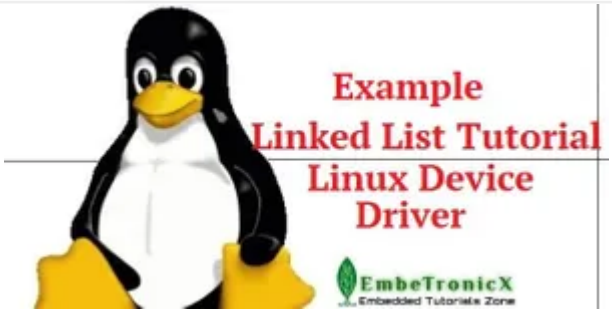
EmbeTronicX
Embedded Tutorials Zone

Sidebar

Home → Tutorials → Linux → Device Drivers → **Linux Device Driver Tutorial Part 18 – Linked List in Linux Kernel Part 2**

Device Drivers



Linux Device Driver Tutorial Part 18 – Linked List in Linux Kernel Part 2

This is the Series on Linux Device Driver. The aim of this series is to provide easy and practical examples that anyone can understand. In our previous tutorial, we have seen the functions used in Kernel Linked List. So this is the Linux Device Driver Tutorial Part 18 – Example Linked List in Linux Kernel which is the continuation (Part 2) of the Previous Tutorial.

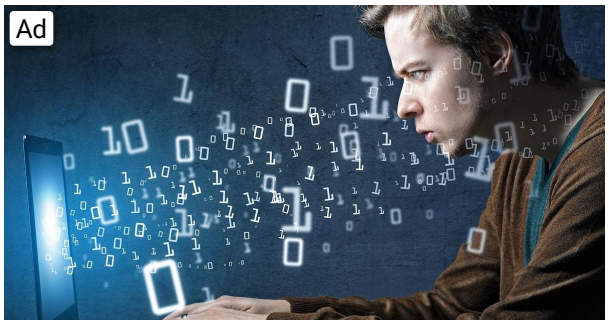
1

Apple Music Turns 5 as It Continues Rivalry With Spotify

Post Contents [hide]

Linux Device Driver Tutorial Part 18 – Example Linked List in Linux Kernel

If you don’t know the functions used in the linked list, please refer to this previous tutorial for the detailed explanation about all linked list functions.



Creation of web and network

NodeJS has an built in feature as HTTP

nodejsscripts.net

So now we can directly enter into the Linux Linked List Kernel programming. I took the source code form the previous tutorial. First, I will explain how this code works.

1. When we write the value to our device file using `echo value > /dev/etx_value`, it will invoke the interrupt. Because we configured the interrupt by using the software. If you don't know how it works, please refer to this tutorial.
2. The interrupt will invoke the ISR function.
3. In ISR we are allocating work to the Workqueue.
4. Whenever Workqueue executing, we are creating the Linked List Node and adding the Node to the Linked List.
5. When we are reading the driver using `cat /dev/etx_device`, printing all the nodes which are present in the Linked List using traverse.
6. When we are removing the driver using `rmmod`, it will remove all the nodes in Linked List and free the memory.

Note: We are not using the sysfs functions. So I kept empty sysfs functions.

Creating Head Node

```
1 /*Declare and init the head node of the linked list*/
2 LIST_HEAD(Head_Node);
```

This will create the head node in the name of **Head_Node** and initialize that.

Creating Node and add that into Linked List

```
1          /*Creating Node*/
2          temp_node = kmalloc(sizeof(struct my_list), GFP_KERNEL);
3
4          /*Assgin the data that is received*/
5          temp_node->data = etx_value;
6
7          /*Init the list within the struct*/
8          INIT_LIST_HEAD(&temp_node->list);
9
10         /*Add Node to Linked List*/
11         list_add_tail(&temp_node->list, &Head_Node);
```

This will create the node, assign the data to its members. Then finally add that node to the Linked List using **list_add_tail**. (*This part will be present in the workqueue function*)

