Linux Device Driver Tutorial Part 21 – Tasklets | Dynamic Method

Post Contents [hide]

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
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 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 ( structtasklet_struct *t,
 void(*)(unsigned long) func,
 unsigned long data
)
```

Where,

t – taskletstruct that should be initialized
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”.

Example

```
/* Tasklet by Dynamic Method */
structtasklet_struct *tasklet;

/* Init the taskletbt Dynamic Method */
tasklet  =kmalloc(sizeof(structtasklet_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.

```
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). 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

voidtasklet_fn(unsigned long);

/* Tasklet by Dynamic Method */
structtasklet_struct *tasklet;


/*Tasklet Function*/
voidtasklet_fn(unsigned long arg)
{
printk(KERN_INFO "Executing Tasklet Function : arg = %ld\n", arg);
}


//Interrupt handler for IRQ 11.
```

```
staticirqreturn_tirq_handler(intirq,void *dev_id) {
printk(KERN_INFO "Shared IRQ: Interrupt Occurred");
      /*Scheduling Task to Tasklet*/
tasklet_schedule(tasklet);

return IRQ_HANDLED;
}


volatileintetx_value = 0;


dev_tdev = 0;
staticstruct class *dev_class;
staticstructcdevetx_cdev;
structkobject *kobj_ref;

staticint __initetx_driver_init(void);
static void __exit etx_driver_exit(void);

/*************** Driver Fuctions **********************/
staticintetx_open(structinode *inode, struct file *file);
staticintetx_release(structinode *inode, struct file *file);
staticssize_tetx_read(struct file *filp,
char __user *buf, size_tlen,loff_t * off);
staticssize_tetx_write(struct file *filp,
const char *buf, size_tlen, loff_t * off);

/*************** SysfsFuctions **********************/
staticssize_tsysfs_show(structkobject *kobj,
structkobj_attribute *attr, char *buf);
staticssize_tsysfs_store(structkobject *kobj,
structkobj_attribute *attr,const char *buf, size_t count);

structkobj_attributeetx_attr = __ATTR(etx_value, 0660, sysfs_show, sysfs_store);

staticstructfile_operations fops =
{
      .owner         = THIS_MODULE,
      .read          = etx_read,
      .write         = etx_write,
      .open          = etx_open,
      .release       = etx_release,
};

staticssize_tsysfs_show(structkobject *kobj,
structkobj_attribute *attr, char *buf)
{
```

```
printk(KERN_INFO "Sysfs - Read!!!\n");
returnsprintf(buf, "%d", etx_value);
}

staticssize_tsysfs_store(structkobject *kobj,
structkobj_attribute *attr,const char *buf, size_t count)
{
printk(KERN_INFO "Sysfs - Write!!!\n");
sscanf(buf,"%d",&etx_value);
return count;
}

staticintetx_open(structinode *inode, struct file *file)
{
printk(KERN_INFO "Device File Opened...!!!\n");
return 0;
}

staticintetx_release(structinode *inode, struct file *file)
{
printk(KERN_INFO "Device File Closed...!!!\n");
return 0;
}

staticssize_tetx_read(struct file *filp,
char __user *buf, size_tlen, loff_t *off)
{
printk(KERN_INFO "Read function\n");
asm("int $0x3B");  // Corresponding to irq 11
return 0;
}
staticssize_tetx_write(struct file *filp,
const char __user *buf, size_tlen, loff_t *off)
{
printk(KERN_INFO "Write Function\n");
return 0;
}


staticint __initetx_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));
```

```
        /*Creatingcdev 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");
gotor_class;
        }

        /*Creatingstruct class*/
if((dev_class = class_create(THIS_MODULE,"etx_class")) == NULL){
printk(KERN_INFO "Cannot create the struct class\n");
gotor_class;
        }

        /*Creating device*/
if((device_create(dev_class,NULL,dev,NULL,"etx_device")) == NULL){
printk(KERN_INFO "Cannot create the Device 1\n");
gotor_device;
        }

        /*Creating a directory in /sys/kernel/ */
kobj_ref = kobject_create_and_add("etx_sysfs",kernel_kobj);

        /*Creatingsysfs file for etx_value*/
if(sysfs_create_file(kobj_ref,&etx_attr.attr)){
printk(KERN_INFO"Cannot create sysfs file......\n");
gotor_sysfs;
        }
if (request_irq(IRQ_NO, irq_handler, IRQF_SHARED, "etx_device", (void *)(irq_handler))) {
printk(KERN_INFO "etx_device: cannot register IRQ ");
gotoirq;
        }

        /* Init the taskletbt Dynamic Method */
tasklet  =kmalloc(sizeof(structtasklet_struct),GFP_KERNEL);
if(tasklet == NULL) {
printk(KERN_INFO "etx_device: cannot allocate Memory");
gotoirq;
        }
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 sudoinsmoddriver.ko
- To trigger 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 sudormmod driver