==========

1.SIMPLE_DEVICE_DRIVER_CODE

```
#include <linux/kernel.h>
#include <linux/module.h>

int init_module(void)
{
        printk(KERN_INFO"This is the simple kenel program\n");
        return 0;
}

void cleanup_module(void)
{
        printk(KERN_INFO"Bye...\n");
}

MODULE_LICENSE("Dual BSD/GPL");
```

==========

2.VARIBLE_DEVICE_DRIVER_CODE

```
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/stat.h>
#include <linux/moduleparam.h>
#include <linux/init.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("SRIRAM");

static short int myshort = 1;

static int myint = 420;

static long int mylong = 9999;

static char *mystring = "hello";

static int myintarray[2] = {-1,-1};
static int arr_argc = 0;

module_param(myshort,short,S_IRUSR|S_IWUSR|S_IRGRP|S_IWGRP);
MODULE_PARM_DESC(myshort,"A short integer");

module_param(myint,int,S_IRUSR|S_IWUSR|S_IRGRP|S_IROTH);
MODULE_PARM_DESC(myint,"An integer");

module_param(mylong,long,S_IRUSR);
MODULE_PARM_DESC(mylong,"A long integer");

module_param(mystring,charp,0000);
MODULE_PARM_DESC(mystring,"A character string");
```

```
module_param_array(myintarray,int,&arr_argc,0000);
MODULE_PARM_DESC(myintarray,"A array of integers");

static int __init start(void)
{
	int i;
	printk(KERN_INFO"Hello,World 5\n==========\n");
	printk(KERN_INFO"myshort is a short integer: %d\n",myshort);
	printk(KERN_INFO"myint is an integer: %d\n",myint);
	printk(KERN_INFO"mylong is a long integer: %ld\n",mylong);
	printk(KERN_INFO"mystring is a string: %s\n",mystring);
	for(i=0;i < (sizeof myintarray / sizeof (int));i++)
	{
		printk(KERN_INFO"myintarray[%d] = %d\n",i,myintarray[i]);
	}
	printk(KERN_INFO"got %d argument for myintarray.\n",arr_argc);
	return 0;
}

static void __exit stop(void)
{
	printk(KERN_INFO"Goodbye,world \n");
}
module_init(start);
module_exit(stop);
```

==========

3.a.SYMBOLTABLE_DEVICE_DRIVER_CODE

```
#include <linux/kernel.h>
#include <linux/module.h>

static int value=555;
static void func(void);

EXPORT_SYMBOL_GPL(value);
EXPORT_SYMBOL_GPL(func);

static void func(void)
{
	printk(KERN_INFO"The value is %d\n",value);
}

static int __init start(void)
{
	func();
	return 0;
}

static void __exit stop(void)
{
	printk(KERN_INFO"Bye...\n");
}
```

```
module_init(start);
module_exit(stop);

MODULE_LICENSE("Dual BSD/GPL");
```

3.b.SYMBOL_TABLE_CHANGE_DEVICE_DRIVER_CODE

```
#include <linux/kernel.h>
#include <linux/module.h>

static int *ptr;



static int __init start(void)
{
	ptr=(int*)__symbol_get("value");
	if(ptr)
	{
		*ptr=666;
		__symbol_put("value");
		return 0;
	}
	else
	{
		printk(KERN_INFO"This is value in the symbol\n");
		return -EINVAL;
	}
}

static void __exit stop(void)
{
	printk(KERN_INFO"Bye...\n");
}

module_init(start);
module_exit(stop);

MODULE_LICENSE("Dual BSD/GPL");
```

3.b.SYMBOL_DISPLAY_DEVICE_DRIVER_CODE

```
#include <linux/kernel.h>
#include <linux/module.h>

extern int value;
extern void func(void);

static int __init start(void)
{
	printk(KERN_INFO"The value is %d\n",value);
	func();
```

```
	return 0;
}

static void __exit stop(void)
{
	printk(KERN_INFO"Bye....\n");
}

module_init(start);
module_exit(stop);

MODULE_LICENSE("Dual BSD/GPL");
```

