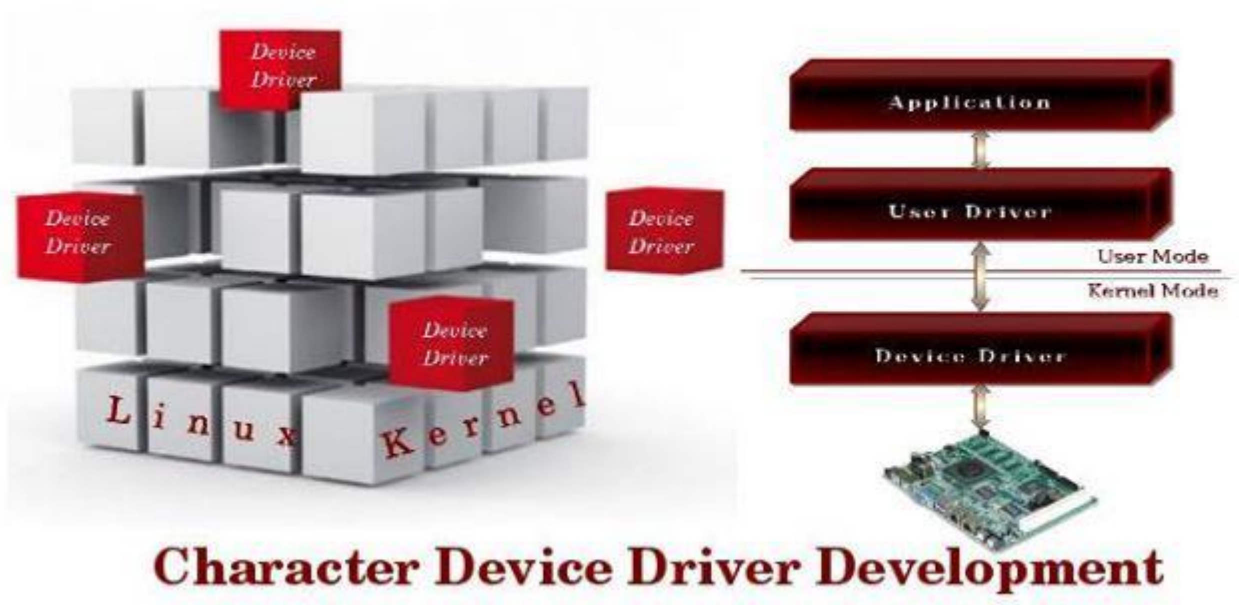
EMBEDDED SYSTEMS TRAINING

Khóa học : "Basic Embedded Linux"

❖ **Email** : *training.laptrinhnhung@gmail.com*

❖ **Website** : *http://hethongnhung.com* – *http://laptrinhnhung.com*


Device Driver
Device Driver
Device Driver
Device Driver
Linux Kernel
Application
User Driver
User Mode
Kernel Mode
Device Driver
Character Device Driver Development

Types of Device Drivers

- **Char device drivers**
  - ✓ A character (char) device is one that can be accessed as a stream of bytes (like a file); a char driver is in charge of implementing this behavior. Such a driver usually implements at least the *open, close, read,* and *write* system calls.
  - ✓ Char devices are accessed by means of *filesystem nodes*, such as */dev/tty1* and */dev/lp0*.
  - ✓ The only difference between a char device and a regular file is that you can always move back and forth in the regular file, whereas most *char devices are just data channels*, which you can only access sequentially.

Types of Device Drivers

- **Types of Device Drivers**
  - ***Block device drivers***
    - ✓ Like char devices, *Block devices* are accessed by filesystem nodes in the */dev* directory.
    - ✓ A block device is something that can host a filesystem, such as a disk.
    - ✓ Linux allows the application to read and write a block device like a char device—it permits the *transfer of any number of bytes at a time*. As a result, block and char devices differ only in the way data is managed internally by the kernel, and thus in the kernel/driver software interface.
    - ✓ A block driver offers the kernel the same interface as a char driver, as well as an additional *block -oriented interface* that is invisible to the user or applications opening the */dev* entry points. That block interface, though, is essential to be able to *mount* a filesystem.

