*IOSR Journal of Electronics and Communication Engineering (IOSR-JECE)*
*e-ISSN: 2278-2834,p- ISSN: 2278-8735.Volume 11, Issue 6, Ver. II (Nov.-Dec .2016), PP 46-52*
*www.iosrjournals.org*

# Coding for Pseudo Device by Linux Character Device Driver

Navneet Kr. Pandey[1], Prof. Prabhakar Dubey[2,] and Saurabh Bhutani[3 ,]
[1](M Tech Scholar Electronics Dept, RRIMT / Aktu, India)
[2](Prof. ECE,RRIMT/ Aktu,India)
[3](M Tech Scholar Electronics Dept, RRIMT / Aktu, India)

***Abstract:*** *Linux is a free Operating system and have major advantage that their internal are open for all to view. Usually OS (Operating system) has a dark and mysterious area whose code is restricted to small number of programmer's. Linux helped to democratize OS. Device Driver provide gateway to approach the code without being overwhelmed by complexity. To interact with hardware devices Device Driver is the most important software of OS. DD (DeviceDriver) must be reliable and efficient because any wrong operation can fatal system error and hardware performance depend on device driver.*

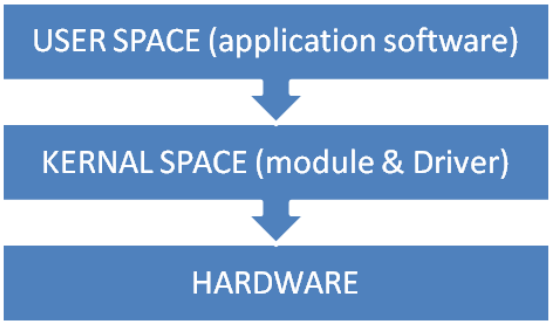
## I. Introduction

The DD is Integral Unit of OS and act as translator between Hardware device and the application of OS that uses it. Device Driver are the distinct BLACK BOX that make particular piece of hardware respond to well defined internal programming interface , they hide completely the details of how the device work. This programming interface is such that the driver can built separately from rest of the kernel and "plugged in" at the runtime when needed. This modularity make linux driver easy to write. LKM (Loadable kernel module) which facilitate to insert piece of code along with running module into the kernel at run time. Device Driver is a program that control particular type of device that is attached to our computer. There are device driver for printer/display/CD rom reader/diskette driver and so on. When we buy an OS many device driver are built into the product. However if we buy a new type of device that OS didn't recognize we will have to install new device driver. A DD is essentially convert the more general input, output instruction of OS to manage that device type can understand.

**Pre-Requisite-**
In order to develop Linux Device driver it is necessary to understand following

- C Programming: Deep knowledge of C programming is necessary specially function call, Structure, bit manipulation function and dynamic memory allocation
- Microprocessor programming: knowledge of tool chain and how microcomputer work internally, memory addressing and interrupt etc. All these concept is necessary to an assembler programmer.

Before writing a device driver programming it's important to make distinction between "userspace" & "kernel space".
**Userspace:** End user program like unix shell or other GUI
based application (kedit) are the part user space. Obviously these application need to interact with system hardware. However they don't perform directly but through kernel support function.
**Kernal space:** The kernel i.e. the core of OS and in particular its Device Driver form a bridge or interface between end user programmer and hardware.
Any subroutine or function forming part of the kernel are considered to be kernel space.



Classes of Device & Module: Linux classify the module into three fundamental categories which are

- CHAR MODULE: the device which can be accessed by stream of bytes (like a file). Such driver usually contain *open*, *close*, *read* and *write* system call.
- BLOCK MODULE: The device that can be accessed by filesystem. This module can handle I/O operation that transfer one or whole more blocks, which are usually 512 bytes in length.
- NETWORK INTERFACE: device that able to exchange data with other host.

