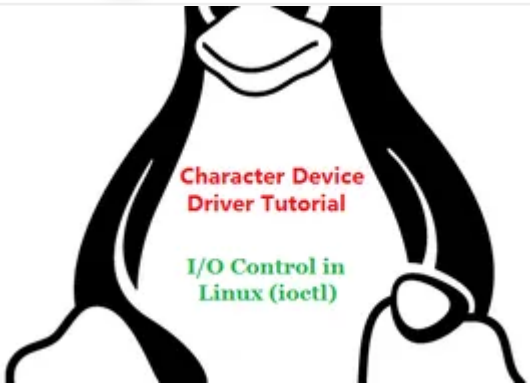
EmbeTronicX
Embedded Tutorials Zone

Sidebar

Home → Tutorials → Linux → Device Drivers → **Linux Device Driver Tutorial Part 8 – I/O Control in Linux IOCTL()**

Device Drivers



Linux Device Driver Tutorial Part 8 – I/O Control in Linux IOCTL()

This article is a continuation of the Series on Linux Device Driver and carries on the discussion on character drivers and their implementation. This is Part 8 of the Linux device driver tutorial. Now we will discuss IOCTL Tutorial in Linux.

19

Post Contents [hide]

19

Introduction

The operating system segregates virtual memory into kernel space and userspace. Kernel space is strictly reserved for running the kernel,

kernel extensions, and most device drivers. In contrast, user space is the memory area where all user-mode applications work, and this memory can be swapped out when necessary.
There are many ways to Communicate between the Userspace and Kernel Space, they are:

- IOCTL
- Procfs
- Sysfs
- Configfs
- Debugfs
- Sysctl
- UDP Sockets
- Netlink Sockets

In this tutorial, we will see IOCTL.

IOCTL Tutorial in Linux

IOCTL

IOCTL is referred to as Input and Output Control, which is used to talking to device drivers. This system call, available in most driver categories. The major use of this is in case of handling some specific operations of a device for which the kernel does not have a system call by default.

Some real-time applications of ioctl are Ejecting the media from a “cd” drive, to change the Baud Rate of Serial port, Adjust the Volume, Reading or Writing device registers, etc. We already have the write and read function in our device driver. But it is not enough for all cases.

Steps involved in IOCTL

19

There are some steps involved to use IOCTL.

- Create IOCTL command in driver
- Write IOCTL function in the driver
- Create IOCTL command in a Userspace application
- Use the IOCTL system call in a Userspace

Create IOCTL Command in the Driver

To implement a new ioctl command we need to follow the following steps.

1. Define the ioctl code

```
#define "ioctl name" __IOX("magic number","command number","argument type")
```

where ***IOX*** can be :
"**IO**": an ioctl with no parameters
"**IOW**": an ioctl with write parameters (copy_from_user)
"**IOR**": an ioctl with read parameters (copy_to_user)
"**IOWR**": an ioctl with both write and read parameters

- The **Magic Number** is a unique number or character that will differentiate our set of ioctl calls from the other ioctl calls. some times the major number for the device is used here.
- Command Number is the number that is assigned to the ioctl . This is used to differentiate the commands from one another.
- The last is the type of data.

2. Add the header file **linux/ioctl.h** to make use of the above-mentioned calls.

Example

```
1 #include <linux/ioctl.h>
2
3 #define WR_VALUE _IOW('a','a',int32_t*)
4 #define RD_VALUE _IOR('a','b',int32_t*)
```

Write IOCTL function in the driver

The next step is to implement the ioctl call we defined into the corresponding driver. We need to add the ioctl function to our driver. Find the prototype of the function below.

```
int  ioctl(struct inode *inode,struct file *file,unsigned int cmd,unsigned long arg)
```

Where,

<inode> : is the inode number of the file being worked on.
<file> : is the file pointer to the file that was passed by the application.
<cmd> : is the ioctl command that was called from the userspace.
<arg> : are the arguments passed from the userspace.

Within the function “ioctl” we need to implement all the commands that we defined above (**WR_VALUE**, **RD_VALUE**). We need to use the same commands in the **switch** statement which is defined above.

Then we need to inform the kernel that the ioctl calls are implemented in the function “**etx_ioctl**“. This is done by making the **fops** pointer “**unlocked_ioctl**” to point to “**etx_ioctl**” as shown below.

