# Part 4 – Character Device Driver

[ <https://embetronicx.com/tutorials/linux/device-drivers/character-device-driver-major-number-and-minor-number/> ]

# 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 a character driver.

# How Applications will communicate Hardware device?

This below diagram will show the full path of the communication.

A screenshot of a cell phone

Description automatically generated

* First Application will open the device file. This device file is created by device driver).
* Then This device file will find the correspond device driver using major and minor number.
* 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 like file. Driver will create the special file for every hardware devices. We can communicate to the hardware using those special files (device file).

If you want to create the special file, we should know about the Major number and minor number in 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.

**A screenshot of a cell phone

Description automatically generated**

These numbers are major numbers.

## Minor Number

Major number is identify the corresponding driver. Many devices may use same major number. So we need to assign the number to each devices which is using 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 by two ways.

1. Statically allocating
2. Dynamically Allocating

## Statically allocating

If you want to set the particular major number to 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 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 below function.

*MKDEV(int major, int minor);*

If you want to get your major number and minor number from dev\_t, use 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,

dev\_t dev = MKDEV(235, 0);

register\_chrdev\_region(dev,1, "Embetronicx\_Dev");

## Dynamically Allocating

If we dont want fixed major and minor number please use this method. This method will allocate the major number dynamically to your driver which is available.

int alloc\_chrdev\_region(dev\_t \*dev, unsigned int firstminor, unsigned int count, char \*name);

Where,

*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

Static method is only really useful if you know in advance which major number you want to start with. With Static method , you tell the kernel 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 tell the kernel 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.

#include<linux/kernel.h>

#include<linux/init.h>

#include<linux/module.h>

#include <linux/fs.h>

dev\_t dev = MKDEV(235, 0);

static int \_\_init hello\_world\_init(void)

{

register\_chrdev\_region(dev, 1, "Embetronicx\_Dev");

printk(KERN\_INFO "Major = %d Minor = %d \n",MAJOR(dev), MINOR(dev));

printk(KERN\_INFO "Kernel Module Inserted Successfully...\n");

return 0;

}

void \_\_exit hello\_world\_exit(void)

{

unregister\_chrdev\_region(dev, 1);

printk(KERN\_INFO "Kernel Module Removed Successfully...\n");

}

module\_init(hello\_world\_init);

module\_exit(hello\_world\_exit);

MODULE\_LICENSE("GPL");

MODULE\_AUTHOR("EmbeTronicX <embetronicx@gmail.com or admin@embetronicx.com>");

MODULE\_DESCRIPTION("A simple hello world driver");

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*

$ cat /proc/devices | grep "Embetronicx\_Dev"

235 Embetronicx\_Dev

* Unload the driver using *sudo rmmod*

# Program for Dynamically Allocating Major Number

This program will allocate major number dynamically.

#include<linux/kernel.h>

#include<linux/init.h>

#include<linux/module.h>

#include <linux/kdev\_t.h>

#include <linux/fs.h>

dev\_t dev = 0;

static int \_\_init hello\_world\_init(void)

{

/\*Allocating Major number\*/

if((alloc\_chrdev\_region(&dev, 0, 1, "Embetronicx\_Dev")) <0){

printk(KERN\_INFO "Cannot allocate major number for device 1\n");

return -1;

}

printk(KERN\_INFO "Major = %d Minor = %d \n",MAJOR(dev), MINOR(dev));

printk(KERN\_INFO "Kernel Module Inserted Successfully...\n");

return 0;

}

void \_\_exit hello\_world\_exit(void)

{

unregister\_chrdev\_region(dev, 1);

printk(KERN\_INFO "Kernel Module Removed Successfully...\n");

}

module\_init(hello\_world\_init);

module\_exit(hello\_world\_exit);

MODULE\_LICENSE("GPL");

MODULE\_AUTHOR("EmbeTronicX <embetronicx@gmail.com or admin@embetronicx.com>");

MODULE\_DESCRIPTION("A simple hello world driver");

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*

$ cat /proc/devices | grep "Embetronicx\_Dev"

243 Embetronicx\_Dev

* Unload the driver using *sudo rmmod*

This function allocates 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.