Dr.K.Ramesh NIT Warangal

Linux Device Drivers

Disclaimer: Kernel Modules are dangerous and can render a system useless by corrupting hardware as well as software, not responsible for any damage.

What is Linux?

- Linux is a very widely used open source operating system.
- Why Linux? Why not windows or Mac Os?
- What is the difference between Linux and Unix?
- What Linux is good “Ubuntu”,“Fedora”,…?

Linux File System

- Almost everything in Linux is a file, some that can be read directly with programs like vi, gedit… and others require special programs which parse the data.
- Some directories
  - /usr stores application programs
  - /var stores logfiles, mails…
  - /tmp stores temporary stuff
  - /dev stores all the device drivers
  - /proc stores all running process and their information
  - /mnt contains all the file systems that are mounted(C drive cdrom drive…)
  - /etc contains all the configuration files and so on.

Types of Devices

- In Linux drivers written outside of the kernel are referred to as modules.
- Virtual
  - The individual Hardware simply does not exist. ex. Loop devices.
- Physical
  - They have a hardware component.
- Bus-Based
  - Drives don’t have direct communication with the hardware but only through a Bus-Driver.

Types of Device Drivers

- There are simply two types of device drivers
  - Character based device drivers
    - They can be read just like files with open, read, write commands.
  - Block based device drivers
    - They can give block of data, which makes them ideal for file system drivers.
- A tape driver is a character based device while a disk is a block based device.

Our First Device Driver

- Lets go against convention and skip the “Hello World” device driver and get down to the real deal.
- Our first device will be a device which will display and set the clock.
- A little background on clocks.
- There is a hardware clock and a software clock
- Software clock does not work when the computer is off.
- But when the computer starts the hardware clock is used to set the software clock.
- Once the computer starts the software clock is dominant and sets the hardware clock every 11 minutes or so. Why 11? Someones luck number

Project Goals

- Outline of what we have to do:
- Name our device, lets name it TickTock
- Read the software time
- Write the software time
- That's all!!

The Code (Header)

```
#include <linux/module.h>
#include <linux/string.h>
#include <linux/kernel.h>
#include <linux/time.h>
#include <linux/fs.h>
#include <asm/uaccess.h>
static int Tick_init(void);
static void Tock_exit(void);
static int opendev(struct inode *, struct file *);
static int closedev(struct inode *, struct file *);
static ssize_t readme(struct file *, char *, size_t, loff_t *);
static ssize_t writeme(struct file *, const char *, size_t, loff_t *);
static int Major;
#define SUCCESS 0
#define DEVICE_NAME "TickTock"
static struct file_operations fops = {
  .read = readme,
  .write = writeme,
  .open = opendev,
  .release = closedev
};
```

Init

```
static int Tick_init(void){
    Major = register_chrdev(0, DEVICE_NAME, &fops);
        if(Major>0){
            printk("TickTock is Ready For Requests!:       %d\n",Major);
            return 0;
        }else{
            printk("Something Bad happened : %d",Major);
        }
}
module_init(Tick_init);
```

Deinit

```
static void Tock_exit(void)
{
    unregister_chrdev(Major, DEVICE_NAME);
    printk("Goodbye Crewl World!\n");
}
module_exit(Tock_exit);
```

License?

```
MODULE_LICENSE("GPL");
```

Open and Close

```
static int opendev(struct inode * a, struct file * b){
printk("Someone opened me!\n");
//nothing to do!!!
return SUCCESS;
}
static int closedev(struct inode * a, struct file * b){
//nothing to do!!
printk("Someone Closed me!\n");
return SUCCESS;
}
```

Random Utility Functions

```
int pow(int a, int b);
static int itoa(int val,char** ca);
int atoi(char * buff);
```

Read

```
static ssize_t readme(struct file *filp,char *buffer,size_t length,loff_t *offset)  {
int len;
   struct timeval time;
   do_gettimeofday(&time);
   char *b;
   b=kmalloc(sizeof(char)*40,GFP_KERNEL);
   len=itoa(time.tv_sec,&b);
   copy_to_user(buffer,b,len);
   printk("Return val:%s\n",b);
   return length;
}
```

Write

```
static ssize_t writeme(struct file *filp,const char *buff,size_t len,
 loff_t *off)
{
  int i=atoi(buff);
  struct timespec ab;
  ab.tv_sec=i;
  ab.tv_nsec=0;
  do_settimeofday(&ab);
  return len;
}
```

Finished?

- And the coding is complete!
- There is really no logic involved, just coding.
  - So whats next?

The MakeFile

- obj-m := TickTock.o
- KDIR := /lib/modules/$(shell uname -r)/build
- PWD   := $(shell pwd)
- default:
- $(MAKE) -C $(KDIR) M=$(PWD) modules

How many more Steps?!?!

- In shell type: make
- It will generate a bunch of files
- The one that interests us is the file ending in *.ko
- Go into super user su
- Insmod ./TickTock.ko
- Dmesg | tail
- Mknod /dev/TickTock c Major 0

Done!!

- That's all its finished!
- Just open the /dev/TickTock as a normal file with read /write / or both
- Read will give a string representing the number of seconds since a certain day usually Jan 1, 1970 0:00
- Write will accept a string of a number representing the number of seconds since a certain day usually Jan 1, 1970 0:00 and set the clock accordingly
- Now block device drivers are the same as character drivers with a few changes that allows them to transfer blocks of data instead of character by character