EmbeTronicX
Embedded Tutorials Zone

Sidebar

Home → Tutorials → Linux → Device Drivers → **Linux Device Driver Tutorial Part 25 – Sending Signal from Linux Device Driver to User Space**

Device Drivers

Linux Device Driver Tutorial Part 25 – Sending Signal from Linux Device Driver to User Space

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 25 – Sending Signal from Linux Device Driver to User Space.

6

Post Contents [hide]

Prerequisites

In the example section, we explained signals using the interrupt program. So I would recommend you to explore interrupts using the below links before starting this.

1. Interrupts Concepts

2. Interrupts Examples Program
3. IOCTL Tutorial

Signals

Introduction

Generally, A **signal** is an action that is intended to send a particular message. It can be sound, gesture, event, etc. Below are the normal signals which we are using the day to day life.

- When we take money in ATM, we will get a message
- Calling someone by making sound or gesture
- Microwave oven making a sound when it finishes its job
- etc.

What about Linux? **Signals** are a way of sending simple messages which are used to notify a process or thread of a particular event. In Linux, there are many processes that will be running at a time. We can send a signal from one process to another process. Signals are one of the oldest inter-process communication methods. These signals are asynchronous. Like User space signals, can we send a signal to userspace from kernel space? Yes, why not. We will see the complete Signals in upcoming tutorials. In this tutorial, we will learn how to send a signal from Linux Device Driver to User Space.

Sending Signal from Linux Device Driver to User Space

Using the following steps easily we can send the signals.

1. Decide the signal that you want to send.
2. Register the user space application with the driver.
3. Once something happened (in our example we used interrupts) send signals to userspace.
4. Unregister the user space application when you have done with it.

Decide the signal that you want to send

First, select the signal number which you want to send. In our case, we are going to send signal 44.

Example:

```
#define SIGETX    44
```

Register the user space application with driver

Before sending the signal, your device driver should know to whom it needs to send the signal. For that, we need to register the process to the driver. So we need to send the PID to the driver first. Then that driver will use the PID and sends the signal. You can register the application PID in anyways like IOCTL, Open/read/write call. In our example, we are going to register using IOCTL.

Example:

```
static long etx_ioctl(struct file *file, unsigned int cmd, unsigned long arg)
{
    if (cmd == REG_CURRENT_TASK) {
        printk(KERN_INFO "REG_CURRENT_TASK\n");
        task = get_current();
        signum = SIGETX;
    }
    return 0;
}
```

Send signals to user space

After registering the application to the driver, then the driver can able to send the signal when it requires. In our example, we will send the signal when we get the interrupt.

Example:

```
//Interrupt handler for IRQ 11.
static irqreturn_t irq_handler(int irq,void *dev_id) {
    struct siginfo info;
    printk(KERN_INFO "Shared IRQ: Interrupt Occurred");

    //Sending signal to app
    memset(&info, 0, sizeof(struct siginfo));
    info.si_signo = SIGETX;
    info.si_code = SI_QUEUE;
    info.si_int = 1;

    if (task != NULL) {
        printk(KERN_INFO "Sending signal to app\n");
        if(send_sig_info(SIGETX, &info, task) < 0) {
            printk(KERN_INFO "Unable to send signal\n");
```

```
        }
    }
    return IRQ_HANDLED;
}
```

Unregister the user space application

When you are done with your task, you can unregister your application. Here we are unregistering when that application closes the driver.

Example:

```
static int etx_release(struct inode *inode, struct file *file)
{
    struct task_struct *ref_task = get_current();
    printk(KERN_INFO "Device File Closed...!!!\n");

    //delete the task
    if(ref_task == task) {
        task = NULL;
    }
    return 0;
}
```

Device Driver Source Code

The complete device driver code is given below. In this source code, When we read the **/dev/etx_device** interrupt will hit (To understand interrupts in Linux go to this tutorial). Whenever interrupt hits, I'm sending the signal to userspace application who registered already. Since it is a tutorial post, I'm not going to do any job in interrupt handler except sending the signal.

