

BAN CƠ YẾU CHÍNH PHỦ
HỌC VIỆN KỸ THUẬT MẬT MÃ



PHÁT TRIỂN TRÌNH ĐIỀU KHIỂN CHO THIẾT BỊ TRUYỀN DỮ LIỆU DẠNG KÝ TỰ

Ngành: Công nghệ thông tin

Chuyên ngành: Kỹ thuật phần
mềm nhúng và di động

Mã số: 52.48.02.01

Nội dung

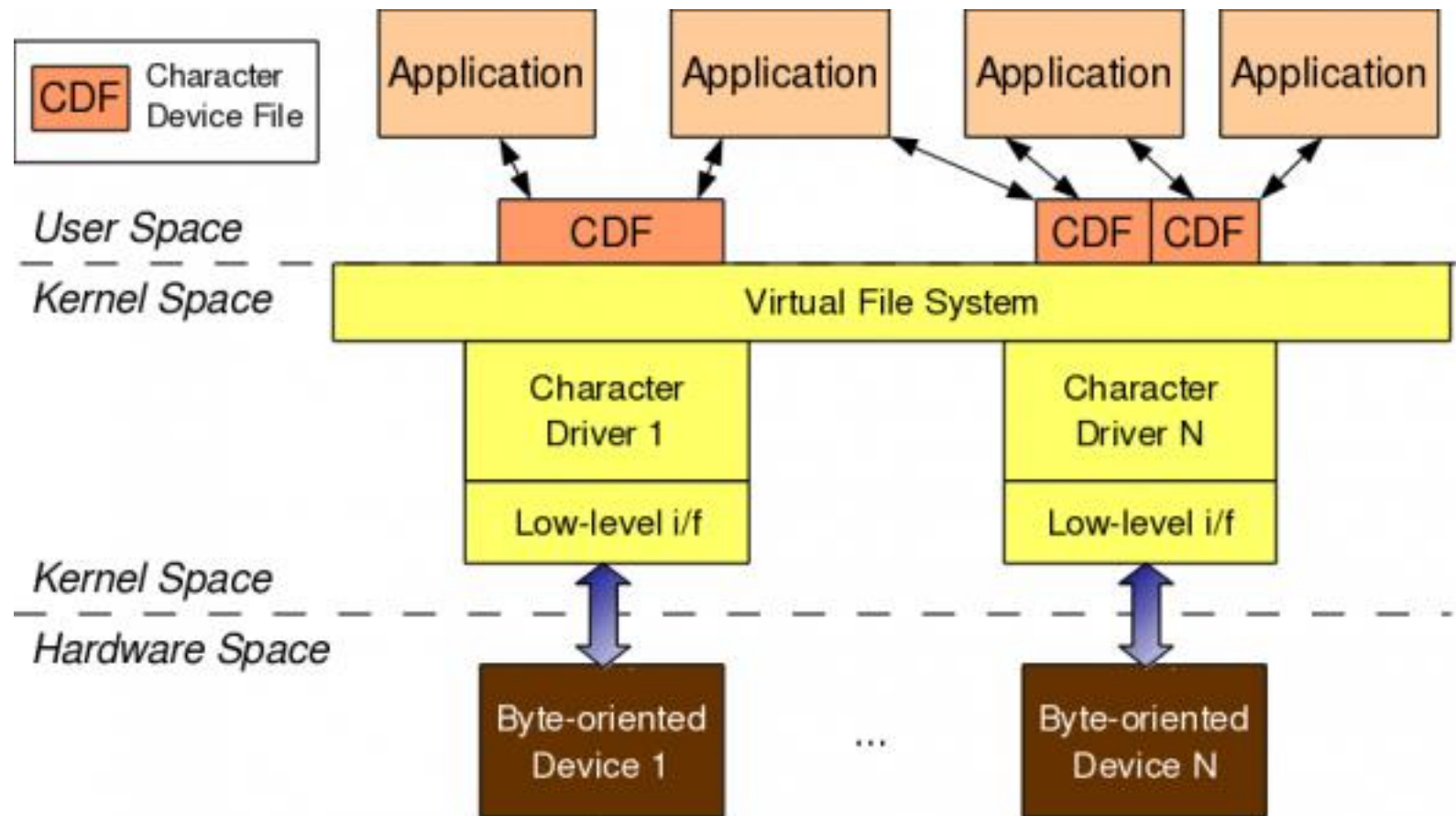
- Khái niệm và mô hình hoạt động
- Device file (device node)
- Số hiệu file thiết bị (Major và minor number)
- Cấu trúc của character driver
- Cấu trúc dữ liệu quan trọng
- Đăng ký thiết bị
- Cài đặt các thao tác cơ bản
- Ví dụ minh họa