Example

```
static long etx_ioctl(struct file *file, unsigned int cmd, unsigned long arg)
{
         switch(cmd) {
                case WR_VALUE:
                        copy_from_user(&value ,(int32_t*) arg, sizeof(value));
                        printk(KERN_INFO "Value = %d\n", value);
                        break;
                case RD_VALUE:
                        copy_to_user((int32_t*) arg, &value, sizeof(value));
                        break;
        }
        return 0;
}


static struct file_operations fops =
{
  .owner = THIS_MODULE,
  .read = etx_read,
  .write = etx_write,
  .open = etx_open,
  .unlocked_ioctl = etx_ioctl,
  .release = etx_release,
};
```

Now we need to call the new ioctl command from a user application.

Create IOCTL command in a Userspace

application

Just define the ioctl command like how we defined in driver.

Example:

```
1 #define WR_VALUE _IOW('a','a',int32_t*)
2 #define RD_VALUE _IOR('a','b',int32_t*)
```

Use IOCTL system call in Userspace

Include the header file **<sys/ioctl.h>**.Now we need to call the new ioctl command from a user application.

```
long ioctl( "file descriptor","ioctl command","Arguments");
```

Where,

<file descriptor>: This the open file on which the ioctl command needs to be executed, which would generally be device files.
<ioctl command>: ioctl command which is implemented to achieve the desired functionality
<arguments>: The arguments that need to be passed to the ioctl command.

Example

```
1 ioctl(fd, WR_VALUE, (int32_t*) &number);
2
3 ioctl(fd, RD_VALUE, (int32_t*) &value);
```

Now we will see the complete driver and application.

Device Driver Source Code

19

In this example we only implemented IOCTL. In this driver, I've defined one variable (**int32_t value**). Using ioctl command we can read or change the variable. So other functions like open, close, read, write, We simply left empty. Just go through the code below.

driver.c

```
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kdev_t.h>
#include <linux/fs.h>
#include <linux/cdev.h>
#include <linux/device.h>
#include<linux/slab.h>                 //kmalloc()
#include<linux/uaccess.h>              //copy_to/from_user()
#include <linux/ioctl.h>


#define WR_VALUE _IOW('a','a',int32_t*)
#define RD_VALUE _IOR('a','b',int32_t*)

int32_t value = 0;

dev_t dev = 0;
static struct class *dev_class;
static struct cdev etx_cdev;

static int __init etx_driver_init(void);
static void __exit etx_driver_exit(void);
static int etx_open(struct inode *inode, struct file *file);
static int etx_release(struct inode *inode, struct file *file);
static ssize_t etx_read(struct file *filp, char __user *buf, size_t len,loff_t * of
static ssize_t etx_write(struct file *filp, const char *buf, size_t len, loff_t * o
static long etx_ioctl(struct file *file, unsigned int cmd, unsigned long arg);

static struct file_operations fops =
{
        .owner          = THIS_MODULE,
        .read           = etx_read,
        .write          = etx_write,
        .open           = etx_open,
        .unlocked_ioctl = etx_ioctl,
        .release        = etx_release,
};

static int etx_open(struct inode *inode, struct file *file)
{
        printk(KERN_INFO "Device File Opened...!!!\n");
        return 0;
}

static int etx_release(struct inode *inode, struct file *file)
{
        printk(KERN_INFO "Device File Closed...!!!\n");
        return 0;
}

static ssize_t etx_read(struct file *filp, char __user *buf, size_t len, loff_t *of
{
        printk(KERN_INFO "Read Function\n");
        return 0;
}
static ssize_t etx_write(struct file *filp, const char __user *buf, size_t len, lof
{
        printk(KERN_INFO "Write function\n");
        return 0;
}

static long etx_ioctl(struct file *file, unsigned int cmd, unsigned long arg)
```

```
64  {
65           switch(cmd) {
66                  case WR_VALUE:
67                          copy_from_user(&value ,(int32_t*) arg, sizeof(value));
68                          printk(KERN_INFO "Value = %d\n", value);
69                          break;
70                  case RD_VALUE:
71                          copy_to_user((int32_t*) arg, &value, sizeof(value));
72                          break;
73          }
74          return 0;
75  }
76
77
78  static int __init etx_driver_init(void)
79  {
80          /*Allocating Major number*/
81          if((alloc_chrdev_region(&dev, 0, 1, "etx_Dev")) <0){
82                  printk(KERN_INFO "Cannot allocate major number\n");
83                  return -1;
84          }
85          printk(KERN_INFO "Major = %d Minor = %d \n",MAJOR(dev), MINOR(dev));
86
87          /*Creating cdev structure*/
88          cdev_init(&etx_cdev,&fops);
89
90          /*Adding character device to the system*/
91          if((cdev_add(&etx_cdev,dev,1)) < 0){
92              printk(KERN_INFO "Cannot add the device to the system\n");
93              goto r_class;
94          }
95
96          /*Creating struct class*/
97          if((dev_class = class_create(THIS_MODULE,"etx_class")) == NULL){
98              printk(KERN_INFO "Cannot create the struct class\n");
99              goto r_class;
100         }
101
102         /*Creating device*/
103         if((device_create(dev_class,NULL,dev,NULL,"etx_device")) == NULL){
104             printk(KERN_INFO "Cannot create the Device 1\n");
105             goto r_device;
106         }
107         printk(KERN_INFO "Device Driver Insert...Done!!!\n");
108     return 0;
109
110 r_device:
111         class_destroy(dev_class);
112 r_class:
113         unregister_chrdev_region(dev,1);
114         return -1;
115 }
116
117 void __exit etx_driver_exit(void)
118 {
119         device_destroy(dev_class,dev);
120         class_destroy(dev_class);
121         cdev_del(&etx_cdev);
122         unregister_chrdev_region(dev, 1);
123     printk(KERN_INFO "Device Driver Remove...Done!!!\n");
124 }
125
126 module_init(etx_driver_init);
```