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>
#include <linux/interrupt.h>
#include <asm/io.h>

#define SIGETX 44

#define REG_CURRENT_TASK _IOW('a','a',int32_t*)
```

```
17
18  #define IRQ_NO 11
19
20  /* Signaling to Application */
21  static struct task_struct *task = NULL;
22  static int signum = 0;
23
24  int32_t value = 0;
25
26  dev_t dev = 0;
27  static struct class *dev_class;
28  static struct cdev etx_cdev;
29
30  static int __init etx_driver_init(void);
31  static void __exit etx_driver_exit(void);
32  static int etx_open(struct inode *inode, struct file *file);
33  static int etx_release(struct inode *inode, struct file *file);
34  static ssize_t etx_read(struct file *filp, char __user *buf, size_t len,loff_t * of
35  static ssize_t etx_write(struct file *filp, const char *buf, size_t len, loff_t * o
36  static long etx_ioctl(struct file *file, unsigned int cmd, unsigned long arg);
37
38  static struct file_operations fops =
39  {
40          .owner          = THIS_MODULE,
41          .read           = etx_read,
42          .write          = etx_write,
43          .open           = etx_open,
44          .unlocked_ioctl = etx_ioctl,
45          .release        = etx_release,
46  };
47
48  //Interrupt handler for IRQ 11.
49  static irqreturn_t irq_handler(int irq,void *dev_id) {
50      struct siginfo info;
51      printk(KERN_INFO "Shared IRQ: Interrupt Occurred");
52
53      //Sending signal to app
54      memset(&info, 0, sizeof(struct siginfo));
55      info.si_signo = SIGETX;
56      info.si_code = SI_QUEUE;
57      info.si_int = 1;
58
59      if (task != NULL) {
60          printk(KERN_INFO "Sending signal to app\n");
61          if(send_sig_info(SIGETX, &info, task) < 0) {
62              printk(KERN_INFO "Unable to send signal\n");
63          }
64      }
65
66      return IRQ_HANDLED;
67  }
68
69  static int etx_open(struct inode *inode, struct file *file)
70  {
71      printk(KERN_INFO "Device File Opened...!!!\n");
72      return 0;
73  }
74
75  static int etx_release(struct inode *inode, struct file *file)
76  {
77      struct task_struct *ref_task = get_current();
78      printk(KERN_INFO "Device File Closed...!!!\n");
79
```

```
80      //delete the task
81      if(ref_task == task) {
82          task = NULL;
83      }
84      return 0;
85  }
86
87  static ssize_t etx_read(struct file *filp, char __user *buf, size_t len, loff_t *of
88  {
89      printk(KERN_INFO "Read Function\n");
90      asm("int $0x3B");  //Triggering Interrupt. Corresponding to irq 11
91      return 0;
92  }
93  static ssize_t etx_write(struct file *filp, const char __user *buf, size_t len, lof
94  {
95      printk(KERN_INFO "Write function\n");
96      return 0;
97  }
98
99  static long etx_ioctl(struct file *file, unsigned int cmd, unsigned long arg)
100 {
101     if (cmd == REG_CURRENT_TASK) {
102         printk(KERN_INFO "REG_CURRENT_TASK\n");
103         task = get_current();
104         signum = SIGETX;
105     }
106     return 0;
107 }
108
109
110 static int __init etx_driver_init(void)
111 {
112     /*Allocating Major number*/
113     if((alloc_chrdev_region(&dev, 0, 1, "etx_Dev")) <0){
114             printk(KERN_INFO "Cannot allocate major number\n");
115             return -1;
116     }
117     printk(KERN_INFO "Major = %d Minor = %d \n",MAJOR(dev), MINOR(dev));
118
119     /*Creating cdev structure*/
120     cdev_init(&etx_cdev,&fops);
121
122     /*Adding character device to the system*/
123     if((cdev_add(&etx_cdev,dev,1)) < 0){
124         printk(KERN_INFO "Cannot add the device to the system\n");
125         goto r_class;
126     }
127
128     /*Creating struct class*/
129     if((dev_class = class_create(THIS_MODULE,"etx_class")) == NULL){
130         printk(KERN_INFO "Cannot create the struct class\n");
131         goto r_class;
132     }
133
134     /*Creating device*/
135     if((device_create(dev_class,NULL,dev,NULL,"etx_device")) == NULL){
136         printk(KERN_INFO "Cannot create the Device 1\n");
137         goto r_device;
138     }
139
140     if (request_irq(IRQ_NO, irq_handler, IRQF_SHARED, "etx_device", (void *)(irq_ha
141         printk(KERN_INFO "my_device: cannot register IRQ ");
142         goto irq;
```