Khái niệm và mô hình hoạt động

- Khái niệm
 - Thiết bị truyền dữ liệu dạng ký tự: là thiết hướng byte, sử dụng đơn vị truyền nhỏ nhất là byte
 - Phần lớn driver thiết bị là driver hướng byte
 - Ví dụ: serial drivers, audio drivers, video drivers, camera drivers
- Hoạt động: qua 4 bước
 - 1. Application (ứng dụng)
 - 2. Character device file (CDF - File thiết bị)
 - 3. Character device driver (Driver thiết bị)
 - 4. Character device (Thiết bị)

Khái niệm và mô hình hoạt động

- Mô hình hoạt động

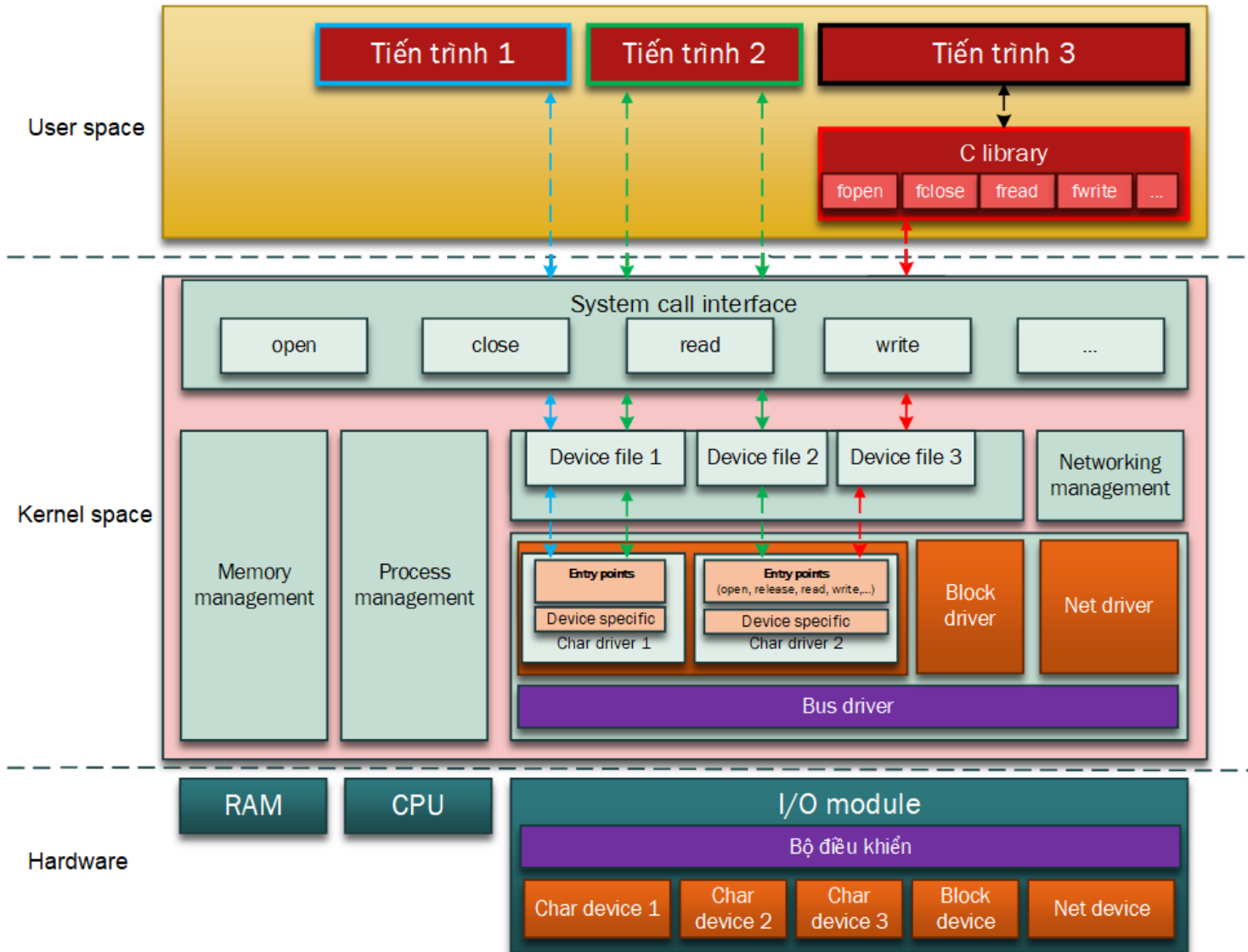


Khái niệm và mô hình hoạt động

- Chú ý:

- Các thực thể trong mô hình có thể tồn tại một cách độc lập trên 1 hệ thống mà không cần sự tham gia của các thực thể khác
- Các kết nối giữa chúng cần được thực hiện một cách tường minh
- Một ứng dụng kết nối đến file thiết bị bằng cách **gọi một hàm mở file thiết bị** đó
- Các **file thiết bị liên kết đến driver của nó bằng thao tác đăng ký được thực hiện trong driver**
- Một driver được kết nối đến một thiết bị thông qua các thao tác mức thấp với thiết bị phần cứng đặc trưng
- File thiết bị không phải là thiết bị thực sự, nó chỉ là một thực thể nắm giữ thiết bị thực sự

Khái niệm và mô hình hoạt động



Số hiệu file thiết bị

- Việc kết nối giữa ứng dụng và file thiết bị được thực hiện thông qua **tên file thiết bị**
- Tuy nhiên, kết nối giữa **file thiết bị và device driver** được thực hiện dựa trên **số hiệu của file thiết bị** chứ không thông qua tên file
- Điều này cho phép
 - các ứng dụng ở không gian người dùng có thể sử dụng bất kỳ tên file thiết bị nào
 - không gian nhân có thể sử dụng các kết nối thông qua số hiệu thông thường giữa file thiết bị và driver
- Các số hiệu file thiết bị này thường được biết đến như là một cặp số <major, minor>