Split view of kernel with process management, memory management, filesystem and process control along with drivers are as follows

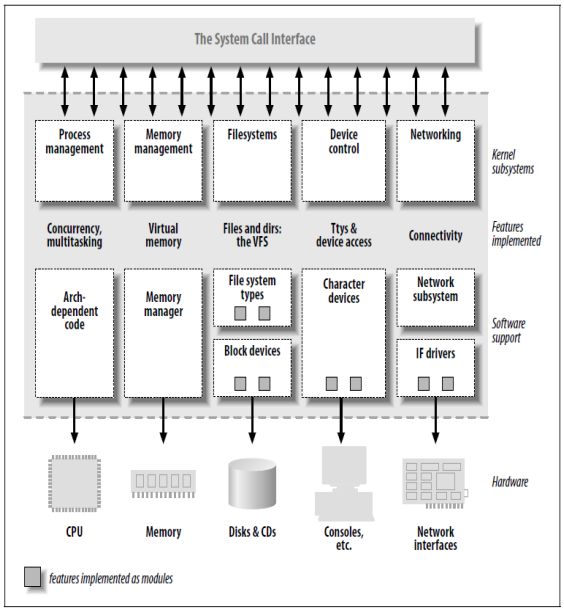



**Figure 1.**Split View of Kernel [1]

Loading and Unloading Device Driver

- INSMOD command is used to load the module into the running kernel. The river have to be in the form of module to be a part of kernel.
- RMMOD command is used to remove module from the kernel .

Software requirement and specification: Pentium machine or above
Linux 2.6.32-71.elf Environment/ any linux OS platform32 bit processor

- The coding contain implementation of Device driver for registering any device on kernel.
- We should properly compile the device driver code as it work along with the kernel any error may corrupt the OS.

***Appendix A1:***

Source code: FirstModule.c

```
  #include<linux/init.h>
  #include<linix/module.h>
   Static intnkp_init(void)
    {
     //code
Printk(KERN_INFO “HELLO WORLD..\n”);
   return(0);
    }
Static void nkp_exit(void)
    {
     //code
Printk(KERN_INFO “exiting from the MODULE..\n”);
   return(0);
    }
module_init(nkp_init);
module_exit(nkp_exit);
```

MODULE_LISCENSE("Dual BSD/GPL);
MODULE_AUTHOR("NAVNEET PANDEY");
MODULE_DESCRIPTION("First Kernel Module..");
Steps to execute:
1. Create a text file write a module code and name it as FirstModule.c
2. Create another text file with single line as shown under a name "MAKEFILE"
Obj-m :-FirstMoodule.o
3. To build our module/driverexecute following command..
# make –c /lib/module/'uname-r'/uild M='pwd' modules
Many files will be created and t will be visible by using command 'ls'
4. # su
5. To insert and load module "FirstModule.ko type
#insmodFirstModule.ko
6. Check whether our module is insered or loaded in the kernel by..
#lsmod | head -5 (modue name is seen)
7. To see the driver's loaded printk() message , use ..
#dmesg | tail -5 (display message)
8. To remove /unload "FirstModule.ko" module..
#rmmodFirstModule.ko (module name will be disappeared)
9. To go back user login type
# exit
10. To remove all files type..
# make –c /lib/module/'uname-r'/uild M='pwd' clean