==========

4.PROESS_DEVICE_DRIVER_CODE

```
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/sched.h>

static int __init start(void)
{
	struct task_struct *task;
	for_each_process(task){
		printk(KERN_INFO"process name:%s,process id:%d,process state:%ld\n",task->comm,task->pid,task->state);
	};
	return 0;
}

static void __exit stop(void)
{
	printk(KERN_INFO"Bye.....\n");
}

module_init(start);
module_exit(stop);

MODULE_LICENSE("Dual BSD/GPL");
```

==========

5.ATOMIC_DEVICE_DRIVER_CODE

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <asm/atomic.h>

static int __init start(void)
{
	atomic_t av = ATOMIC_INIT (10);
	printk(KERN_INFO"The value of atomic is %d\n",atomic_read(&av));
	atomic_add(10,&av);
	printk(KERN_INFO"The value is %d\n",atomic_read(&av));
	atomic_sub(10,&av);
	printk(KERN_INFO"The value is %d\n",atomic_read(&av));
```

```
	atomic_inc(&av);
	printk(KERN_INFO"The value is %d\n",atomic_read(&av));
	atomic_dec(&av);
	printk(KERN_INFO"The value is %d\n",atomic_read(&av));
	return 0;
}

static void __exit stop(void)
{
	printk(KERN_INFO"Bye...\n");
}

module_init(start);
module_exit(stop);

MODULE_LICENSE("Dual BSD/GPL");
```

==========

6.INTERRUPT_DEVICE_DRIVER_CODE

```
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/interrupt.h>

#define SHARED_IRQ 28

static int irq = SHARED_IRQ;
module_param(irq,int,S_IRWXU);
MODULE_PARM_DESC(irq,"This is display in the /var/log/kern.log");

static int MY_ID=100;
static int irqcount = 0;

static irqreturn_t my_interrupt(int irq,void *dev_id)
{
	irqcount++;
	printk(KERN_INFO"The interrupt is occured these many times %d\n",irqcount);
	return IRQ_NONE;
}

static int __init start(void)
{
	if(request_irq(irq,my_interrupt,IRQF_SHARED,"my_interrupt",&MY_ID)< 0)
	{
		printk(KERN_INFO"Interrupt is not happened\n");
		return -1;
	}
	printk(KERN_INFO"The interrupt number in the table %d\n",irq);
	return 0;
}

static void __exit stop(void)
{
```

```
	free_irq(irq,&MY_ID);
	printk(KERN_INFO"Interrupt is removed from the table\n");
}

module_init(start);
module_exit(stop);