Số hiệu file thiết bị

- Từ phiên bản 2.6, nhiều driver có thể sử dụng cùng số hiệu major nhưng số hiệu minor khác nhau.
- Có thể sử dụng câu lệnh như sau để liệt kê các file thiết bị kiểu character trên hệ thống:

```
$ ls -l /dev/ | grep "^c"
```


Cấu trúc của character driver

- **Phần OS specific** gồm các nhóm hàm sau:
 - Hàm khởi tạo. Hàm này chịu trách nhiệm:
 - ✓ Yêu cầu kernel cấp phát device number.
 - ✓ Yêu cầu kernel tạo device file.
 - ✓ Yêu cầu kernel cấp phát bộ nhớ cho các cấu trúc dữ liệu của driver và khởi tạo chúng.
 - ✓ Yêu cầu khởi tạo thiết bị vật lý.
 - ✓ Đăng ký các hàm entry point với kernel.
 - ✓ Đăng ký hàm xử lý ngắt.
 - Hàm kết thúc. Hàm này làm ngược lại những gì hàm khởi tạo đã làm.
 - Các hàm entry point: Ví dụ *open()*, *release()*, *read()*, *write()*, *ioctl()*, *mmap()*...

Cấu trúc của character driver

- **Phần device specific** gồm các nhóm hàm sau:
 - Nhóm các hàm khởi tạo/giải phóng thiết bị.
 - Nhóm các hàm đọc/ghi vào các thanh ghi của thiết bị.
 - ✓ Đọc/ghi các thanh ghi dữ liệu.
 - ✓ Lấy thông tin từ các thanh ghi trạng thái.
 - ✓ Thiết lập lệnh cho các thanh ghi điều khiển.
 - Nhóm các hàm xử lý ngắt.

Cấu trúc dữ liệu quan trọng

- Hầu hết các hoạt động trình điều khiển dạng ký tự cơ bản liên quan đến ba cấu trúc dữ liệu nhân quan trọng là *file_operations*, *file* và *inode*
- **Cấu trúc *file_operations***
 - Là cách trình điều khiển ký tự thiết lập kết nối
 - Được định nghĩa trong `<linux/fs.h>` là một tập hợp các con trỏ hàm
 - Mỗi lần mở, tệp tin được liên kết với tập hợp hàm riêng của nó (bằng cách bao gồm một trường với lời gọi `f_op` mà trỏ tới cấu trúc *file_operations*)
 - Các hoạt động chủ yếu phụ trách thực hiện các lời gọi hệ thống như `read`, `write`, v.v.