Traversing Linked List

```
1          struct my_list *temp;
2          int count = 0;
3          printk(KERN_INFO "Read function\n");
4
5          /*Traversing Linked List and Print its Members*/
6          list_for_each_entry(temp, &Head_Node, list) {
7              printk(KERN_INFO "Node %d data = %d\n", count++, temp->data);
8          }
9
10         printk(KERN_INFO "Total Nodes = %d\n", count);
```

Here, we are traversing each node using **list_for_each_entry** and print those values. (*This part will be present in the read function*)

Deleting Linked List

```
1         /* Go through the list and free the memory. */
2         struct my_list *cursor, *temp;
3         list_for_each_entry_safe(cursor, temp, &Head_Node, list) {
4             list_del(&cursor->list);
```

```
5             kfree(cursor);
6         }
```

This will traverse each node using **list_for_each_entry_safe** and delete that using `list_del`. Finally, we need to free the memory which is allocated using **kmalloc**.

Programming

Driver Source Code

```
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 #include <linux/workqueue.h>            // Required for workqueues
15
16
17 #define IRQ_NO 11
18
19 volatile int etx_value = 0;
20
21 dev_t dev = 0;
22 static struct class *dev_class;
23 static struct cdev etx_cdev;
24 struct kobject *kobj_ref;
25
26 static int __init etx_driver_init(void);
27 static void __exit etx_driver_exit(void);
28
29 static struct workqueue_struct *own_workqueue;
30
31
32 static void workqueue_fn(struct work_struct *work);
33
34 static DECLARE_WORK(work, workqueue_fn);
35
36 /*Linked List Node*/
37 struct my_list{
38     struct list_head list;     //linux kernel list implementation
39     int data;
40 };
41
42 /*Declare and init the head node of the linked list*/
43 LIST_HEAD(Head_Node);
44
45 /*************** Driver Fuctions **********************/
46 static int etx_open(struct inode *inode, struct file *file);
```

```
47  static int etx_release(struct inode *inode, struct file *file);
48  static ssize_t etx_read(struct file *filp,
49                  char __user *buf, size_t len,loff_t * off);
50  static ssize_t etx_write(struct file *filp,
51                  const char *buf, size_t len, loff_t * off);
52
53  /*************** Sysfs Fuctions **********************/
54  static ssize_t sysfs_show(struct kobject *kobj,
55                  struct kobj_attribute *attr, char *buf);
56  static ssize_t sysfs_store(struct kobject *kobj,
57                  struct kobj_attribute *attr,const char *buf, size_t count);
58
59  struct kobj_attribute etx_attr = __ATTR(etx_value, 0660, sysfs_show, sysfs_store);
60  /*****************************************************/
61
62
63  /*Workqueue Function*/
64  static void workqueue_fn(struct work_struct *work)
65  {
66          struct my_list *temp_node = NULL;
67
68          printk(KERN_INFO "Executing Workqueue Function\n");
69
70          /*Creating Node*/
71          temp_node = kmalloc(sizeof(struct my_list), GFP_KERNEL);
72
73          /*Assgin the data that is received*/
74          temp_node->data = etx_value;
75
76          /*Init the list within the struct*/
77          INIT_LIST_HEAD(&temp_node->list);
78
79          /*Add Node to Linked List*/
80          list_add_tail(&temp_node->list, &Head_Node);
81  }
82
83
84  //Interrupt handler for IRQ 11.
85  static irqreturn_t irq_handler(int irq,void *dev_id) {
86          printk(KERN_INFO "Shared IRQ: Interrupt Occurred\n");
87          /*Allocating work to queue*/
88          queue_work(own_workqueue, &work);
89
90          return IRQ_HANDLED;
91  }
92
93  static struct file_operations fops =
94  {
95          .owner          = THIS_MODULE,
96          .read           = etx_read,
97          .write          = etx_write,
98          .open           = etx_open,
99          .release        = etx_release,
100 };
101
102 static ssize_t sysfs_show(struct kobject *kobj,
103                 struct kobj_attribute *attr, char *buf)
104 {
105         printk(KERN_INFO "Sysfs - Read!!!\n");
106         return sprintf(buf, "%d", etx_value);
107 }
108
109 static ssize_t sysfs_store(struct kobject *kobj,
```

```
110                 struct kobj_attribute *attr,const char *buf, size_t count)
111 {
112         printk(KERN_INFO "Sysfs - Write!!!\n");
113         return count;
114 }
115
116 static int etx_open(struct inode *inode, struct file *file)
117 {
118         printk(KERN_INFO "Device File Opened...!!!\n");
