# Part 21 – Tasklets | Dynamic Method

[ <https://embetronicx.com/tutorials/linux/device-drivers/tasklets-dynamic-method/> ]

This is the Linux Device Driver Tutorial Part 21 – Tasklets Dynamic Method Tutorial.

# Prerequisites

This is the continuation of Interrupts in Linux Kernel. So I’d suggest you to know some ideas about Linux Interrupts. You can find the some useful tutorials about Interrupts and Bottom Halves below.

1. [Interrupts in Linux Kernel](https://www.embetronicx.com/tutorials/linux/device-drivers/interrupts-in-linux-kernel/)
2. [Interrupts Example Program](https://www.embetronicx.com/tutorials/linux/device-drivers/linux-device-driver-tutorial-part-13-interrupt-example-program-in-linux-kernel/)
3. [Workqueue Example – Static Method](https://www.embetronicx.com/tutorials/linux/device-drivers/workqueue-in-linux-kernel/)
4. [Workqueue Example – Dynamic Method](https://www.embetronicx.com/tutorials/linux/device-drivers/workqueue-in-linux-dynamic-creation/)
5. [Workqueue Example – Own Workqueue](https://www.embetronicx.com/tutorials/linux/device-drivers/work-queue-in-linux-own-workqueue/)
6. [Tasklet Example – Static Method](https://embetronicx.com/tutorials/linux/device-drivers/tasklet-static-method/)

# Tasklets in Linux Driver

# Introduction

In our [Previous Tutorial](https://embetronicx.com/tutorials/linux/device-drivers/tasklet-static-method/) we have seen the Tasklet using Static Method. In that method we had initialized the tasklet statically. But in this tutorial we are going to initialize the tasklet using dynamically. **So except creation of the tasklet, everything will be same as Previous tutorial. Please refer previous tutorial for Scheduling, Enable, Disable, Kill the Tasklet.**

# Dynamically Creation of Tasklet

## tasklet\_init

|  |
| --- |
| This function used to Initialize the tasklet in dynamically.  **void tasklet\_init ( struct tasklet\_struct \*t,**  **void(\*)(unsigned long) func,** **unsigned long data**  **);**  Where,  t – tasklet struct that should be initialized  func – This is the main function of the tasklet. Pointer to the function that needs to be scheduled for execution at a later time.  data – Data to be passed to the function “func”. |

### Example

/\* Tasklet by Dynamic Method \*/

struct tasklet\_struct \*tasklet;

/\* Init the tasklet bt Dynamic Method \*/

tasklet = kmalloc(sizeof(struct tasklet\_struct),GFP\_KERNEL);

if(tasklet == NULL) {

printk(KERN\_INFO "etx\_device: cannot allocate Memory");

}

tasklet\_init(tasklet,tasklet\_fn,0);

Now we will see how the function is working in background. When I call the function like above, it assigns the parameter to the passed tasklet structure. It will be looks like below.

|  |  |
| --- | --- |
| 1  2  3  4 | tasklet->func = tasklet\_fn;             //function  tasklet->data = 0;                      //data arg  tasklet->state = TASKLET\_STATE\_SCHED;  //Tasklet state is scheduled  atomic\_set(&tasklet->count, 0);        //taskelet enabled |

**NOTE : Please refer previous tutorial for rest of the function like Scheduling, Enable, Disable, Kill the Tasklet.**

# Programming

## Driver Source Code

In that 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 scheduling the task to the tasklet. I’m not going to do any job in both interrupt handler and tasklet function (only print),  since it is a tutorial post. But in real tasklet, this function can be used to carry out any operations that need to be scheduled.

*NOTE: In this source code many unwanted functions will be there (which is not related to the Tasklet). Because I’m just maintaining the source code throughout these Device driver series.*

#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/sysfs.h>

#include<linux/kobject.h>

#include <linux/interrupt.h>

#include <asm/io.h>

#include <asm/hw\_irq.h>

#define IRQ\_NO 11

void tasklet\_fn(unsigned long);

/\* Tasklet by Dynamic Method \*/

struct tasklet\_struct \*tasklet;

/\*Tasklet Function\*/

void tasklet\_fn(unsigned long arg)

{

printk(KERN\_INFO "Executing Tasklet Function : arg = %ld\n", arg);

}

//Interrupt handler for IRQ 11.