Cấu trúc dữ liệu quan trọng

- Minh họa cấu trúc

```
loff_t (*llseek) (struct file *, loff_t, int);
ssize_t (*read) (struct file *, char __user *, size_t,
loff_t *);
ssize_t (*aio_read)(struct kiocb *, char __user *,
size_t, loff_t);
ssize_t (*write) (struct file *, const char __user *,
size_t, loff_t *);
ssize_t (*aio_write)(struct kiocb *, const char __user
*, size_t, loff_t *);
int (*readdir) (struct file *, void *, filldir_t);
unsigned int (*poll) (struct file *, struct
poll_table_struct *);
int (*ioctl) (struct inode *, struct file *, unsigned
int, unsigned long);
int (*mmap) (struct file *, struct vm_area_struct *);
int (*open) (struct inode *, struct file *);
```

Cấu trúc dữ liệu quan trọng

- **Cấu trúc file_operations**

- Cấu trúc này gồm các nguyên mẫu hàm
- Được sử dụng như một khuôn mẫu chung
- Khi xây dựng một driver dạng ký tự, cần khai báo một thể hiện cụ thể của cấu trúc và định nghĩa các hàm trong cấu trúc
- Ví dụ: driver dạng ký tự đơn giản scull

```
struct file_operations tenCauTrucDriverThucTe = {  
    .owner = THIS_MODULE,  
    .llseek = scull_llseek,  
    .read = scull_read,  
    .write = scull_write,  
    .ioctl = scull_ioctl,  
    .open = scull_open,  
    .release = scull_release,  
};
```

Cấu trúc dữ liệu quan trọng

- Cấu trúc file
 - được định nghĩa trong <linux/fs.h>
 - đại diện cho file đang mở (nó không dành riêng cho trình điều khiển thiết bị, mọi tệp tin đang mở trong hệ thống đều có một cấu trúc file trong không gian nhân)
 - Các trường quan trọng nhất của cấu trúc file

```
mode_t f_mode;
loff_t f_pos;
unsigned int f_flags;
struct file_operations *f_op;
void *private_data;
struct dentry *f_dentry;
```

Cấu trúc dữ liệu quan trọng

- Cấu trúc inode

- Cấu trúc inode được dùng bên trong nhân để thể hiện các tệp tin
- Cấu trúc inode chứa nhiều thông tin về tệp tin
- Có thể có nhiều cấu trúc tệp tin đại diện cho nhiều mô tả mở trên một tệp tin, nhưng chúng đều trỏ tới một cấu trúc inode duy nhất
- Hai trường của cấu trúc này được quan tâm để viết mã tình điều khiển

`dev_t i_rdev`: inode đại diện cho tệp tin thiết bị, trường này chứa số hiệu thiết bị thực tế.

`struct cdev *i_cdev`: `struct sdev` là cấu trúc trong của nhân mà đại diện cho thiết bị ký tự