MODULE_LICENSE("Dual BSD/GPL");
```

==========

7.CHARACTER_DEVICE_DRIVER_CODE_MANUAL

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/types.h>
#include <linux/fs.h>
#include <linux/cdev.h>
#include <linux/slab.h>
#include <asm/current.h>
#include <asm/uaccess.h>

#define MAX_SIZE 2048
#define MAJORNO 300
#define MINORNO 0
#define CHAR_NAME "manual"

static dev_t first;
static int count = 1;
static struct cdev *mydev;
static char *kbuf;
static int inuse = 0;
static int balance = 0;

static int open(struct inode *inode, struct file *filp)
{
	inuse++;
	printk(KERN_INFO"This device is invoked by these many device %d\n",inuse);
	printk(KERN_INFO"The Majorno:%d and Minorid:%d\n",imajor(inode),iminor(inode));
	printk(KERN_INFO"process name:%s\t,process id:%d\t,process state:%ld\n",current-
>comm,current->pid,current->state);
	printk(KERN_INFO"The Refno=%d\n",module_refcount(THIS_MODULE));
	return 0;
}

static ssize_t write(struct file *filp, const char __user *buf, size_t ln, loff_t *pos)
{
	int nbytes,byte_do_to;
	balance = MAX_SIZE-*pos;
	if(ln < balance)
	{
		byte_do_to = ln;
	}
	else
```

```
	{
		byte_do_to = balance;
	}
	if(byte_do_to == 0)
	{
		printk(KERN_ERR"This device is reached the end\n");
		return -ENOSPC;
	}

	nbytes = byte_do_to - copy_from_user(kbuf+(*pos),buf,byte_do_to);
	*pos += nbytes;
	return nbytes;
}

static ssize_t read(struct file *filp, char __user *buf, size_t ln, loff_t *pos)
{
	int bytes_do_to,nbytes;
	balance=MAX_SIZE - *pos;
	if(ln < balance)
	{
		bytes_do_to = ln;
	}
	else
	{
		bytes_do_to = balance;
	}
	if(bytes_do_to == 0)
	{
		printk(KERN_ERR"This device is reached the end\n");
		return -ENOSPC;
	}

	nbytes = bytes_do_to - copy_to_user(buf,kbuf+(*pos),bytes_do_to);
	*pos += nbytes;
	return nbytes;
}

static int close(struct inode *inode, struct file *filp)
{
	printk(KERN_INFO"This device is ejected from the kernel\n");
	return 0;
}

static struct file_operations f_ops={
	.owner = THIS_MODULE,
	.open  = open,
	.write = write,
	.read  = read,
	.release=close,
};
```

```
static int __init start(void)
{
	first = MKDEV(MAJORNO,MINORNO);
	if(register_chrdev_region(first,count,CHAR_NAME) < 0)
	{
		printk(KERN_ERR"This deivce driver is not register in the kernel\n");
		return -1;
	}
	mydev=cdev_alloc();
	cdev_init(mydev,&f_ops);
	if(cdev_add(mydev,first,count) < 0)
	{
		unregister_chrdev_region(first,count);
		cdev_del(mydev);
		printk(KERN_ERR"This deivce not able to add to kernel\n");
	}
	kbuf=(char*)kzalloc(MAX_SIZE,GFP_KERNEL);
	printk(KERN_INFO"This device is sucessflly create with the name %s\n",CHAR_NAME);
	printk(KERN_INFO"The Majorno:%d and Minorno:%d\n",MAJOR(first),MINOR(first));
	printk(KERN_INFO"Process name:%s\t,process id:%d\t,process state:%ld\n",current-
>comm,current->pid,current->state);
	return 0;
}

static void __exit stop(void)
{
	kfree(kbuf);
	cdev_del(mydev);
	unregister_chrdev_region(first,count);
	printk(KERN_INFO"This device is removed from the kernel\n");
}

module_init(start);
module_exit(stop);

MODULE_LICENSE("Dual BSD/GPL");
```

==========

8.CHARACTER_DEVICE_DRIVER_CODE_AUTO

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/types.h>
#include <linux/fs.h>
#include <linux/cdev.h>
#include <linux/slab.h>
#include <linux/device.h>
#include <asm/current.h>
#include <asm/uaccess.h>

#define MAX_SIZE 2048
#define CHAR_NAME "auto"
```

```
static dev_t first;
static int count = 1;
static struct cdev *mydev;
static char *kbuf;
static int minor = 0;
static int inuse = 0;
static int balance = 0;
static struct class *cl;

static int open(struct inode *inode, struct file *filp)
{
	inuse++;
	printk(KERN_INFO"This device is invoked by these many device %d\n",inuse);
	printk(KERN_INFO"The Majorno:%d and Minorid:%d\n",imajor(inode),iminor(inode));
	printk(KERN_INFO"process name:%s\t,process id:%d\t,process state:%ld\n",current-
>comm,current->pid,current->state);
	printk(KERN_INFO"The Refno=%d\n",module_refcount(THIS_MODULE));
	return 0;
}