```
143     }
144
145     printk(KERN_INFO "Device Driver Insert...Done!!!\n");
146     return 0;
147 irq:
148     free_irq(IRQ_NO,(void *)(irq_handler));
149 r_device:
150     class_destroy(dev_class);
151 r_class:
152     unregister_chrdev_region(dev,1);
153     return -1;
154 }
155
156 void __exit etx_driver_exit(void)
157 {
158     free_irq(IRQ_NO,(void *)(irq_handler));
159     device_destroy(dev_class,dev);
160     class_destroy(dev_class);
161     cdev_del(&etx_cdev);
162     unregister_chrdev_region(dev, 1);
163     printk(KERN_INFO "Device Driver Remove...Done!!!\n");
164 }
165
166 module_init(etx_driver_init);
167 module_exit(etx_driver_exit);
168
169 MODULE_LICENSE("GPL");
170 MODULE_AUTHOR("EmbeTronicX <embetronicx@gmail.com>");
171 MODULE_DESCRIPTION("A simple device driver - Signals");
172 MODULE_VERSION("1.20");
```

Makefile:

```
1 obj-m += 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
```

Application Source Code

This application register with the driver using IOCTL. Once it registered, it will be waiting for the signal from the driver. If we want to close this application we need to press CTRL+C. Because we it will run infinitely. We have installed CTRL+C signal handler.

test_app.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
```

```c
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <signal.h>

#define REG_CURRENT_TASK _IOW('a','a',int32_t*)

#define SIGETX 44

static int done = 0;
int check = 0;

void ctrl_c_handler(int n, siginfo_t *info, void *unused)
{
    if (n == SIGINT) {
        printf("\nrecieved ctrl-c\n");
        done = 1;
    }
}

void sig_event_handler(int n, siginfo_t *info, void *unused)
{
    if (n == SIGETX) {
        check = info->si_int;
        printf ("Received signal from kernel : Value =  %u\n", check);
    }
}

int main()
{
    int fd;
    int32_t value, number;
    struct sigaction act;

    printf("********************************\n");
    printf("*******WWW.EmbeTronicX.com*******\n");
    printf("********************************\n");

    /* install ctrl-c interrupt handler to cleanup at exit */
    sigemptyset (&act.sa_mask);
    act.sa_flags = (SA_SIGINFO | SA_RESETHAND);
    act.sa_sigaction = ctrl_c_handler;
    sigaction (SIGINT, &act, NULL);

    /* install custom signal handler */
    sigemptyset(&act.sa_mask);
    act.sa_flags = (SA_SIGINFO | SA_RESTART);
    act.sa_sigaction = sig_event_handler;
    sigaction(SIGETX, &act, NULL);

    printf("Installed signal handler for SIGETX = %d\n", SIGETX);

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

    printf("Registering application ...");
    /* register this task with kernel for signal */
```

```
67      if (ioctl(fd, REG_CURRENT_TASK,(int32_t*) &number)) {
68          printf("Failed\n");
69          close(fd);
70          exit(1);
71      }
72      printf("Done!!!\n");
73
74      while(!done) {
75          printf("Waiting for signal...\n");
76
77          //blocking check
78          while (!done && !check);
79          check = 0;
80      }
81
82      printf("Closing Driver\n");
83      close(fd);
84  }
```

Building Driver and Application

- Build the driver by using Makefile (***sudo make***)
- Use the below the 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**)

```
*********************************
*******WWW.EmbeTronicX.com*******
*********************************
Installed signal handler for SIGETX = 44
Opening Driver
Registering application ...Done!!!
Waiting for signal...
```

- This application will be waiting for the signal
- To send the signal from driver to app, we need to trigger the interrupt by reading the driver (**sudo cat /dev/etx_device**).