Types of Device Drivers

- **Types of Device Drivers**
  - ***Network device drivers***
    - ✓ A *Network device* is in charge of sending and receiving data packets, driven by the network subsystem of the kernel, without knowing how individual transactions map to the actual packets being transmitted.
    - ✓ Not being a stream-oriented device, a network interface isn’t easily mapped to a node in the filesystem, as */dev/tty1* is. The Linux way to provide access to interfaces is still by assigning a unique name to them (*such as eth0*), but that name doesn’t have a corresponding entry in the filesystem.
    - ✓ Communication between the kernel and a network device driver is completely different from that used with char and block drivers. *Instead of read and write*, the kernel calls functions related to packet transmission.

Char Device Drivers

Overview

- Char devices
- Definition of MAJOR and MINOR numbers
- Implementing file operations
- Installing and running the Char modules
- Writing a simple Char device driver

Char devices

- **Basics of Char devices**
  - ✓ A character (char) device is one that can be accessed as a stream of bytes (like a file); a char driver is in charge of implementing this behavior. Such a driver usually implements at least the *open, close, read,* and write system calls.
  - ✓ The Char device drivers will be "seen" to the users as some "special files". The whole operation on these devices are similar to file operations. ie, User can Open,Read,Write and Close these devices as if it is a file.

Char devices

- **Major number & Minor number**
  - ✓ The *Major* number identifies the driver associated with the device. The kernel uses the major number at open time to dispatch execution to the appropriate driver.
  - ✓ The *Minor* number is used only by the driver specified by the Major number. It is common for a driver to control several devices .The minor number provides a way for the driver to differentiate among them. Check the Eg below
  - ✓ There are 2 hard-disks in a PC with names */dev/hdc* and */dev/hdb*. The Major number distinguishes which physical disk to use (ie *hdc* or *hdb*).
  - ✓ If these disks have the partitions namely *dev/hdc0*, */dev/hdc1* and */dev/hdb0*, */dev/hdb1*. The Minor numbers chooses which partition to choose from a specific disk

Char devices

- **Creating a Char device module**
  - As mentioned before, the char devices will act like files. So in order to use them, we have to create a module that will have the following functions
    - ✓ `init_module()`
    - ✓ `Device_open()`
    - ✓ `device_read()`
    - ✓ `device_write()`
    - ✓ `device_close()`
    - ✓ `cleanup_module()`
    - ✓ `etc`

Char devices

- **Registering a Module**
  - ✓ Registering a module virtually “tells” all other processes that a Module with “module_name” exists and is ready for use.
  - ✓ This is done by using the System call
  - ✓ `int register_chrdev(unsigned int major, const char *name,struct file_operations *fops);`
  - ✓ The *major* argument is the major number being requested.
  - ✓ *name* is the name of your device, which will appear in */proc/devices*.
  - ✓ *fops* is the pointer to an array of function pointers, used to invoke your driver’s entry points.

  Contd…

Char devices

- **The Structure file_operations**

```
struct file_operations
{
    int (*lseek)(struct inode *,struct file *,off_t,int);
    int (*read)(struct inode *,struct file *,char *,int);
    int (*write)(struct inode *,struct file *,char *,int);
    int (*readdir)(struct inode *,struct file *,struct dirent *,int);
    int (*select)(struct inode *,struct file *,int,select_table *);
    int (*ioctl)(struct inode *,struct file *,unsigned int,unsigned long);
    int (*mmap)(struct inode *,struct file *,struct vm_area_struct *);
    int (*open)(struct inode *,struct file *);
    void (*release)(struct inode *, struct file *);
    int (*fsync)(struct inode *,struct file *);
    int (*fasync)(struct inode *,struct file *,int);
    int (*check_media_change)(dev_t dev);
    int (*revalidate)(dev_t dev);
};
```