static ssize_t write(struct file *filp, const char __user *buf, size_t ln, loff_t *pos)
{
	int nbytes,byte_do_to;
	balance = MAX_SIZE-*pos;
	if(ln < balance)
	{
		byte_do_to = ln;
	}
	else
	{
		byte_do_to = balance;
	}
	if(byte_do_to == 0)
	{
		printk(KERN_ERR"This device is reached the end\n");
		return -ENOSPC;
	}

	nbytes = byte_do_to - copy_from_user(kbuf+(*pos),buf,byte_do_to);
	*pos += nbytes;
	return nbytes;
}

static ssize_t read(struct file *filp, char __user *buf, size_t ln, loff_t *pos)
{
	int bytes_do_to,nbytes;
	balance=MAX_SIZE - *pos;
	if(ln < balance)
	{
		bytes_do_to = ln;
	}
```

```
	else
	{
		bytes_do_to = balance;
	}
	if(bytes_do_to == 0)
	{
		printk(KERN_ERR"This device is reached the end\n");
		return -ENOSPC;
	}

	nbytes = bytes_do_to - copy_to_user(buf,kbuf+(*pos),bytes_do_to);
	*pos += nbytes;
	return nbytes;
}

static int close(struct inode *inode, struct file *filp)
{
	printk(KERN_INFO"This device is ejected from the kernel\n");
	return 0;
}

static struct file_operations f_ops={
	.owner = THIS_MODULE,
	.open  = open,
	.write = write,
	.read  = read,
	.release=close,
};

static int __init start(void)
{
	if(alloc_chrdev_region(&first,minor,count,CHAR_NAME) < 0)
	{
		printk(KERN_ERR"This deivce driver is not register in the kernel\n");
		return -1;
	}
	mydev=cdev_alloc();
	cdev_init(mydev,&f_ops);
	if(cdev_add(mydev,first,count) < 0)
	{
		unregister_chrdev_region(first,count);
		cdev_del(mydev);
		printk(KERN_ERR"This deivce not able to add to kernel\n");
	}
	cl=class_create(THIS_MODULE,"MY_DEVICE");
	device_create(cl,NULL,first,NULL,"%s",CHAR_NAME);
	kbuf=(char*)kzalloc(MAX_SIZE,GFP_KERNEL);
	printk(KERN_INFO"This device is sucessflly create with the name %s\n",CHAR_NAME);
	printk(KERN_INFO"The Majorno:%d and Minorno:%d\n",MAJOR(first),MINOR(first));
	printk(KERN_INFO"Process name:%s\t,process id:%d\t,process state:%ld\n",current-
>comm,current->pid,current->state);
```

```
	return 0;
}

static void __exit stop(void)
{
	kfree(kbuf);
	device_destroy(cl,first);
	class_destroy(cl);
	cdev_del(mydev);
	unregister_chrdev_region(first,count);
	printk(KERN_INFO"This device is removed from the kernel\n");
}

module_init(start);
module_exit(stop);