***Appendix A2:***
Source code: Chardev.c

```
# include<linux/module.h>
#include<linux/version.h>
#include<linux/kernel.h>
#include<linux/types.h>
#include<linux/kdev_t.h>
#include<linux/fs.h>
#include<linux/device.h>
#include<linux/cdev.h>
#include<asm/uaccess.h>
#include<linux/sched.h>
//global variable declaration
static  dev_t  first;
staticstructcdevmy_char_dev;
staticstruct class *my_class_device_ptr;
#define MAXLEN 4000 // correspond to 4k memory page size
static char mybuffer[MAXLEN];
//global function declaration
static  intnkp_open(structinode *inode_ptr, struct file *file_ptr)
{
   //code
   Printk("KERN_INFO "Driver's is now open() : PID of
process using this device is  %d\n", current->pid );
return(0);
}
Static  ssize_tnkp_read(struct file *file_ptr, char_user *buffer , size_tlength,loff_t *offset)
{  // variable declartation
   intmax_bytes;
   intbytes_to_read;
   int bytes;
   //code
   max_bytes=MAXLEN-*offset;
   if(max_bytes>length)
   bytes_to_read=length;
```

```
   else
bytes_to_read=max_bytes;
if(bytes_to_read==0)
   {
   Printk(KERN_INFO "YOU CAN read() DRIVER'S MODULE : EOD (END OF DEVICE) \n)
Return(-ENOSPC);  //ERROR NO SPACE
}
   nbytes=bytes_to_read-copy_to_user(buffer,my+buffer *offset,bytes_to_read);
   return(nbytes);
}
staticssize_tnkp_write(struct file *file_ptr, const char _user *buffer,size_tlength,loff_t *offset)
{
   //variable declaration
   intmax_bytes;
   intbytes_to_write;
intnbytes;
   //code
   max_bytes=MAXLEN- *offset;
if(max_bytes>length)
         bytes_to_write=length;
   else
bytes_to_write=max_bytes;
if(bytes_to_write==0)
   {
   Printk(KERN_INFO "YOU CAN write() DRIVER'S MODULE : EOD (END OF DEVICE) \n)
return(-ENOSPC);  //ERROR NO SPACE
}
nbytes=bytes_to_write-copy_from_user(mybuffer+ *offset,buffer,bytes_to_write);
*offset=*offset+nbytes;
   return(nbytes);
staticloff_tnkp_lseek(struct file *file_ptr,loff_t offset, int origin)
{
   //variable declaration
   loff_tnew_pos=0;
   //code
   Switch(origin)
   {
      Case 0:
         new_pos=offset;
         Break;
      Case 1:
         new_pos=file_ptr->f_pos+offset;
         Break;
      Case 2:
         new_pos=MAXLEN-offset;
         Break;
   }
   if(new_pos>MAXLEN)
   new_pos=MAXLEN;
If(new_pos<0)
new_pos=0;
file_ptr->f_pos=new_pos;
return(new_pos);
}


// global structure/union/typedef/enum declarations
Static structfile_operationsnkp_fops={.owner=THIS_MODULE,
                  .open=nkp_open,
```

```
                .release=nkp_close,
                .read=nkp_read,
                .write=nkp_write
                .llseek=nkp_lseek};

staticint _initnkp_init(void)
{
   //code
int major=250; //hard coded : 250 may vary dynamically from system to system and hence we may get 251.
int minor=0; // hard coded.
first=MKDEV(major,minor);

if(alloc_chrdev_region(&first,0,1,"NKP")<0)
return(-1);

if(my_device_class_ptr=class_create(THIS MODULE,"nkp_chrdev_class"))== NULL)
{
unregister_chrdev_region(first,1);
return(-1);
}
//for linux kernel >2.6.26 :
device_create(my_device_class_ptr,NULL,first,NULL,"nkp_chardev")
// if used for linuxkernrl< 2.6.26,"warning: too many arguments for format"occurs
if(device_create(my_device_class_ptr,NULL,first,"nkp_chardev")==NULL)
{
class_destroy(my_device_class_ptr);
unregister_chrdev_region(first,1);
return(-1);
}
cdev_init(&my_char_dev,&nkp_fops);
if(cdev_add(&my_char_dev_first,1)==-1)
{
device_destroy(my_device_class_ptr,first);
class_destroy(my_device_class_ptr);
unregister_chrdev_region(first,1);
return(-1);
}
printk(KERN_INFO "Welcome By NKP : Linux Character Device is Successfully registered.\n");
return(0);
}
staticvoid_exitvdg_exit(void)
{
// code
cdev_del(&my_char_dev);
device_destroy(my_device_class_ptr,first);
class_destroy(my_device_class_ptr);
unregister_chrdev_region(first,1);
printk(KERN_INFO "GoodBYE By NKP : Linux Character Device Driver is Successfully Unregistered.\n");
}
module_init(nkp_init);
module_exit(nkp_exit);
MODULE_LICENSE("GPL");
MODULE_AUTHOR("NAVNEET PANDEY");
MODULE_DESCRIPTION("Linux Character Driver");
```

**Implementaton:**

**Layout of Driver-** The presently driver/module is to be added running kernel dynamically. After complilation**.ko** file created which is actual driver file created by 'modpost' by *insmod*utility

INSMOD:

- Through this command the module get path in running kernel and initialization of driver is done.
- Following function get active through this

cdev_init()
The functionality of this system call will initialize the character device
module_init()
alloc_chrdev_region()

RMMOD:

- device_destroy()
- class_destroy()
- unregister_chrdev_region()

**Steps to Insert the Module:**

- Create a text file, write a module's code and name it as "Chardev.c"
- Create anothr text file, type a single line as shown on following line and name it as "Makefile"
- obj-m :=Chardev.o
- To build our module/driver ( Chardev.ko ) execute following command …
- # make –C /lib/modules/ uname –r / build M='pwd' modules
- #su
- To insert/load "Chardev.ko" module …
- # insmodChardev.ko
- Check whether our device's device class directory is created or not by …
- # ls / sys/class
- In the output there is a name "nkp_character_class directory.
- Read device directory "dev" file …
- # cat/sys/class/nkp_chardev_class/nkp_chardev/dev
- Output shows our devices <major num,minornum> as 251:0

Even though we specified 250, it is showing 251. Because number 250 may vary from system to system and may not be allocated dynamically and still we may get 251.

