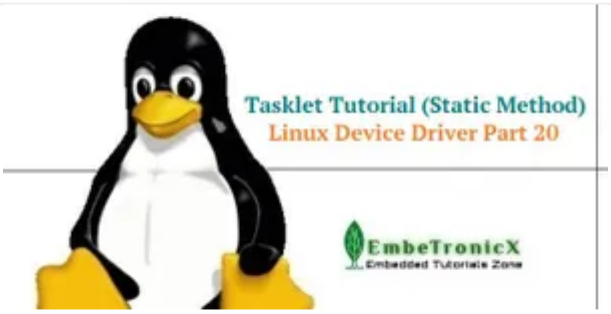
EmbeTronicX
Embedded Tutorials Zone

Sidebar

Home → Tutorials → Linux → Device Drivers → **Linux Device Driver Tutorial Part 20 – Tasklet | Static Method**

Device Drivers



Linux Device Driver Tutorial Part 20 – Tasklet | Static 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 20 – Tasklet Static Method Tutorial.

1

Post Contents [hide]

Prerequisites

This is the continuation of Interrupts in the Linux Kernel. So I'd suggest you, know some ideas about Linux Interrupts. You can find the 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

Bottom Half

When Interrupt triggers, Interrupt Handler should be execute very quickly and it should not run for more time (it should not perform time-consuming tasks). If we have the interrupt handler which is doing more tasks then we need to divide into two halves.

1. Top Half
2. Bottom Half

Top Half is nothing but our interrupt handler. If our interrupt handler is doing less task, then the top half is more than enough. No need of the bottom half in that situation. But if we have more work when interrupt hits, then we need bottom half. The bottom half runs in the future, at a more convenient time, with all interrupts enabled. So, The job of bottom halves is to perform any interrupt-related work not performed by the interrupt handler.

There are 4 bottom half mechanisms are available in Linux:

1. Workqueue
2. Threaded IRQs
3. Softirqs
4. **Tasklets**

In this tutorial, we will see Tasklets in Linux Kernel.

Tasklets in Linux Kernel

Tasklets are used to queue up work to be done at a later time. Tasklets can be run in parallel, but the same tasklet cannot be run on multiple

CPUs at the same time. Also, each tasklet will run only on the CPU that schedules it, to optimize cache usage. Since the thread that queued up the tasklet must complete before it can run the tasklet, race conditions are naturally avoided. However, this arrangement can be suboptimal, as other potentially idle CPUs cannot be used to run the tasklet. Therefore workqueues can, and should be used instead, and workqueues were already discussed here.

In short, a **tasklet** is something like a very small thread that has neither stack, not the context of its own. Such “threads” work quickly and completely.

Points To Remember

Before using Tasklets, you should consider these below points.

- Tasklets are atomic, so we cannot use **sleep()** and such synchronization primitives as mutexes, semaphores, etc. from them. But we can use spinlock.
- A tasklet only runs on the same core (CPU) that schedules it.
- Different tasklets can be running in parallel. But at the same time, a tasklet cannot be called concurrently with itself, as it runs on one CPU only.
- Tasklets are executed by the principle of non-preemptive scheduling, one by one, in turn. We can schedule them with two different priorities: ***normal*** and ***high***.

We can create tasklet in Two ways.

1. **Static Method**
2. **Dynamic Method**

In this tutorial, we will see a static method.

Tasklet Structure

This is the important data structure for the tasklet.

```
    struct tasklet_struct *next;
    unsigned long state;
    atomic_t count;
    void (*func)(unsigned long);
    unsigned long data;
};
```

Here,

`next` – The next tasklet in line for scheduling.

`state` – This state denotes Tasklet's State. **`TASKLET_STATE_SCHED`** (**Scheduled**) or **`TASKLET_STATE_RUN`** (**Running**).

`count` – It holds a nonzero value if the tasklet is disabled and 0 if it is enabled.

`func` – This is the main function of the tasklet. Pointer to the function that needs to scheduled for execution at a later time.

`data` – Data to be passed to the function "func".

Create Tasklet

The below macros used to create a tasklet.

DECLARE_TASKLET

This macro used to create the tasklet structure and assigns the parameters to that structure.

If we are using this macro then tasklet will be in enabled state.

```
DECLARE_TASKLET(name, func, data);
```

`name` – name of the structure to be create.

`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

```
DECLARE_TASKLET(tasklet,tasklet_fn, 1);
```

Now we will see how the macro is working. When I call the macro like above, first it creates tasklet structure with the name of tasklet. Then it assigns the parameter to that structure. It will be looks like below.

```
struct tasklet_struct tasklet = { NULL, 0, 0, tasklet_fn, 1 };

                      (or)

struct tasklet_struct tasklet;
tasklet.next = NULL;
taklet.state = TASKLET_STATE_SCHED;  //Tasklet state is scheduled
tasklet.count = 0;                   //taskelet enabled
tasklet.func = tasklet_fn;           //function
tasklet.data = 1;                    //data arg
```

DECLARE_TASKLET_DISABLED

The tasklet can be declared and set at a disabled state, which means that tasklet can be scheduled, but will not run until the tasklet is specifically enabled. You need to use **`tasklet_enable`** to enable.

`DECLARE_TASKLET_DISABLED(name, func, data);`

`name` – name of the structure to be created.

`func` – This is the main function of the tasklet. Pointer to the function that needs to scheduled for execution at a later time.