static irqreturn\_t irq\_handler(int irq,void \*dev\_id) {

printk(KERN\_INFO "Shared IRQ: Interrupt Occurred");

/\*Scheduling Task to Tasklet\*/

tasklet\_schedule(tasklet);

return IRQ\_HANDLED;

}

volatile int etx\_value = 0;

dev\_t dev = 0;

static struct class \*dev\_class;

static struct cdev etx\_cdev;

struct kobject \*kobj\_ref;

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

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Driver Fuctions \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Sysfs Fuctions \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

static ssize\_t sysfs\_show(struct kobject \*kobj,

struct kobj\_attribute \*attr, char \*buf);

static ssize\_t sysfs\_store(struct kobject \*kobj,

struct kobj\_attribute \*attr,const char \*buf, size\_t count);

struct kobj\_attribute etx\_attr = \_\_ATTR(etx\_value, 0660, sysfs\_show, sysfs\_store);

static struct file\_operations fops =

{

.owner = THIS\_MODULE,

.read = etx\_read,

.write = etx\_write,

.open = etx\_open,

.release = etx\_release,

};

static ssize\_t sysfs\_show(struct kobject \*kobj,

struct kobj\_attribute \*attr, char \*buf)

{

printk(KERN\_INFO "Sysfs - Read!!!\n");

return sprintf(buf, "%d", etx\_value);

}

static ssize\_t sysfs\_store(struct kobject \*kobj,

struct kobj\_attribute \*attr,const char \*buf, size\_t count)

{

printk(KERN\_INFO "Sysfs - Write!!!\n");

sscanf(buf,"%d",&etx\_value);

return count;

}

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 \*off)

{

struct irq\_desc \*desc;

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

/\* New way of mapping irq lines on newer kernels \*/

desc = irq\_to\_desc(11);

if (!desc)

return -EINVAL;

\_\_this\_cpu\_write(vector\_irq[59], desc);

/\* Triggering Interrupt \*/

asm("int $0x3B"); // 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 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;

}

/\*Creating a directory in /sys/kernel/ \*/

kobj\_ref = kobject\_create\_and\_add("etx\_sysfs",kernel\_kobj);

/\*Creating sysfs file for etx\_value\*/

if(sysfs\_create\_file(kobj\_ref,&etx\_attr.attr)){

printk(KERN\_INFO"Cannot create sysfs file......\n");

goto r\_sysfs;

}

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

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

goto irq;

}

/\* Init the tasklet bt Dynamic Method \*/

tasklet = kmalloc(sizeof(struct tasklet\_struct),GFP\_KERNEL);

if(tasklet == NULL) {

printk(KERN\_INFO "etx\_device: cannot allocate Memory");

goto irq;

}

tasklet\_init(tasklet,tasklet\_fn,0);

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

return 0;

irq:

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

r\_sysfs:

kobject\_put(kobj\_ref);

sysfs\_remove\_file(kernel\_kobj, &etx\_attr.attr);

r\_device:

class\_destroy(dev\_class);

r\_class:

unregister\_chrdev\_region(dev,1);

cdev\_del(&etx\_cdev);

return -1;

}

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

{

/\* Kill the Tasklet \*/

tasklet\_kill(tasklet);

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

kobject\_put(kobj\_ref);

sysfs\_remove\_file(kernel\_kobj, &etx\_attr.attr);

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 - Tasklet part 2");

MODULE\_VERSION("1.16");

**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

# Building and Testing Driver

* Build the driver by using Makefile (sudo make)
* Load the driver using sudo insmod driver.ko
* To trigger interrupt read device file (sudo cat /dev/etx\_device)
* Now see the Dmesg (dmesg)

*$ dmesg*

*[12372.451624] Major = 246 Minor = 0  
[12372.456927] Device Driver Insert…Done!!!  
[12375.112089] Device File Opened…!!!  
[12375.112109] Read function  
[12375.112134] Shared IRQ: Interrupt Occurred  
[12375.112139] Executing Tasklet Function : arg = 0  
[12375.112147] Device File Closed…!!!  
[12377.954952] Device Driver Remove…Done!!!*

* We can able to see the print “**Shared IRQ: Interrupt Occurred**“ and “**Executing Tasklet Function : arg = 0**“
* Unload the module using sudo rmmod driver