Khung chương trình của driver dạng kt - File *vchar_driver.c*

```
#include <linux/module.h>
#define DRIVER_AUTHOR "ten"
#define DRIVER_DESC "mo tar"
#define DRIVER_VERSION "0.1"
/***** device specific - START *****/
/* ham khoi tao thiet bi */
/* ham giai phong thiet bi */
/* ham doc tu cac thanh ghi du lieu cua thiet bi */
/* ham ghi vao cac thanh ghi du lieu cua thiet bi */
/* ham doc tu cac thanh ghi trang thai cua thiet bi */
/* ham ghi vao cac thanh ghi dieu khien cua thiet bi */
/* ham xu ly tin hieu ngat gui tu thiet bi */
/***** device specific - END *****/
/***** OS specific - START *****/
/* cac ham entry points */
/* ham khoi tao driver */
static int __init vchar_driver_init(void)
{
    /* cap phat device number */
    /* tao device file */
    /* cap phat bo nho cho cac cau truc du lieu cua driver va khoi
    tao */
    /* khoi tao thiet bi vat ly */
    /* dang ky cac entry point voi kernel */
    /* dang ky ham xu ly ngat */
    printk("Initialize vchar driver successfully\n");
    return 0;
}
```

```
/* ham ket thuc driver */
static void __exit vchar_driver_exit(void)
{
    /* huy dang ky xu ly ngat */
    /* huy dang ky entry point voi kernel */
    /* giai phong thiet bi vat ly */
    /* giai phong bo nho da cap phat cau truc du lieu cua driver */
    /* xoa bo device file */
    /* giai phong device number */
    printk("Exit vchar driver\n");
}
/***** OS specific - END *****/
module_init(vchar_driver_init);
module_exit(vchar_driver_exit);
MODULE_LICENSE("GPL"); /* giay phep su dung cua module */
MODULE_AUTHOR(DRIVER_AUTHOR); /* tac gia cua module */
MODULE_DESCRIPTION(DRIVER_DESC); /* mo ta chuc nang cua
module */
MODULE_VERSION(DRIVER_VERSION); /* mo ta phien ban cuar
module */
MODULE_SUPPORTED_DEVICE("testdevice"); /* kieu device ma
module ho tro */
```


Các thao tác với file thiết bị

- Tất cả những lời gọi hệ thống (hay thao tác file) đối với một file thông thường đều có thể áp dụng cho file thiết bị
- Trong user space, hầu hết mọi thứ đều là một file
 - Điều khác biệt nằm ở tầng nhân hệ điều hành, trong đó hệ thống file ảo (VFS) sẽ giải mã các loại file và diễn giải các thao tác file truyền đến driver thiết bị tương ứng đối với file thiết bị
- Để VFS truyền các thao tác trên file thiết bị vào driver, nó phải được thông báo về các thao tác đó
 - Việc làm này chính là **đăng ký các thao tác file với VFS được thực hiện trong mã nguồn driver**

Đăng ký thiết bị

- **Bước 1:**

- Tạo ra biến cấu trúc `struct file_operations pugs_fops` và điền vào cấu trúc này các thao tác xử lý muốn dùng đối với file thiết bị đang viết driver, các thao tác thông thường như `my_open`, `my_close`, `my_read`, `my_write`, ...
- Khởi tạo một cấu trúc thiết bị kiểu `character` bằng cách **khai báo biến cấu trúc `struct cdev c_dev` và gọi hàm `cdev_init()`**.

- **Bước 2:**

- Điều khiển cấu trúc này đến hệ thống file ảo VFS bằng cách **gọi hàm `cdev_add()`**
- Các hàm `cdev_init()` và `cdev_add()` được khai báo trong `<linux/cdev.h>`

Ví dụ minh họa

- **Phân tích code trong file**

CodeMinhHoa_CharDriver_PhanTich_Chapter3.docx

- **Biên dịch driver trên và nạp vào nhân theo các bước:**

- 1. Viết Makefile và chạy make để biên dịch driver tạo ra file .ko.
- 2. Nạp driver vào nhân hệ thống sử dụng lệnh insmod.
- 3. Liệt kê danh sách các module đã nạp sử dụng lệnh lsmod.
- 4. Xem thông tin số hiệu major và module đã nạp sử dụng lệnh cat /proc/devices.
- 5. Xem thông tin về driver đã nạp bằng cách dùng lệnh dmesg
- 6. Gỡ bỏ driver sử dụng lệnh rmmod