- To cross check our device and its major , minor num, see it in /dev directory

#ls -1 /dev/nkp_chardev
Output shows…
Crw-rw---- 1 root root**251 ,0** 2016-29-9 15:34 /dev/nkp_chaedev
Where 'c' indicate character type device 251-major, 0- minornumber

- Check whether our module is inserted / loaded in kernel by..
- #lsmod | head -5
- Module name is seen
- To see the driver "loaded " printk() message use ..
- #dmesg | tail -5 (display message from /var/log/message
- Welcome By NKP: Linux character device is successfully registered .
- To perform lseek(), open (),close(), read(),write()

Mention some kind of c code to check the above defined function the driver module.

**Steps to remove a module:**

- To remove /unload "chardev.ko
- #rmmodchardev.ko
- Check whether our module is removed /unloaded from kernel by …
- # lsmod | head-5
- Module name should be disappeared.
- To see the driver "unloaded " printk() message use ..
- #dmesg | tail -5 (display message from /var/log/message )
- GoodBYE By NKP: Linux character device is successfully unregistered .
- To go back to user login type ..
- #exit
- To remove all the files created by "makes's clean" type

# make –C /lib/modules/ uname –r / build M='pwd' clean

## References


[1]. A. Rubini, Linux Device Drivers, O'Reilly &Associates,Sebastopol, Calif., 1998
[2]. T. Burke, M.A. Parenti, A. Wojtas. Writing Device Drivers: Tutorial and Reference, Digital Press, Boston,1995.
[3]. Linux Operating System Documentation,http://www.sunsite.unc.edu/pub/Linux
[4]. Robert Love, Linux Kernel Development, Second Edition, 2005
[5]. A. Rubini, "Dynamic Kernels: Modularize Device Drivers," Linux J.,Issue 23, Mar. 1996,
[6]. Y. Zhou, M.S. Li, "Research and implementing of real-time support of Linux kernel", Journal of computer research and development,Vo1.39, No. 4, April 2002.
[7]. P. Mantegazza, E. Bianchi, L. Dozio, S. Papachar-alambous, RTAI: Real Time Application Interface,Linux Journal, April 2000.
[8]. Chen lijun, "Understanding Linux kernel source code deeply"[M],Beijing: Posts & Telecom Press. 2002.
[9]. Linux Kernel http://www.kernel.org.
[10]. ELF specifications be downloaded from ftp://sunsite.unc.edu/pub/Linux/GCC/ELF.doc.tar.gz.
[11]. Murli. B. A, "Linux Device Driver coding for Pseudo device" , International Journal of computational Engineering Research , Pg No. 17-29.
[12]. M. Spear et al. Solving the starting problem: Device drivers as selfdescribingartifacts. In *Eurosys*, 2006.
[13]. M. M. Swift, M. Annamalai, B. N. Bershad, and H. M. Levy.Recovering device drivers.In *OSDI*, 2004.
[14]. L. Wittie. Laddie: The language for automated device drivers (ver 1). Technical Report 08-2, Bucknell CS-TR, 2008.
[15]. A. Silberschatz, P. B. Galvin, and G. Gagne.*Operating System Concepts*.John Wiley and Sons, eighth edition, 2009.
[16]. M. M. Swift, M. Annamalai, B. N. Bershad, and H. M. Levy.Recovering device drivers.In *OSDI*, 2004.
[17]. A. Kadav and M. Swift.Live migration of direct-access devices.*Operating Systems Review*, 43(3):95–104, 2009.
*[18]*. B. Leslie et al. User-level device drivers: Achieved performance. *Jour. Comp. Sci. and Tech.*, 2005.
[19]. Z. Li, S. Lu, S. Myagmar, and Y. Zhou. CP-Miner: A tool for finding copy-paste and related bugs in operating system code. In *OSDI*, 2004.
[20]. F. M. David et al. CuriOS: Improving reliability through operating system structure. In *OSDI*, 2008.
[21]. D. Williams, P. Reynolds, K. Walsh, E. G. Sirer, and F. B. Schneider. Device driver safety through a reference validation mechanism. In *OSDI*, 2008.
[22]. http://opensourceforu.com/2012/01/device-drivers-partitions-hard-disk/