19

```
127 module_exit(etx_driver_exit);
128
129 MODULE_LICENSE("GPL");
130 MODULE_AUTHOR("EmbeTronicX <embetronicx@gmail.com or admin@embetronicx.com>");
131 MODULE_DESCRIPTION("A simple device driver");
132 MODULE_VERSION("1.5");
```

Makefile:

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

Application Source Code

This application is used to write the value to the driver. Then read the value again.

test_app.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <sys/types.h>
5  #include <sys/stat.h>
6  #include <fcntl.h>
7  #include <unistd.h>
8  #include<sys/ioctl.h>
9
10 #define WR_VALUE _IOW('a','a',int32_t*)
```

```
#define RD_VALUE _IOR('a','b',int32_t*)

int main()
{
        int fd;
        int32_t value, number;
        printf("*********************************\n");
        printf("*******WWW.EmbeTronicX.com*******\n");

        printf("\nOpening Driver\n");
        fd = open("/dev/etx_device", O_RDWR);
        if(fd < 0) {
                printf("Cannot open device file...\n");
                return 0;
        }

        printf("Enter the Value to send\n");
        scanf("%d",&number);
        printf("Writing Value to Driver\n");
        ioctl(fd, WR_VALUE, (int32_t*) &number);

        printf("Reading Value from Driver\n");
        ioctl(fd, RD_VALUE, (int32_t*) &value);
        printf("Value is %d\n", value);

        printf("Closing Driver\n");
        close(fd);
}
```

Building Driver and Application

- Build the driver by using Makefile (*sudo make*)
- Use the below line in the terminal to compile the user space application.

```
gcc -o test_app test_app.c
```

Execution (Output)

As of now, we have driver.ko and test_app. Now we will see the output.

- Load the driver using **sudo insmod driver.ko**
- Run the application (**sudo ./test_app**)

```
19
*********************************
*******WWW.EmbeTronicX.com*******

Opening Driver
Enter the Value to send
```

- Enter the value to pass

```
23456

Writing Value to Driver
Reading Value from Driver
Value is 23456
Closing Driver
```

- Now check the value using **dmesg**

