



Sidebar▼

[Home](#) → [Tutorials](#) → [Linux](#) → [Device Drivers](#) → **Linux Device Driver**
Tutorial Part 21 - Tasklets | Dynamic Method

📁 Device Drivers



Linux Device Driver Tutorial Part 21 - Tasklets | Dynamic Method

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 21 - Tasklet Dynamic Method Tutorial.

2

Apple Music Turns 5 as It Continues Rivalry With Spotify

Post Contents [\[hide\]](#)

- 1 Prerequisites
- 2 Tasklets in Linux Driver
- 3 Introduction
- 4 Dynamically Creation of Tasklet
 - 4.1 tasklet_init
 - 4.1.1 Example
- 5 Tasklet Programming
 - 5.1 Driver Source Code
 - 5.2 MakeFile
- 6 Building and Testing Driver
 - 6.0.1 Share this:
 - 6.0.2 Like this:
 - 6.0.3 Related

Prerequisites

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

1. [Interrupts in Linux Kernel](#)
2. [Interrupts Example Program](#)
3. [Workqueue Example - Static Method](#)
4. [Workqueue Example - Dynamic Method](#)
5. [Workqueue Example - Own Workqueue](#)
6. [Tasklet Example - Static Method](#)

Tasklets in Linux Driver

Introduction

In our [Previous Tutorial](#) 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 the same as the previous tutorial. Please refer the 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 schedule for execution at a later time.

data - Data to be passed to the function "func".

Example

```
1 /* Tasklet by Dynamic Method */
2 struct tasklet_struct *tasklet;
3
4 /* Init the tasklet bt Dynamic Method */
5 tasklet = kmalloc(sizeof(struct tasklet_struct),GFP_KERNEL);
6 if(tasklet == NULL) {
7     printk(KERN_INFO "etx_device: cannot allocate Memory");
8 }
9 tasklet_init(tasklet,tasklet_fn,0);
```

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

```
1 tasklet->func = tasklet_fn;           //function
2 tasklet->data = 0;                     //data arg
3 tasklet->state = TASKLET_STATE_SCHED; //Tasklet state is scheduled
4 atomic_set(&tasklet->count, 0);        //taskelet enabled
```

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

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](#)). 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.

```

1  #include <linux/kernel.h>
2  #include <linux/init.h>
3  #include <linux/module.h>
4  #include <linux/kdev_t.h>
5  #include <linux/fs.h>
6  #include <linux/cdev.h>
7  #include <linux/device.h>
8  #include <linux/slab.h>           //kmalloc()
9  #include <linux/uaccess.h>       //copy_to/from_user()
10 #include <linux/sysfs.h>
11 #include <linux/kobject.h>
12 #include <linux/interrupt.h>
13 #include <asm/io.h>
14
15
16 #define IRQ_NO 11
17
18 void tasklet_fn(unsigned long);
19
20 /* Tasklet by Dynamic Method */
21 struct tasklet_struct *tasklet;
22
23
24 /*Tasklet Function*/
25 void tasklet_fn(unsigned long arg)
26 {
27     printk(KERN_INFO "Executing Tasklet Function : arg = %ld\n", arg);
28 }
29
30
31 //Interrupt handler for IRQ 11.
32 static irqreturn_t irq_handler(int irq, void *dev_id) {
33     printk(KERN_INFO "Shared IRQ: Interrupt Occurred");
34     /*Scheduling Task to Tasklet*/
35     tasklet_schedule(tasklet);
36
37     return IRQ_HANDLED;
38 }
39
40
41 volatile int etx_value = 0;
42
43

```

```

44 dev_t dev = 0;
45 static struct class *dev_class;
46 static struct cdev etx_cdev;
47 struct kobject *kobj_ref;
48
49 static int __init etx_driver_init(void);
50 static void __exit etx_driver_exit(void);
51
52 /***** Driver Fuctions *****/
53 static int etx_open(struct inode *inode, struct file *file);
54 static int etx_release(struct inode *inode, struct file *file);
55 static ssize_t etx_read(struct file *filp,
56                         char __user *buf, size_t len, loff_t * off);
57 static ssize_t etx_write(struct file *filp,
58                         const char *buf, size_t len, loff_t * off);
59
60 /***** Sysfs Fuctions *****/
61 static ssize_t sysfs_show(struct kobject *kobj,
62                          struct kobj_attribute *attr, char *buf);
63 static ssize_t sysfs_store(struct kobject *kobj,
64                          struct kobj_attribute *attr, const char *buf, size_t count);
65
66 struct kobj_attribute etx_attr = __ATTR(etx_value, 0660, sysfs_show, sysfs_store);
67
68 static struct file_operations fops =
69 {
70     .owner          = THIS_MODULE,
71     .read           = etx_read,
72     .write          = etx_write,
73     .open           = etx_open,
74     .release        = etx_release,
75 };
76
77 static ssize_t sysfs_show(struct kobject *kobj,
78                          struct kobj_attribute *attr, char *buf)
79 {
80     printk(KERN_INFO "Sysfs - Read!!!\n");
81     return sprintf(buf, "%d", etx_value);
82 }
83
84 static ssize_t sysfs_store(struct kobject *kobj,
85                          struct kobj_attribute *attr, const char *buf, size_t count)
86 {
87     printk(KERN_INFO "Sysfs - Write!!!\n");
88     sscanf(buf, "%d", &etx_value);
89     return count;
90 }
91
92 static int etx_open(struct inode *inode, struct file *file)
93 {
94     printk(KERN_INFO "Device File Opened...!!!\n");
95     return 0;
96 }
97
98 static int etx_release(struct inode *inode, struct file *file)
99 {
100     printk(KERN_INFO "Device File Closed...!!!\n");
101     return 0;
102 }
103
104 static ssize_t etx_read(struct file *filp,
105                       char __user *buf, size_t len, loff_t *off)
106 {