MODULE_LICENSE("Dual BSD/GPL");
```

==========

9.CHARACTER_DEVICE_DRIVER_CODE_LLSEEK

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/types.h>
#include <linux/fs.h>
#include <linux/cdev.h>
#include <linux/slab.h>
#include <linux/device.h>
#include <asm/current.h>
#include <asm/uaccess.h>

#define MAX_SIZE 2048
#define CHAR_NAME "llseek"

static dev_t first;
static int count = 1;
static struct cdev *mydev;
static char *kbuf;
static int minor = 0;
static int inuse = 0;
static int balance = 0;
static struct class *cl;

static int open(struct inode *inode, struct file *filp)
{
	inuse++;
	printk(KERN_INFO"This device is invoked by these many device %d\n",inuse);
	printk(KERN_INFO"The Majorno:%d and Minorid:%d\n",imajor(inode),iminor(inode));
	printk(KERN_INFO"process name:%s\t,process id:%d\t,process state:%ld\n",current-
>comm,current->pid,current->state);
	printk(KERN_INFO"The Refno=%d\n",module_refcount(THIS_MODULE));
	return 0;
}
```

```
static ssize_t write(struct file *filp, const char __user *buf, size_t ln, loff_t *pos)
{
	int nbytes,byte_do_to;
	balance = MAX_SIZE-*pos;
	if(ln < balance)
	{
		byte_do_to = ln;
	}
	else
	{
		byte_do_to = balance;
	}
	if(byte_do_to == 0)
	{
		printk(KERN_ERR"This device is reached the end\n");
		return -ENOSPC;
	}

	nbytes = byte_do_to - copy_from_user(kbuf+(*pos),buf,byte_do_to);
	*pos += nbytes;
	return nbytes;
}

static ssize_t read(struct file *filp, char __user *buf, size_t ln, loff_t *pos)
{
	int bytes_do_to,nbytes;
	balance=MAX_SIZE - *pos;
	if(ln < balance)
	{
		bytes_do_to = ln;
	}
	else
	{
		bytes_do_to = balance;
	}
	if(bytes_do_to == 0)
	{
		printk(KERN_ERR"This device is reached the end\n");
		return -ENOSPC;
	}

	nbytes = bytes_do_to - copy_to_user(buf,kbuf+(*pos),bytes_do_to);
	*pos += nbytes;
	return nbytes;
}

static loff_t seek(struct file *filp, loff_t pos, int lk)
{
	loff_t  testpos;
	switch(lk)
```

```
	{
		case 0:
			testpos=pos;
			break;
		case 1:
			testpos = filp->f_pos + pos;
			break;
		case 2:
			testpos = MAX_SIZE + pos;
			break;
		default:
			return -EINVAL;
	}
	testpos = testpos < MAX_SIZE ? testpos : MAX_SIZE;
	testpos = testpos < 0 ? 0 :testpos;
	printk(KERN_INFO"Seek is %ld\n",(long)testpos);
	filp->f_pos = testpos;
	return testpos;
}

static int close(struct inode *inode, struct file *filp)
{
	printk(KERN_INFO"This device is ejected from the kernel\n");
	return 0;
}

static struct file_operations f_ops={
	.owner = THIS_MODULE,
	.open  = open,
	.write = write,
	.read  = read,
	.llseek=seek,
	.release=close,
};

static int __init start(void)
{
	if(alloc_chrdev_region(&first,minor,count,CHAR_NAME) < 0)
	{
		printk(KERN_ERR"This deivce driver is not register in the kernel\n");
		return -1;
	}
	mydev=cdev_alloc();
	cdev_init(mydev,&f_ops);
	if(cdev_add(mydev,first,count) < 0)
	{
		unregister_chrdev_region(first,count);
		cdev_del(mydev);
		printk(KERN_ERR"This deivce not able to add to kernel\n");
	}
	cl=class_create(THIS_MODULE,"MY_DEVICE");
```

```
	device_create(cl,NULL,first,NULL,"%s",CHAR_NAME);
	kbuf=(char*)kzalloc(MAX_SIZE,GFP_KERNEL);
	printk(KERN_INFO"This device is sucessflly create with the name %s\n",CHAR_NAME);
	printk(KERN_INFO"The Majorno:%d and Minorno:%d\n",MAJOR(first),MINOR(first));
	printk(KERN_INFO"Process name:%s\t,process id:%d\t,process state:%ld\n",current-
>comm,current->pid,current->state);
	return 0;
}

static void __exit stop(void)
{
	kfree(kbuf);
	device_destroy(cl,first);
	class_destroy(cl);
	cdev_del(mydev);
	unregister_chrdev_region(first,count);
	printk(KERN_INFO"This device is removed from the kernel\n");
}

module_init(start);
module_exit(stop);

MODULE_LICENSE("Dual BSD/GPL");
```
==========