```
Device File Opened...!!!
Value = 23456
Device File Closed...!!!
```

- Our value 23456 was passed to the kernel and it was updated.

This is a simple example using ioctl in the driver. If you want to send multiple arguments, put those variables into the structure, and pass the structure.

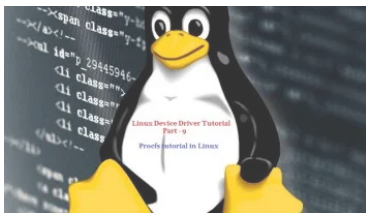
In our next tutorial, we will see another userspace and kernel space communication method which is procfs.

Share this:

Like this:

Loading...

Related



Linux Device Driver Tutorial Part 9 - Procfs in Linux
In "Device Drivers"

Linux Device Driver Tutorial Part 11 – Sysfs in Linux Kernel
In "Device Drivers"

Linux Device Driver Part 1 : Introduction
In "Device Drivers"

Subscribe

Connect with | Login

Join the discussion

{} [+]

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

19 COMMENTS

Oldest

19

ganesan guru

August 27, 2018 2:34 AM

Hi fellas, I’m getting a error namely, error: expected expression before ‘<<’ token #define WR _IOC(_IOC_WRITE,245,0,char*) while compiling user application program with gcc app.c. The code of app.c is as follows: #include #include #include #include #include #include #include #include #include #include #define WR _IOC(_IOC_WRITE,245,0,char*) int main() { int fd; int32_t value, number; printf("*******EXECUTION STARTED*******n"); printf("nOpening Drivern"); fd = open("/dev/gurudeve", O_RDWR); if(fd < 0) { printf("Cannot open device file...n"); return 0; } printf("Enter the Value to sendn"); scanf("%d",&number); printf("Writing Value to Drivern"); ioctl(fd, WR, (int32_t) &number); // printf("Reading Value from Drivern"); // ioctl(fd, RD_VALUE, (int32_t) &value); // printf("Value is %dn", value); printf("Closing… Read more »

Last edited 1 month ago by owl

0 Reply

19

EmbeTronicx India

Reply to ganesan guru August 29, 2018 1:41 AM

Hi Ganesan Guru,

Can you please verify the header files that you are including? It should be
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/ioctl.h>

instead of

#include
#include
#include <fcntl.h>
#include <unistd.h>

#include
#include

Thanks.

Loading...

2 Reply

Mohan

October 7, 2018 11:55 PM

etx_cdev.owner = THIS_MODULE;
etx_cdev.ops = &fops;
These two lines are throughing error.
Above two lines are not required.
And the program contains compilation warnings

19

Loading...

0 Reply

EmbeTronicx India

Reply to Mohan October 13, 2018 10:54 AM

Hi Mohan,

You are correct. That is not required. We've updated the source code... Cdev_init will take care of that.

Thank you.

Loading...

1 Reply

Siddhi Verma

October 12, 2018 1:04 AM

Can somebody just tell me here how are we assigning the major and minor number for the driver?

Loading...

0 Reply

EmbeTronicx India

Reply to Siddhi Verma October 14, 2018 5:51 AM

Hi Siddhi Verma,
In this driver we are allocating major and minor number dynamically. Please see this below post for further information.
https://embetronicx.com/tutorials/linux/device-drivers/character-device-driver-major-number-and-minor-number/#Dynamically_Allocating
Thanks.

Loading...

19

0 Reply

Alok Prasad

December 19, 2019 10:38 PM

HI Team, Good blog ..but try to use updated API

1. pr_* instead of printk
2.dev_* for devices
3.Not to use proc for driver instead debugfs
4.concept of sysfs for configuration parameters

Loading...

2 Reply

castiel

February 17, 2020 2:49 PM

This will be better if you can provide the codes in github

Loading...

0 Reply

EmbeTronicX

Reply to castiel April 1, 2020 2:16 AM

Hi Castiel,

Initially we were putting code in github also. But when we want to update the code, we need to do it in multiple places. That's why we dropped it. In future we will do that.

Loading...

1 Reply

19

Jake John

May 2, 2020 3:19 AM

Really Nice blog. I appreciate your hard work. Please put some new posts on device driver.

Loading...

1 Reply

sagi

May 16, 2020 6:11 PM

Hi i was going through the above steps, this is really detailed explanation for newbies like me. i've few queries (pardon my stupidity); on the steps mentioned above; 1. Define the ioctl code #define "ioctl name" __IOX("magic number","command number","argument type") which file this define should be added? (is it linux/ioctl.h?) can you please elaborate a bit on where exactly this should be added; 2. Add the header file linux/ioctl.h to make use of the above-mentioned calls. –> this linux/ioctl.h means in /usr/include/linux/ioctl.h? or other path. in the my existing linux system /usr/include/linux/ioctl.h-> do i need to add additional line 3.... Read more »

0 Reply

19

sagi

Reply to sagi May 16, 2020 9:41 PM

i tried, but getting below errors- please help and help to answer above questions:

[root@wlr11s04 driver_ioctl_test]# sudo make
make -C /lib/modules/5.7.0-rc5+/build M=/usr/driver_ioctl_test modules
make[1]: Entering directory ‘/usr/src/kernels/5.7.0-rc5+’
make[2]: *** No rule to make target ‘/usr/driver_ioctl_test /driver.o’, needed by ‘__build’. Stop.
make[1]: *** [Makefile:1729: /usr/driver_ioctl_test] Error 2
make[1]: Leaving directory ‘/usr/src/kernels/5.7.0-rc5+’
make: *** [Makefile:4: all] Error 2

Loading…

0 Reply

19

sagi

Reply to sagi May 16, 2020 11:45 PM

some more progress but, still there are issues in make, can someone help here? [root@wlr11s04 driver_ioctl_test]# sudo make make -C /lib/modules/5.7.0-rc5+/build M=/usr/driver_ioctl_test modules make[1]: Entering directory ‘/usr/src/kernels/5.7.0-rc5+’ CC [M] /usr/driver_ioctl_test/driver_ioctl.o /usr/driver_ioctl_test /driver_ioctl.c: In function ‘etx_ioctl’: /usr/driver_ioctl_test /driver_ioctl.c:67:25: warning: ignoring return value of ‘copy_from_user’, declared with attribute warn_unused_result [-Wunused-result] copy_from_user(&value ,(int32_t) arg, sizeof(value)); ^~~~~~~ /usr/driver_ioctl_test /driver_ioctl.c:71:25: warning: ignoring return value of ‘copy_to_user’, declared with attribute warn_unused_result [-Wunused-result] copy_to_user((int32_t) arg, &value, sizeof(value)); ^~~~~~~ MODPOST 1 modules FATAL: modpost: GPL-incompatible module driver_ioctl.ko uses GPL-only symbol ‘device_destroy’ make[2]: *** [scripts/Makefile.modpost:94: __modpost] Error 1 make[1]: *** [Makefile:1642: modules] Error 2 make[1]: Leaving directory ‘/usr/src/kernels/5.7.0-rc5+’ make: *** [Makefile:4: all]... Read more »

0 Reply

19

sagi

Reply to sagi May 17, 2020 12:12 AM

I think i figured out; GPL license issue; was able to build do you’ve similar approach for testing DMA drivers, i mean i want to know further on how do we test the DMA functionality and make sure driver does not have any issues, as such. features i was looking for clarity in terms of testing is memcpy, block fill, block fill with NULL bits, standard DMA operation [root@wlr11s04 driver_ioctl_test]# sudo make make -C /lib/modules/5.7.0-rc5+/build M=/usr/driver_ioctl_test modules make[1]: Entering directory ‘/usr/src/kernels/5.7.0-rc5+’ CC [M] /usr/driver_ioctl_test/driver_ioctl.o /usr/driver_ioctl_test/driver_ioctl.c: In function ‘etx_ioctl’: /usr/driver_ioctl_test /driver_ioctl.c:67:25: warning: ignoring return value of ‘copy_from_user’, declared with attribute warn_unused_result [-Wunused-result]... Read more »

0 Reply

19

sagi

Reply to sagi May 17, 2020 12:20 AM

error seen while compiling application for testing driver please help; /usr/include/linux /ioctl.h:4:9: error: macro names must be identifiers #define “ioctl name” __IOX(“magic number”,”command number”,”argument type”) /added by Sagar on WW20P6/ ^~~~~~~~~~~~ In file included from ioctl_test.c:8: /usr/include /sys/ioctl.h:42:13: error: expected declaration specifiers or ‘…’ before string constant long ioctl(“file descriptor”,”ioctl command”,”Arguments”); /* added by Sagar on WW20p4 to test ioctl operation/ ^~~~~~~~~~~~~~~~~ /usr/include /sys/ioctl.h:42:31: error: expected declaration specifiers or ‘…’ before string constant long ioctl(“file descriptor”,”ioctl command”,”Arguments”); / added by Sagar on WW20p4 to test ioctl operation/ ^~~~~~~~~~~~~~~~ /usr/include /sys/ioctl.h:42:47: error: expected declaration specifiers or ‘…’ before string constant long ioctl(“file... Read more »

0 Reply

19

sagi

Reply to sagi May 17, 2020 12:44 AM

sorry for the thrash i was able to clear the errors and was able to do clean compilation and was able to test the driver with test app; if you could able to help on shading more light on DMA specific questions; i want to know further on how do we test the DMA functionality and make sure driver does not have any issues, as such. features i was looking for clarity in terms of testing DMA driver is; a. memcpy, b. block fill, c. block fill with NULL your detailed analysis will help me ramp up on this appreciate... Read more »

0 Reply

Ehsan

June 20, 2020 3:02 AM

Hi,

I use IOCTL in android source code, at surface flinger, it is ok and work well, but when I run Pubg game this ioctl not works and give the error 25, means not a typewriter. I change the magic number some times but it did not fixed the error.

I would appreciate if any body could help me. Is there a way to debug it.

Best,
Ehsan

Loading...

19

0 Reply

cvam

June 20, 2020 8:58 PM

why is the __exit function is not static ?

Loading...

0 Reply

EmbeTronicX

Reply to cvam June 20, 2020 9:11 PM

Hi,

Yes. We have missed the static keyword. But if you don't put static, this function can be accessed from other file also.

Loading...

0 Reply

19

19