Char devices

- **Registering a Char device Contd…**
  - ✓ The function `register_chrdev()` will return 0 on success or any other value on error.
  - ✓ Registering a module must be always included in the ***`init_module()`*** function so that the module is installed whenever we run
    - *`$:>insmod module_name.o`*
  - ✓ If the module is installed successfully, it will not display any ‘error’ messages.
  - ✓ To check the module has installed properly, type
    - *`$:>cat /proc/modules`*
    - *`Or $:> cat /proc/devices`*

Char devices

- **Unregistering a Char device**
  - When a Module is unloaded from the system, the Major number must be released. This is accomplished by the following function in the ***`cleanup_module()`***

    ✓ ***`int unregister_chrdev(unsigned int major, const char *name)`***

    ✓ *major* -> is the MAJOR number associated with the device

    ✓ *name* -> is the name associated with the device

A Simple Char module

Eg:

```
--------Your Header declarations--------
struct file_operations parFops1=
{
   //No file operations are implemented
};
int init_module(void)
{
   int major;
   printk("In the INIT MODULE....\n");
   major=register_chrdev(MAJ_NO,“my_device",&parFops1);
   if(major){
      printk("Cannot register...\n");
      return -1;
   }
   return 0;
}
```

A Simple Char module

➢ **Contd…**

```
int cleanup_module(void)
{
    unregister_chrdev(MAJ_NO,“my_device");
    printk("Bye....\n");
    return 0;
}
```

After creating the source file (*driver.c*), Compile it using the option

```
$:> gcc -D__KERNEL__ -DMODULE -I/usr/src/linux-2.4.20-8/include/ -Wall -c driver.c -o driver.o
```

Testing your Module

- **Steps for testing your Module**
  - If the files are compiled successfully, we will get an object file (*.o) at the current directory
  - For installing the module, type the command
    - *`$:>/sbin/insmod driver.o`* OR
    - *`$:> insmod driver.o`*
  - Now check the device module using the following commands
    - *`$:> cat /proc/modules`* and
    - *`$:> cat /proc/devices`*
    - The first command will give the module listing and the second command will give the **CHAR** and **BLOCK** device listings

Char devices

- **Creating the Entrypoint**
  - After the module installation is successful, the next step is to create the device entry at the ***/dev*** directory.
  - This is done by creating a special file using the ***mknod*** command.
  - ***`$:>mknod /dev/new_device c 254 1`***
    - ***`new_device`*** : The device name; same that is used in ***`register_chrdev()`***
    - ***`c`*** : Indicates that a character device is created
    - ***254*** : The Major number
    - ***1*** : The Minor number

❖ Major numbers in the ranges 60 to 63, 120 to 127, and 240 to 254 are reserved for local and experimental use: no real device will be assigned such major numbers. Minor numbers should be in the range 0 to 255 because, they are stored in a single byte.

Char devices

- **Performing the file operations**

  - As described before, the char devices support different file operations as described in the ***`file_operations`*** structure

  - The most common among them is as follows
    - ***`int (*open)(struct inode *,struct file *)`***
      - Though this is always the first operation performed on the device file, the driver is not required to declare a corresponding method. If this entry is ***NULL***, opening the device always succeeds, but your driver isn't notified.

Char devices

- **Performing file operations Contd…**
  - ***ssize_t (*write)(struct file *,const char *, size_t,loff_t *)***
    - Sends data to the device. If missing, ***-EINVAL*** is returned to the program calling the ***write*** system call. The return value, if non-negative, represents the number of bytes successfully written.
  - ***ssize_t (*read)(struct file *,char *, size_t,loff_t *)***
    - Used to retrieve data from the device. A null pointer in this position causes the ***read*** system call to fail with ***–EINVAL*** ("Invalid argument"). A non-negative return value represents the number of bytes successfully read.

