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

[ <https://embetronicx.com/tutorials/linux/device-drivers/sending-signal-from-linux-device-driver-to-user-space/> ]

# Prerequisites

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

**Signals Introduction**

Generally A **signal** is a action which is intended to send a particular message. It can be sound, gesture, event etc. Below are the normal signals which we are using 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 sound when it finishes its job
* etc.

What about Linux? **Signals** are a way of sending simple messages which is used to notify a process or thread of a particular event. In Linux, there are many process will be running at a time. We can send signal from one process to another process. Signals are one of the oldest inter-process communication methods. This signals are asynchronous.

Like User space signals, can we send signal to user space from kernel space? Yes why not. We will see the complete Signals in upcoming tutorials. In this tutorial we will learn how to send 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 driver.
3. Once something happened (in our example we used interrupts) send signals to user space.
4. Unregister the user space application when you done.

**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 signal, your device driver should know to whom it needs to send the signal. For that we need to register the process to driver. So we need to send the PID to driver first. Then that driver will use the PID and sends the signal. You can register the application PID in any ways like IOCTL, Open/read/write call. In our example we are going to register using [IOCTL](https://www.embetronicx.com/tutorials/linux/device-drivers/ioctl-tutorial-in-linux/).

**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 driver, then 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 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](https://www.embetronicx.com/tutorials/linux/device-drivers/linux-device-driver-tutorial-part-13-interrupt-example-program-in-linux-kernel/)). Whenever interrupt hits, I’m sending signal to user space application who registered already. Since it is a tutorial post, I’m not going to do any job in interrupt handler except sending 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\*)

#define IRQ\_NO 11

/\* Signaling to Application \*/

static struct task\_struct \*task = NULL;

static int signum = 0;

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 \* off);

static ssize\_t etx\_write(struct file \*filp, const char \*buf, size\_t len, loff\_t \* off);

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,

};

//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;

}

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)

{

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;

}

static ssize\_t etx\_read(struct file \*filp, char \_\_user \*buf, size\_t len, loff\_t \*off)

{

printk(KERN\_INFO "Read Function\n");

asm("int $0x3B"); //Triggering Interrupt. Corresponding to irq 11

return 0;

}

static ssize\_t etx\_write(struct file \*filp, const char \_\_user \*buf, size\_t len, loff\_t \*off)

{

printk(KERN\_INFO "Write function\n");

return 0;

}

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;

}

static int \_\_init etx\_driver\_init(void)

{

/\*Allocating Major number\*/

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

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

return -1;

}

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

/\*Creating cdev structure\*/

cdev\_init(&etx\_cdev,&fops);

/\*Adding character device to the system\*/

if((cdev\_add(&etx\_cdev,dev,1)) < 0){

printk(KERN\_INFO "Cannot add the device to the system\n");

goto r\_class;

}

/\*Creating struct class\*/

if((dev\_class = class\_create(THIS\_MODULE,"etx\_class")) == NULL){

printk(KERN\_INFO "Cannot create the struct class\n");

goto r\_class;

}

/\*Creating device\*/

if((device\_create(dev\_class,NULL,dev,NULL,"etx\_device")) == NULL){

printk(KERN\_INFO "Cannot create the Device 1\n");

goto r\_device;

}

if (request\_irq(IRQ\_NO, irq\_handler, IRQF\_SHARED, "etx\_device", (void \*)(irq\_handler))) {

printk(KERN\_INFO "my\_device: cannot register IRQ ");

goto irq;

}

printk(KERN\_INFO "Device Driver Insert...Done!!!\n");

return 0;

irq:

free\_irq(IRQ\_NO,(void \*)(irq\_handler));

r\_device:

class\_destroy(dev\_class);

r\_class:

unregister\_chrdev\_region(dev,1);

return -1;

}

void \_\_exit etx\_driver\_exit(void)

{

free\_irq(IRQ\_NO,(void \*)(irq\_handler));

device\_destroy(dev\_class,dev);

class\_destroy(dev\_class);

cdev\_del(&etx\_cdev);

unregister\_chrdev\_region(dev, 1);

printk(KERN\_INFO "Device Driver Remove...Done!!!\n");

}

module\_init(etx\_driver\_init);

module\_exit(etx\_driver\_exit);

MODULE\_LICENSE("GPL");

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

MODULE\_DESCRIPTION("A simple device driver - Signals");

MODULE\_VERSION("1.20");

**Makefile:**

obj-m += driver.o

KDIR = /lib/modules/$(shell uname -r)/build

all:

make -C $(KDIR) M=$(shell pwd) modules

clean:

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**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#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 \*/

if (ioctl(fd, REG\_CURRENT\_TASK,(int32\_t\*) &number)) {

printf("Failed\n");

close(fd);

exit(1);

}

printf("Done!!!\n");

while(!done) {

printf("Waiting for signal...\n");

//blocking check

while (!done && !check);

check = 0;

}

printf("Closing Driver\n");

close(fd);

}

**Building Driver and Application**

* Build the driver by using Makefile (*sudo make*)
* Use below line in 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 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, driver has send the signal. Now check the app.

*Received signal from kernel : Value = 1*

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