119         return 0;
120 }
121
122 static int etx_release(struct inode *inode, struct file *file)
123 {
124         printk(KERN_INFO "Device File Closed...!!!\n");
125         return 0;
126 }
127
128 static ssize_t etx_read(struct file *filp,
129                 char __user *buf, size_t len, loff_t *off)
130 {
131         struct my_list *temp;
132         int count = 0;
133         printk(KERN_INFO "Read function\n");
134
135         /*Traversing Linked List and Print its Members*/
136         list_for_each_entry(temp, &Head_Node, list) {
137             printk(KERN_INFO "Node %d data = %d\n", count++, temp->data);
138         }
139
140         printk(KERN_INFO "Total Nodes = %d\n", count);
141         return 0;
142 }
143 static ssize_t etx_write(struct file *filp,
144                 const char __user *buf, size_t len, loff_t *off)
145 {
146         printk(KERN_INFO "Write Function\n");
147         /*Copying data from user space*/
148         sscanf(buf,"%d",&etx_value);
149         /* Triggering Interrupt */
150         asm("int $0x3B");  // Corresponding to irq 11
151         return len;
152 }
153
154
155 static int __init etx_driver_init(void)
156 {
157         /*Allocating Major number*/
158         if((alloc_chrdev_region(&dev, 0, 1, "etx_Dev")) <0){
159                 printk(KERN_INFO "Cannot allocate major number\n");
160                 return -1;
161         }
162         printk(KERN_INFO "Major = %d Minor = %d n",MAJOR(dev), MINOR(dev));
163
164         /*Creating cdev structure*/
165         cdev_init(&etx_cdev,&fops);
166
167         /*Adding character device to the system*/
168         if((cdev_add(&etx_cdev,dev,1)) < 0){
169             printk(KERN_INFO "Cannot add the device to the system\n");
170             goto r_class;
171         }
172
```

```
173         /*Creating struct class*/
174         if((dev_class = class_create(THIS_MODULE,"etx_class")) == NULL){
175             printk(KERN_INFO "Cannot create the struct class\n");
176             goto r_class;
177         }
178 
179         /*Creating device*/
180         if((device_create(dev_class,NULL,dev,NULL,"etx_device")) == NULL){
181             printk(KERN_INFO "Cannot create the Device \n");
182             goto r_device;
183         }
184 
185         /*Creating a directory in /sys/kernel/ */
186         kobj_ref = kobject_create_and_add("etx_sysfs",kernel_kobj);
187 
188         /*Creating sysfs file*/
189         if(sysfs_create_file(kobj_ref,&etx_attr.attr)){
190                 printk(KERN_INFO"Cannot create sysfs file......\n");
191                 goto r_sysfs;
192         }
193         if (request_irq(IRQ_NO, irq_handler, IRQF_SHARED, "etx_device", (void *)(ir
194             printk(KERN_INFO "my_device: cannot register IRQ \n");
195                     goto irq;
196         }
197 
198         /*Creating workqueue */
199         own_workqueue = create_workqueue("own_wq");
200 
201         printk(KERN_INFO "Device Driver Insert...Done!!!\n");
202     return 0;
203 
204 irq:
205         free_irq(IRQ_NO,(void *)(irq_handler));
206 
207 r_sysfs:
208         kobject_put(kobj_ref);
209         sysfs_remove_file(kernel_kobj, &etx_attr.attr);
210 
211 r_device:
212         class_destroy(dev_class);
213 r_class:
214         unregister_chrdev_region(dev,1);
215         cdev_del(&etx_cdev);
216         return -1;
217 }
218 
219 void __exit etx_driver_exit(void)
220 {
221 
222         /* Go through the list and free the memory. */
223         struct my_list *cursor, *temp;
224         list_for_each_entry_safe(cursor, temp, &Head_Node, list) {
225             list_del(&cursor->list);
226             kfree(cursor);
227         }
228 
229         /* Delete workqueue */
230         destroy_workqueue(own_workqueue);
231         free_irq(IRQ_NO,(void *)(irq_handler));
232         kobject_put(kobj_ref);
233         sysfs_remove_file(kernel_kobj, &etx_attr.attr);
234         device_destroy(dev_class,dev);
235         class_destroy(dev_class);
```

```
236         cdev_del(&etx_cdev);
237         unregister_chrdev_region(dev, 1);
238         printk(KERN_INFO "Device Driver Remove...Done!!\n");
239 }
240
241 module_init(etx_driver_init);
242 module_exit(etx_driver_exit);
243
244 MODULE_LICENSE("GPL");
245 MODULE_AUTHOR("EmbeTronicX <embetronicx@gmail.com or admin@embetronicx.com>");
246 MODULE_DESCRIPTION("A simple device driver - Kernel Linked List");
247 MODULE_VERSION("1.13");
```