Char devices

- **Performing file operations**
  - ***void (*release)(struct inode *,struct file *)***
    - The *release* function is used to release the control over the specific device driver similar to a file.
    - The *release* method could not fail, and thus returned void.
  - ***int (*ioctl)(struct inode *inode,struct file *filp,unsigned int cmd,unsigned long arg)***
    - The *inode* and *filp* pointers are the values corresponding to the file descriptor *fd* passed on by the application and are the same parameters passed to the *open* method.
    - The *cmd* argument is passed from the user unchanged, and the optional *arg* argument is passed in the form of an unsigned long, regardless of whether it was given by the user as an integer or a pointer.

Creating a Simple driver

- **Steps for creating a simple Char device driver**
  - Create the source file (*driver.c*)with the following functions
    - `init_module()` With device name "*my_device*"
    - `my_open()`
    - `my_release()`
    - `my_write()`
    - `cleanup_module()`
  - Compile using the following option
    - `gcc -D__KERNEL__ -DMODULE -I/usr/src/linux-2.4.20-8/include/ -Wall -c driver.c -o driver.o`
  - Install the Module using the command
    - `$:>insmod driver.o`
  - Create the Entrypoint using the command
    - `$:>mknod /dev/my_device c 254 1`

Creating a Simple driver

- **Writing the Source**
  - Always include the following header files

    ```
    <linux/module.h>
    <linux/fs.h>
    <linux/mm.h>
    <linux/segment.h>
    <linux/io.h>
    <linux/uaccess.h>
    ```
  - We now write a Char device driver for the Parallel port of the PC. Usually, the Port address of a parallel port is 0x378
  - The actual source codes is as follows

Creating a Simple driver

- **Writing the Source**

```
unsigned int MAJ_NO=254;
static int my_open(struct inode *inode,struct file *filep){
    printk("Device opened\n");
    return 0;
}
static int my_release(struct inode *inode,struct file *filep){
    printk("Device Closed\n");
    return 0;
}
static ssize_t my_write(struct file *file1,const char *ch,size_t count1,loff_t
*offset1){
    printk("Inside the write module....\n");
    get_user(x1,ch); //Explained later in this section
    printk("The passed data is %x \n",x1);
    return 0;
}
```

Creating a Simple char driver

- **Writing the Source**

```
struct file_operations parFops1=
{
    open:my_open, //The file functions implemented
    release:my_release,
    write:my_write,
};
int init_module(void)
{
    int major;
    printk("In the INIT MODULE....\n");
    major=register_chrdev(MAJ_NO,“my_device",&parFops1);
    if(major){
        printk("Cannot register...\n");
        return 0;
    }
    return 0;
}
```

Creating a Simple char driver

- **Writing the Source**

```
int cleanup_module(void)
{
    unregister_chrdev(MAJ_NO,“my_device");
    printk("Bye....\n");
    return 0;
}
```

  - After creating the source file (driver.c), Compile it using the option

```
$:> gcc -D__KERNEL__ -DMODULE -I/usr/src/linux-2.4.20-8/include/ -Wall -c
driver.c -o driver.o
```

Installing the Driver

- **Install the char device driver**
  - To install the module that is created, execute the command

    ```
    $:>insmod driver.o
    ```

  - If the installation is successful, the codes inside the init_module() will execute

```
 -------------//some warning messages------
-----------------------------------------------------
In the INIT MODULE Your message in init_module()
-----------------------------------------------------
```

Testing the Driver

- **Testing the char device driver**
  - To test the char device driver, we create another application which uses our device driver.

```
#include<fcntl.h>
main()
{
    int fp;
    char *mydata="A"; //Single character
    printf("My application…\n");
    fp=open("/dev/my_driver",O_RDWR|0666);
    if(fp<0){
        printf("Cannot open the device\n");
        exit(0);
    }
    write(fp,mydata,1);
    close(fp);
}
```

Testing the Driver

- **Test application Contd…**

  If the test application is run, the following codes will execute

  - ***`my_open()`***

    Device opened…

  - ***`my_write()`***

    Inside the write module....

    The passed data is A

  - ***`my_release()`***

    Device closed…

Questions?