`data` – Data to be passed to the function "**`func`**".

Enable and Disable Tasklet

tasklet_enable

This used to enable the tasklet.

`void tasklet_enable(struct);`

`t` – pointer to the tasklet struct

tasklet_disable

This used to disable the tasklet wait for the completion of tasklet's operation.

```
void tasklet_disable(struct tasklet_struct *t);
```

`t` – pointer to the tasklet struct

tasklet_disable_nosync

This used to disables immediately.

```
void tasklet_disable_nosync(struct tasklet_struct *t);
```

`t` – pointer to the tasklet struct

NOTE: If the tasklet has been disabled, we can still add it to the queue for scheduling, but it will not be executed on the CPU until it is enabled again. Moreover, if the tasklet has been disabled several times, it should be enabled exactly the same number of times, there is the count field in the structure or this purpose.

Schedule the tasklet

When we schedule the tasklet, then that tasklet is placed into one queue out of two, depending on the priority. Queues are organized as singly-linked lists. At that, each CPU has its own queues.

There are two priorities.

1. Normal Priority
2. High Priority

tasklet_schedule

Schedule a tasklet with normal priority. If a tasklet has previously been scheduled (but not yet run), the new schedule will be silently discarded.

void tasklet_schedule (struct tasklet_struct *t);

t – pointer to the tasklet struct

Example

```
/*Scheduling Task to Tasklet*/
tasklet_schedule(&tasklet);
```

tasklet_hi_schedule

Schedule a tasklet with high priority. If a tasklet has previously been scheduled (but not yet run), the
new schedule will be silently discarded.

void tasklet_hi_schedule (struct tasklet_struct *t);

t – pointer to the tasklet struct

tasklet_hi_schedule_first

This version avoids touching any other tasklets. Needed for kmemcheck in order not to take any page faults while enqueueing this tasklet. Consider VERY carefully whether you really need this or **tasklet_hi_schedule()**.

void tasklet_hi_schedule_first(struct tasklet_struct *t);

t – pointer to the tasklet struct

Kill Tasklet

1

Finally, after a tasklet has been created, it's possible to delete a tasklet through these below functions.

tasklet_kill

This will wait for its completion, and then kill it.

void tasklet_kill(struct tasklet_struct *t);

`t` – pointer to the tasklet struct

Example

```
/*Kill the Tasklet */
tasklet_kill(&tasklet);
```

tasklet_kill_immediate

This is used only when a given CPU is in the dead state.

void tasklet_kill_immediate(struct tasklet_struct *t, unsigned int cpu);

t – pointer to the tasklet struct

cpu – cpu num

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, 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>
```

```c
#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>


#define IRQ_NO 11

void tasklet_fn(unsigned long);

/* Init the Tasklet by Static Method */
DECLARE_TASKLET(tasklet,tasklet_fn, 1);


/*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)
{
        printk(KERN_INFO "Read function\n");
        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 *)(ir
            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_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 1");
MODULE_VERSION("1.15");
```

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

```
inux@embetronicx-VirtualBox: dmesg

[ 8592.698763] Major = 246 Minor = 0
[ 8592.703380] Device Driver Insert...Done!!!
[ 8601.716673] Device File Opened...!!!
[ 8601.716697] Read function
[ 8601.716727] Shared IRQ: Interrupt Occurred
[ 8601.716732] Executing Tasklet Function : arg = 1
[ 8601.716741] Device File Closed...!!!
[ 8603.916741] Device Driver Remove...Done!!!
```

- We can able to see the print "**Shared IRQ: Interrupt Occurred**" and "**Executing Tasklet Function : arg = 1**"
- Unload the module using **`sudo rmmod driver`**

In our next tutorial we will discuss Tasklet using Dynamic Method.

5

Article Rating

★★★★★

Share this:

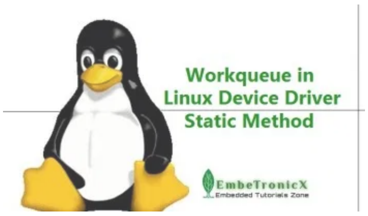
Share 2 Post Tweet SHARE Print WhatsApp Share

2 Email Telegram More

Like this:

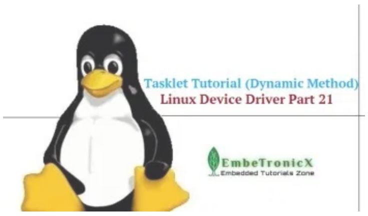
Loading...

Related



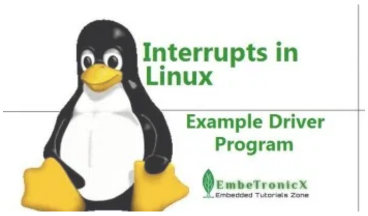
Linux Device Driver Tutorial Part 14 – Workqueue in Linux Kernel Part 1

In "Device Drivers"



Linux Device Driver Tutorial Part 21 – Tasklets | Dynamic Method

In "Device Drivers"



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

In "Device Drivers"

1

Subscribe ▾ Connect with | Login

Join the discussion

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

1 COMMENT

Oldest ▾

madhab

August 16, 2019 2:03 AM

i am not understanding the difference between "tasklet_schedule" and "tasklet_hi_schedule", because both deffinations are same. Can you explain it please.

Loading...

0 Reply

1