MakeFile

```
1 obj-m += driver.o
2
3 KDIR = /lib/modules/$(shell uname -r)/build
4
5 all:
6     make -C $(KDIR) M=$(shell pwd) modules
7
8 clean:
9     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**
- **sudo su**
- To trigger the interrupt read device file (**cat /dev/etx_device**)
- Now see the Dmesg (**dmesg**)

```
[ 5310.125001] Major = 246 Minor = 0 n
[ 5310.133127] Device Driver Insert...Done!!!
[ 5346.839872] Device File Opened...!!!
[ 5346.839950] Read function
[ 5346.839954] Total Nodes = 0
[ 5346.839982] Device File Closed...!!!
```

- By this time there are no nodes available.
- So now write the value to driver using **echo 10 > /dev/etx_device**
- By this time, One node has been added to the linked list.
- To test that read the device file using **cat /dev/etx_device**
- Now see the Dmesg (**dmesg**)

```
[ 5346.839982] Device File Closed...!!!
[ 5472.408239] Device File Opened...!!!
[ 5472.408266] Write Function
[ 5472.408293] Shared IRQ: Interrupt Occurred
```

```
[ 5472.408309] Device File Closed...!!!
[ 5472.409037] Executing Workqueue Function
[ 5551.996018] Device File Opened...!!!
[ 5551.996040] Read function
[ 5551.996044] Node 0 data = 10
[ 5551.996046] Total Nodes = 1
[ 5551.996052] Device File Closed...!!!
```

- Our value has added to the list.
- You can also write many times to create and add the node to the linked list
- Unload the module using **`rmmod driver`**

In our next tutorial, we will discuss kernel threads.

0

Article Rating

Share this:

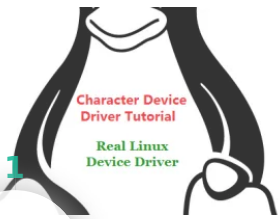
Share 0 Post Tweet SHARE Print WhatsApp Share

2 Email Telegram More

Like this:

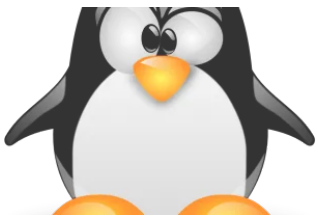
Loading...

Related



Linux Device Driver Tutorial Part 7 - Linux Device Driver Tutorial Programming
In "Device Drivers"

Linux Device Driver Tutorial Part 17 - Linked List in Linux Kernel Part 1
In "Device Drivers"



Linux Device Driver Tutorial Part 2 - First Linux Device Driver
In "Device Drivers"

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 13, 2019 2:30 AM

where can i find my inserted values except dmesg ?
and where and when we should use this linux list ?

Loading...

0 Reply

report this ad

1