# Part 18 – Example Linked List in Linux Kernel

[ <https://embetronicx.com/tutorials/linux/device-drivers/example-linked-list-in-linux-kernel/> ]

Please refer [this previous tutorial](https://www.embetronicx.com/tutorials/linux/device-drivers/linux-device-driver-tutorial-part-17-linked-list-in-linux-kernel/) for the detailed explanation about all linked list functions.

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 software. If you don’t know how it works, Please [refer this tutorial](https://www.embetronicx.com/tutorials/linux/device-drivers/linux-device-driver-tutorial-part-13-interrupt-example-program-in-linux-kernel/).
2. Interrupt will invoke the ISR function.
3. In ISR we are allocating work to the Workqueue.
4. Whenever Workqueue executing, we are creating 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 is present in the Linked List using traverse.
6. When we are removing the driver using rmmod, it will removes 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

/\*Declare and init the head node of the linked list\*/

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

/\*Creating Node\*/

temp\_node = kmalloc(sizeof(struct my\_list), GFP\_KERNEL);

/\*Assgin the data that is received\*/

temp\_node->data = etx\_value;

/\*Init the list within the struct\*/

INIT\_LIST\_HEAD(&temp\_node->list);

/\*Add Node to Linked List\*/

list\_add\_tail(&temp\_node->list, &Head\_Node);

This will create the node, assign the data to its member. 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

struct my\_list \*temp;

int count = 0;

printk(KERN\_INFO "Read function\n");

/\*Traversing Linked List and Print its Members\*/

list\_for\_each\_entry(temp, &Head\_Node, list) {

printk(KERN\_INFO "Node %d data = %d\n", count++, temp->data);

}

printk(KERN\_INFO "Total Nodes = %d\n", count);

Here, we are traversing each nodes using list\_for\_each\_entryand print those values. (*This part will be present in the read function*)

# Deleting Linked List

/\* Go through the list and free the memory. \*/

struct my\_list \*cursor, \*temp;

list\_for\_each\_entry\_safe(cursor, temp, &Head\_Node, list) {

list\_del(&cursor->list);

kfree(cursor);

}

# Programming

## Driver Source Code

#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>

#include <linux/workqueue.h> // Required for workqueues

#include <asm/hw\_irq.h>

#define IRQ\_NO 11

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);

static struct workqueue\_struct \*own\_workqueue;

static void workqueue\_fn(struct work\_struct \*work);

static DECLARE\_WORK(work, workqueue\_fn);

/\*Linked List Node\*/

struct my\_list{

struct list\_head list; //linux kernel list implementation

int data;

};

/\*Declare and init the head node of the linked list\*/

LIST\_HEAD(Head\_Node);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 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);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*Workqueue Function\*/

static void workqueue\_fn(struct work\_struct \*work)

{

struct my\_list \*temp\_node = NULL;

printk(KERN\_INFO "Executing Workqueue Function\n");

/\*Creating Node\*/

temp\_node = kmalloc(sizeof(struct my\_list), GFP\_KERNEL);

/\*Assgin the data that is received\*/

temp\_node->data = etx\_value;

/\*Init the list within the struct\*/

INIT\_LIST\_HEAD(&temp\_node->list);

/\*Add Node to Linked List\*/

list\_add\_tail(&temp\_node->list, &Head\_Node);

}

//Interrupt handler for IRQ 11.

static irqreturn\_t irq\_handler(int irq,void \*dev\_id) {

printk(KERN\_INFO "Shared IRQ: Interrupt Occurred\n");

/\*Allocating work to queue\*/

queue\_work(own\_workqueue, &work);

return IRQ\_HANDLED;

}

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");

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)

{

struct my\_list \*temp;

int count = 0;

printk(KERN\_INFO "Read function\n");

/\*Traversing Linked List and Print its Members\*/

list\_for\_each\_entry(temp, &Head\_Node, list) {

printk(KERN\_INFO "Node %d data = %d\n", count++, temp->data);

}

printk(KERN\_INFO "Total Nodes = %d\n", count);

return 0;

}

static ssize\_t etx\_write(struct file \*filp,

const char \_\_user \*buf, size\_t len, loff\_t \*off)

{

struct irq\_desc \*desc;

printk(KERN\_INFO "Write Function\n");

/\* New way of mapping irq lines on newer kernels \*/

desc = irq\_to\_desc(11);

if (!desc)

return -EINVAL;

\_\_this\_cpu\_write(vector\_irq[59], desc);

/\* Triggering Interrupt \*/

asm("int $0x3B"); // Corresponding to irq 11

return len;

}

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 \n");

goto r\_device;

}

/\*Creating a directory in /sys/kernel/ \*/

kobj\_ref = kobject\_create\_and\_add("etx\_sysfs",kernel\_kobj);

/\*Creating sysfs file\*/

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 \*)(irq\_handler))) {

printk(KERN\_INFO "my\_device: cannot register IRQ \n");

goto irq;

}

/\*Creating workqueue \*/

own\_workqueue = create\_workqueue("own\_wq");

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)

{

/\* Go through the list and free the memory. \*/

struct my\_list \*cursor, \*temp;

list\_for\_each\_entry\_safe(cursor, temp, &Head\_Node, list) {

list\_del(&cursor->list);

kfree(cursor);

}

/\* Delete workqueue \*/

destroy\_workqueue(own\_workqueue);

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 or admin@embetronicx.com>");

MODULE\_DESCRIPTION("A simple device driver - Kernel Linked List");

MODULE\_VERSION("1.13");

## 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
* sudo su
* To trigger 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 is 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 linked list
* Unload the module using rmmod driver