use `chmod` to set the permissions for a device file after creation.

Advantages

- Anyone can create the device file using this method.
- You can create the device file even before loading the driver.

Programming

I took this program from the [previous tutorial](#). I'm going to create a device file manually for this driver.

```
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, "Embetronicx_Dev")) <0){
13         printk(KERN_INFO "Cannot allocate major number for device\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 device file using **ls -l /dev/**. By this time device file is not created for your driver.
- Create a device file using **mknod** and then check using **ls -l /dev/**.

```
linux@embetronicx-VirtualBox:/home/driver/driver$ sudo mknod -m 666 /dev/etx_device c 246 1  
linux@embetronicx-VirtualBox:/home/driver/driver$ ls -l /dev/ | grep "etx_device"  
crw-rw-rw- 1 root root 246, 0 Aug 15 13:53 etx_device
```

- Now our device file got created and registered with a major number.
- Unload the driver using **sudo rmmod**

Automatically Creating Device File

The automatic creation of device files can be handled with udev. Udev is the device manager for the Linux kernel that creates/removes device nodes in the /dev directory dynamically. Just follow the below steps.

1. Include the header file **linux/device.h** and **linux/kdev_t.h**
2. Create the struct Class
3. Create Device with the class which is created by the above step

Create the class

This will create the struct class for our device driver. It will create a structure under **/sys/class/**.

```
struct class * class_create (struct module *owner, const char *name);
```

5 *owner* – pointer to the module that is to “own” this struct class

name – pointer to a string for the name of this class

This is used to create a struct class pointer that can then be used in calls to **class_device_create**.

Note, the pointer created here is to be destroyed when finished by making a call to `class_destroy`.

```
void class_destroy (struct class * cls);
```

Create Device

This function can be used by char device classes. A struct device will be created in sysfs, registered to the specified class.

```
struct device *device_create (struct *class, struct device  
                             *parent, dev_t dev, const char *fmt, ...);
```

class - pointer to the struct class that this device should be registered to

parent - pointer to the parent struct device of this new device, if any

devt - the dev_t for the char device to be added

fmt - string for the device's name

... - variable arguments

A "dev" file will be created, showing the dev_t for the device, if the dev_t is not 0,0. If a pointer to a parent struct device is passed in, the newly created struct device will be a child of that device in `sysfs`. The pointer

to the struct device will be returned from the call. Any further [sysfs](#) files that might be required can be created using this pointer.

Note, you can destroy the device using `device_destroy()`.

```
void device_destroy (struct class * class, dev_t devt);
```

If you don't understand please refer the below program, Then you will understand.

Programming

```

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  #include <linux/device.h>
7
8  dev_t dev = 0;
9  static struct class *dev_class;
10
11
12 static int __init hello_world_init(void)
13 {
14     /*Allocating Major number*/
15     if((alloc_chrdev_region(&dev, 0, 1, "etx_Dev")) < 0){
16         printk(KERN_INFO "Cannot allocate major number for device\n");
17         return -1;
18     }
19     printk(KERN_INFO "Major = %d Minor = %d \n", MAJOR(dev), MINOR(dev));
20
21     /*Creating struct class*/
22     if((dev_class = class_create(THIS_MODULE, "etx_class")) == NULL){
23         printk(KERN_INFO "Cannot create the struct class for device\n");
24         goto r_class;
25     }
26
27     /*Creating device*/
28     if((device_create(dev_class, NULL, dev, NULL, "etx_device")) == NULL){
29         printk(KERN_INFO "Cannot create the Device\n");
30         goto r_device;
31     }
32     printk(KERN_INFO "Kernel Module Inserted Successfully...\n");
33     return 0;
34
35 r_device:
36     class_destroy(dev_class);
37 r_class:
38     unregister_chrdev_region(dev, 1);
39     return -1;
40 }
41
42 void __exit hello_world_exit(void)
43 {

```



```
44     device_destroy(dev_class,dev);
45     class_destroy(dev_class);
46     unregister_chrdev_region(dev, 1);
47     printk(KERN_INFO "Kernel Module Removed Successfully...\n");
48 }
49
50 module_init(hello_world_init);
51 module_exit(hello_world_exit);
52
53 MODULE_LICENSE("GPL");
54 MODULE_AUTHOR("EmbeTronicX <embetronicx@gmail.com or admin@embetronicx.com>");
55 MODULE_DESCRIPTION("A simple hello world driver");
56 MODULE_VERSION("1.2");
```

- Build the driver by using Makefile (***sudo make***)
- Load the driver using ***sudo insmod***
- Check the device file using ***ls -l /dev/ | grep "etx_device"***

```
linux@embetronicx-VirtualBox:/home/driver/driver$ ls -l /dev/ | grep "etx_device"
crw----- 1 root root 246, 0 Aug 15 13:36 etx_device
```

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

By this time, we have created the device file using those methods mentioned above. So Using this device file we can communicate the hardware. In our [next tutorial](#), we will show how to open that file, how to read that device file, how to write a device file, and how to close the device file. If you have any doubt, please comment below.

5

Article Rating

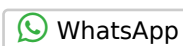


Share this:



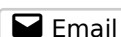
Post

Tweet



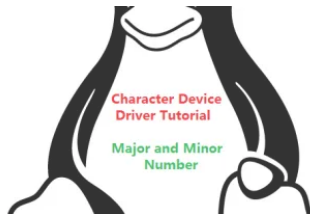
Share

0



Like this:

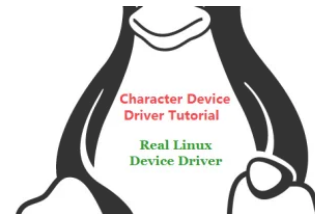
Loading...

Related

Linux Device Driver
Tutorial Part 4 -
Character Device Driver
In "Device Drivers"

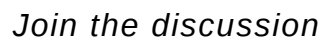


Linux Device Driver
Tutorial Part 32 - Misc
Device Driver
In "Device Drivers"



Linux Device Driver
Tutorial Part 7 - Linux
Device Driver Tutorial
Programming
In "Device Drivers"

Connect with



5 COMMENTS



Oldest ▼



August 21, 2017 1:52 AM

In this post you have created a device file using `mknod` and assigned major number 246, but it is getting dynamically allocating major and minor number in `helloworld` program. How the dynamically allocated hello world uses the major number 246 created by `mknod` for `etx` device ?

Loading...



Reply

5


owl Author

 Reply to [Anonymous](#)

August 21, 2017 10:46 AM

Dynamic method will allocate major number which is not used. Here, before creating major number, i saw that 246 is not used. Thats why i gave 246 for mknod. You can give any number 12 than 246. But that major number should not be using by any driver. I loaded helloworld and this driver at different time. Thats why it gave same major number.

Loading...

 0   Reply

 [Anonymous](#)

August 21, 2017 1:54 AM

Does both the major number created by dynamic allocation and major number created by mknod for etx_device file are same?

Loading...

 0   Reply

owl Author

 Reply to [Anonymous](#)

August 21, 2017 10:40 AM

Both are not same.

Loading...

 0   Reply

5



Sana Srikar

September 2, 2018 9:09 PM

hi embedtronicx ,
General doubt on device_create() ,my question is little off the topic.

any known use case of struct device *parent feild in
device_create() API ?

if any known use case could you please help me with it ?

Loading...



0



Reply

5