```

```

107     printk(KERN_INFO "Read function\n");
108     asm("int $0x3B"); // Corresponding to irq 11
109     return 0;
110 }
111 static ssize_t etx_write(struct file *filp,
112                         const char __user *buf, size_t len, loff_t *off)
113 {
114     printk(KERN_INFO "Write Function\n");
115     return 0;
116 }
117
118
119 static int __init etx_driver_init(void)
120 {
121     /*Allocating Major number*/
122     if((alloc_chrdev_region(&dev, 0, 1, "etx_Dev")) < 0){
123         printk(KERN_INFO "Cannot allocate major number\n");
124         return -1;
125     }
126     printk(KERN_INFO "Major = %d Minor = %d \n", MAJOR(dev), MINOR(dev));
127
128     /*Creating cdev structure*/
129     cdev_init(&etx_cdev, &fops);
130
131     /*Adding character device to the system*/
132     if((cdev_add(&etx_cdev, dev, 1)) < 0){
133         printk(KERN_INFO "Cannot add the device to the system\n");
134         goto r_class;
135     }
136
137     /*Creating struct class*/
138     if((dev_class = class_create(THIS_MODULE, "etx_class")) == NULL){
139         printk(KERN_INFO "Cannot create the struct class\n");
140         goto r_class;
141     }
142
143     /*Creating device*/
144     if((device_create(dev_class, NULL, dev, NULL, "etx_device")) == NULL){
145         printk(KERN_INFO "Cannot create the Device 1\n");
146         goto r_device;
147     }
148
149     /*Creating a directory in /sys/kernel/ */
150     kobj_ref = kobject_create_and_add("etx_sysfs", kernel_kobj);
151
152     /*Creating sysfs file for etx_value*/
153     if(sysfs_create_file(kobj_ref, &etx_attr.attr)){
154         printk(KERN_INFO "Cannot create sysfs file.....\n");
155         goto r_sysfs;
156     }
157     if (request_irq(IRQ_NO, irq_handler, IRQF_SHARED, "etx_device", (void *)0))
158         printk(KERN_INFO "etx_device: cannot register IRQ ");
159     goto irq;
160 }
161
162 /* Init the tasklet bt Dynamic Method */
163 tasklet = kmalloc(sizeof(struct tasklet_struct), GFP_KERNEL);
164 if(tasklet == NULL) {
165     printk(KERN_INFO "etx_device: cannot allocate Memory");
166     goto irq;
167 }
168 tasklet_init(tasklet, tasklet_fn, 0);
169

```

```

170         printk(KERN_INFO "Device Driver Insert...Done!!!\n");
171         return 0;
172
173
174 irq:
175         free_irq(IRQ_NO, (void *) (irq_handler));
176
177 r_sysfs:
178         kobject_put(kobj_ref);
179         sysfs_remove_file(kernel_kobj, &etx_attr.attr);
180
181 r_device:
182         class_destroy(dev_class);
183 r_class:
184         unregister_chrdev_region(dev, 1);
185         cdev_del(&etx_cdev);
186         return -1;
187 }
188
189 void __exit etx_driver_exit(void)
190 {
191     /* Kill the Tasklet */
192     tasklet_kill(tasklet);
193     free_irq(IRQ_NO, (void *) (irq_handler));
194     kobject_put(kobj_ref);
195     sysfs_remove_file(kernel_kobj, &etx_attr.attr);
196     device_destroy(dev_class, dev);
197     class_destroy(dev_class);
198     cdev_del(&etx_cdev);
199     unregister_chrdev_region(dev, 1);
200     printk(KERN_INFO "Device Driver Remove...Done!!!\n");
201 }
202
203 module_init(etx_driver_init);
204 module_exit(etx_driver_exit);
205
206 MODULE_LICENSE("GPL");
207 MODULE_AUTHOR("EmbeTronicX <embetronicx@gmail.com>");
208 MODULE_DESCRIPTION("A simple device driver - Tasklet part 2");
209 MODULE_VERSION("1.16");

```

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

```

Building and Testing Driver

- Build the driver by using Makefile (***sudo make***)
- Load the driver using ***sudo insmod driver.ko***

- To trigger the interrupt read device file (**sudo cat /dev/etx_device**)
- Now see the Dmesg (**dmesg**)

```
linux@embetronicx-VirtualBox: 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**

In our [next tutorial](#), we will discuss Mutex in the Linux device driver.



Share this:



Like this:

Loading...

Related



[Tutorial Part 20 – Tasklet | Static Method](#)
In "Device Drivers"

[Tutorial Part 14 – Workqueue in Linux Kernel Part 1](#)
In "Device Drivers"

[Tutorial Part 23 – Spinlock in Linux Kernel Part 1](#)
In "Device Drivers"

☒ Subscribe ▼

Connect with | [Login](#)



Join the discussion

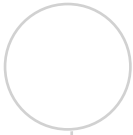
B *I* U          



This site uses Akismet to reduce spam. [Learn how your comment data is processed.](#)

2 COMMENTS

Oldest ▼

**Pallavi Shetty**

May 2, 2020 5:24 PM

Good article. Cheers.

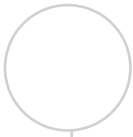
Loading...



0



Reply

**KOTHAPALLY AJAY KUMAR**

May 25, 2020 4:36 PM

Good explanation sir.

Can you please upload device drivers interview questions or provide me if you have any links?

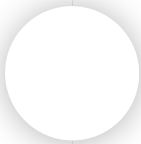
Loading...



0



Reply

2

report this ad



6