Ví dụ minh họa

- Tạo makefile tương ứng

```
#Makefile
KDIR = /lib/modules/`uname-r`/build

all :
    make - C $(KDIR) M = `pwd`

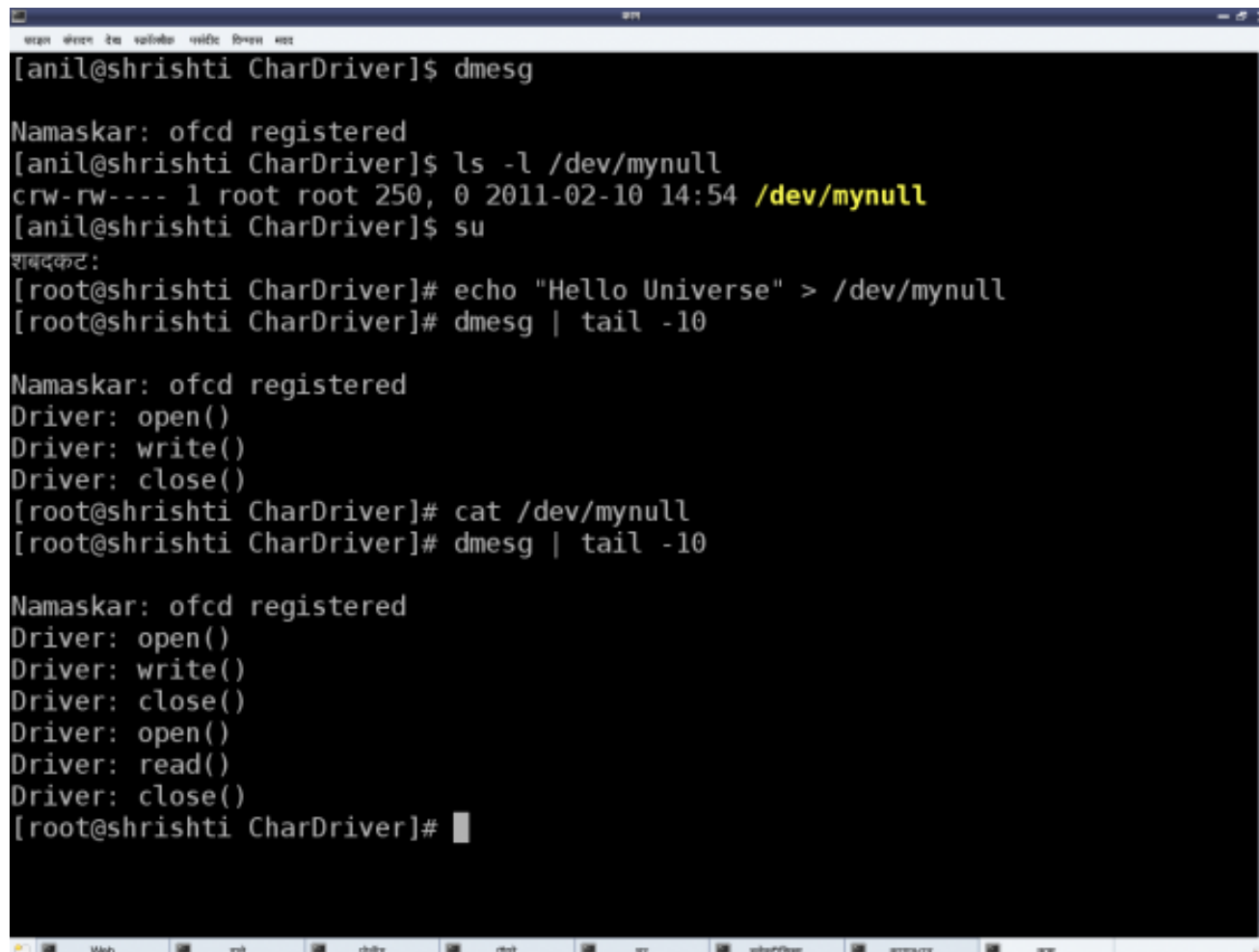
clean :
    make - C $(KDIR) M = `pwd` clean
```

```
#Kbuild
EXTRA_CFLAGS = -Wall

obj - m = vchar_driver.o
```

Ví dụ minh họa

- Kết quả



```
[anil@shrishti CharDriver]$ dmesg
Namaskar: ofcd registered
[anil@shrishti CharDriver]$ ls -l /dev/mynull
crw-rw---- 1 root root 250, 0 2011-02-10 14:54 /dev/mynull
[anil@shrishti CharDriver]$ su
शब्दकट:
[root@shrishti CharDriver]# echo "Hello Universe" > /dev/mynull
[root@shrishti CharDriver]# dmesg | tail -10

Namaskar: ofcd registered
Driver: open()
Driver: write()
Driver: close()
[root@shrishti CharDriver]# cat /dev/mynull
Hello Universe
[root@shrishti CharDriver]# dmesg | tail -10

Namaskar: ofcd registered
Driver: open()
Driver: write()
Driver: close()
Driver: open()
Driver: read()
Driver: close()
[root@shrishti CharDriver]#
```

HỎI - ĐÁP
