EmbeTronicX
Embedded Tutorials Zone

Sidebar▼

Home → Tutorials → Linux → Device Drivers → **Linux Device Driver Tutorial Part 21 – Tasklets | Dynamic Method**

Device Drivers

Tasklet Tutorial (Dynamic Method)
Linux Device Driver Part 21
EmbeTronicX
Embedded Tutorials Zone

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

Post Contents [hide]

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

```
/* 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 the background. When I call the function like above, it assigns the parameter to the passed tasklet structure. It will be looks like below.

```
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 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.

```
#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>


#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)
{
        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 "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 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.

0

Article Rating

☆☆☆☆☆

Share this:

Share 4 Post Tweet SHARE Print WhatsApp Share

0 Email Telegram More

Like this:

Loading...

Related

Linux Device Driver

Linux Device Driver

Linux Device Driver

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

2

Join the discussion

B *I* U ~~S~~ {} [+]

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

2