- Now see the Dmesg (**dmesg**)

```
Major = 246 Minor = 0
Device Driver Insert...Done!!!
Device File Opened...!!!
REG_CURRENT_TASK
Device File Opened...!!!
Read Function
Shared IRQ: Interrupt Occurred
Sending signal to app
Device File Closed...!!!
```

- As per the print, the driver has sent the signal. Now check the app.

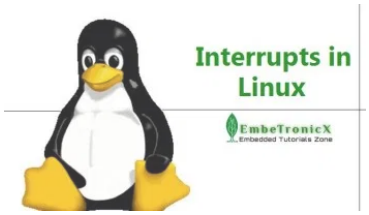
```
Received signal from kernel : Value = 1
```

- So the application also got the signal.
- Close the application by pressing CTRL+C
- Unload the module using **sudo rmmod driver**

Note: If you use newer version of kernel, then you might face some issues. Please go through the below comments and fix it.

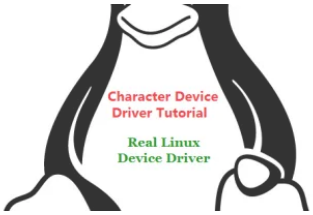
In our next tutorial, we will discuss kernel timer in the Linux device driver.

Related



Linux Device Driver Tutorial Part 12 - Interrupts in Linux Kernel
In "Device Drivers"

Linux Device Driver Tutorial Part 13 – Interrupts Example Program in Linux Kernel
In "Device Drivers"



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

Join the discussion

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

6 COMMENTS Oldest

Mihaita Ivascu

February 14, 2019 10:35 AM

Hello,

Interesting post. Could you please share the application as well? I think the driver code has been copied over.

Thanks,
Mihaita

Loading...

0 Reply

EmbeTronicx India

Reply to Mihaita Ivascu February 15, 2019 9:34 AM

Hi Mihaita,
Thanks for pointing that. Now we have copied the application code.
Thanks.

Loading...

6

0 Reply

swati agarwal

June 9, 2019 6:01 AM

Getting following error in dmesg logs
do_IRQ: 7.59 No irq handler for vector

while triggering interrupt
sudo cat /dev/etx_device

Tried following steps to get rid of this error

sudo gedit /etc/default/grub
change GRUB_CMDLINE_LINUX_DEFAULT=”quiet splash pci=nomsi,noaer”
3.sudo update-grub

but its not working

kernel version :- 4.14.0-041400-generic

Loading...

0 Reply

EmbeTronicX

Reply to swati agarwal April 1, 2020 5:52 AM

Hi,

You might be using newer kernel than we have used it. Please refer this link to fix that.

Loading...

0 Reply

6

shubham Chidrawar

September 18, 2019 12:47 PM

getting error

/home/bucky/Desktop/new_driver/driver.c: In function ‘irq_handler’:
/home/bucky/Desktop/new_driver/driver.c:65:27: error: storage size of ‘info’ isn’t known
struct kernel_siginfo info;//defined in signal.h

Loading...

0 Reply

6

EmbeTronicX

Reply to shubham Chidrawar April 1, 2020 6:03 AM

Hi Shubham,
You might be using the newer kernel than we have used it. You should port our old kernel changes to new kernel. We have provided our changes below.

Include this header file (**`#include "linux/sched/signal.h"`**)
Then typecast the **`info`** argument to (**`struct kernel_siginfo *`**)

Please apply this patch.

```
+ #include "linux/sched/signal.h"
- if(send_sig_info(SIGETX, &info, task) < 0) {
+ if(send_sig_info(SIGETX, (struct kernel_siginfo *)&info, task) < 0) {
```

After applying this you, might face problem with interrupt too. Please refer this link to fix that issue.

Loading...

0 Reply

6

6