

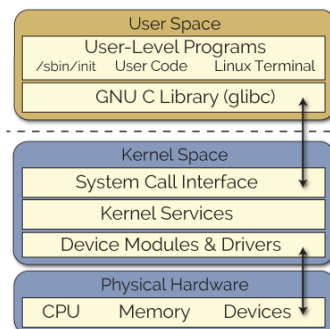
# BÁO CÁO ĐỒ ÁN LINUX KERNEL

*Hoàng Thiên Nữ - 1612880*

## 1. Linux kernel module là gì?

Kernel module là một cơ chế giúp thêm hoặc loại bỏ code ra khỏi Linux kernel ở thời gian thực. Cơ chế này thích hợp cho những driver các thiết bị, kích hoạt kernel giao tiếp với phần cứng mà không cần phải biết rõ nguyên lý hoạt động của phần cứng.

Kernel module chạy ở Kernel Space và các ứng dụng chạy ở User Space (hình 1). Cả kernel space và user space đều có những vùng địa chỉ nhớ riêng biệt mà không bị trùng lặp nhau. Phương pháp này đảm bảo các ứng dụng chạy ở user space có một cái nhìn nhất quán đối với phần cứng, bất kể nền tảng nào. Các dịch vụ ở kernel được cung cấp cho user space thông qua system calls. Kernel ngăn các ứng dụng ở user space có xung đột với nhau hoặc truy cập vào tài nguyên không cho phép thông qua các mức bảo vệ.



**Hình 1** Giao tiếp giữa các thành phần máy tính

## 2. Xây dựng một kernel module

Một phương pháp để xây dựng một kernel module là sử dụng kernel code. Khác với việc viết một chương trình ứng dụng, một kernel module không có hàm main() và có một số điểm lưu ý như sau:

- Không được thực thi một cách tuần tự: một kernel module sẽ tự đăng kí để xử lý những yêu cầu sử dụng các hàm khởi tạo của chính nó. Các hàm đó sẽ chạy và dừng sau đó. Các loại yêu cầu mà nó xử lý có thể được định nghĩa bên trong module code.
- Không tự động dọn dẹp: bất cứ tài nguyên nào được cấp phát ở module code đều phải được giải phóng một cách thủ công khi mà module dừng. Nếu không nó sẽ vẫn chiếm một vùng tài nguyên cho đến khi hệ thống khởi động lại.
- Không có printf(): kernel code không thể truy cập những thư viện code được viết ở tầng user space. Kernel module chỉ chạy ở kernel space và có một vùng nhớ riêng. Có

một sự giao tiếp giữa kernel space và user space được định nghĩa. Do vậy, ta sẽ dùng `printk()` để xuất thông tin từ user space.

- Có thể bị gián đoạn.
- Có đặc quyền thực thi ở cấp độ cao hơn.
- Không hỗ trợ chấm động: kernel code sử dụng traps để chuyển đổi từ số nguyên sang số chấm động cho các ứng dụng ở user space.

Cần phải có Makefile để xây dựng một kernel module. Cấu trúc một Makefile như sau:

```
1 obj-m+= modulename.o
2
3 all:
4     make -C /lib/modules/$(shell uname -r)/build/ M=$(PWD) modules
5 clean:
6     make -C /lib/modules/$(shell uname -r)/build/ M=$(PWD) clean
```

Dòng đầu tiên là goal definition. Nó sẽ khai báo tên module để xây dựng. Sử dụng `obj-m` để khai báo một module. `$(shell uname -r)` là một lời gọi để trả về phiên bản kernel hiện tại. Quá trình cài đặt module được gọi qua **make**.

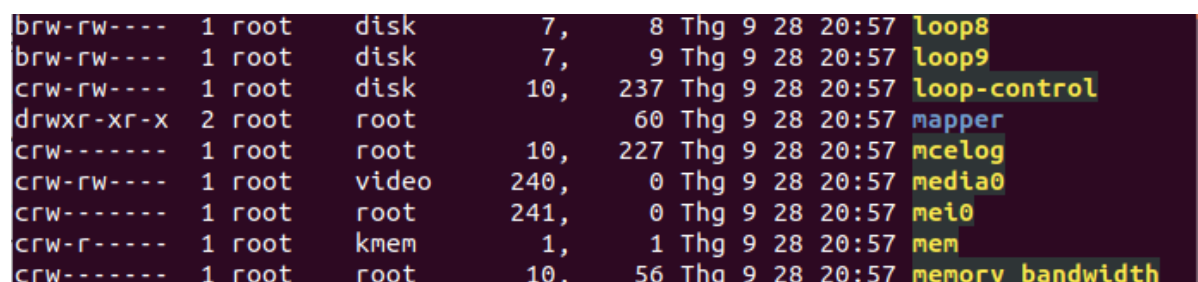
### 3. Hệ thống quản lý file và device trong Linux

#### a. Character Device

Một character device sẽ giúp chuyển dữ liệu qua lại giữa kernel và ứng dụng ở user space. Chúng giống như một cổng để kết nối để đọc hoặc ghi các byte dữ liệu theo một luồng từng kí tự. Chúng cung cấp một nền tảng cho nhiều driver, ví dụ như video hay âm thanh. Một sự thay thế cho character device là block device. Block device cho phép một dãy buffer của cache điều khiển những thao tác như đọc, viết và tìm kiếm. Tất cả các thiết bị đều có thể được truy cập thông qua hệ thống file được đính kèm vào cây file hệ thống. Ví dụ, với đồ án này, toàn bộ code của chương trình khi chạy sẽ trở thành một device `/dev/generDev`

#### b. Major and minor number

Major number được dùng trong kernel để xác định đúng driver của thiết bị khi thiết bị được truy cập. Mặt khác, minor number phụ thuộc vào thiết bị, và chỉ sử dụng bên trong driver. Ta có thể thấy được các cặp major/minor của mỗi device trong thư mục `/dev`.



brw-rw----	1	root	disk	7,	8	Thg 9 28 20:57	loop8
brw-rw----	1	root	disk	7,	9	Thg 9 28 20:57	loop9
crw-rw----	1	root	disk	10,	237	Thg 9 28 20:57	loop-control
drwxr-xr-x	2	root	root		60	Thg 9 28 20:57	mapper
crw-----	1	root	root	10,	227	Thg 9 28 20:57	mcelog
crw-rw----	1	root	video	240,	0	Thg 9 28 20:57	media0
crw-----	1	root	root	241,	0	Thg 9 28 20:57	mei0
crw-r-----	1	root	kmem	1,	1	Thg 9 28 20:57	mem
crw-----	1	root	root	10,	56	Thg 9 28 20:57	memory_bandwidth

Hình 2 Major và minor number của các thiết bị trong Linux

Character device sẽ được xác định bởi chữ 'c' ở cột đầu tiên trong danh sách, và block device là chữ 'b'. Quyền truy cập, chủ sở hữu và nhóm các thiết bị đều được cung cấp với mỗi thiết bị.

```
thiennu@thiennu-UX305UAB:/dev$ groups
thiennu adm cdrom sudo dip plugdev lpadmin sambashare
thiennu@thiennu-UX305UAB:/dev$
```

Hình 3 Các groups có trong HĐH

### c. Cấu trúc quản lý dữ liệu file

Cấu trúc quản lý dữ liệu file được khai báo ở /linux/fs.h. Cấu trúc này chứa những con trỏ hàm trong một driver, cho phép định nghĩa các thao tác xử lý. Hình 4 là một số ví dụ thao tác trên cấu trúc này.

```
// Note: __user refers to a user-space address.
struct file_operations {
    struct module *owner;                // Pointer to the LKM that owns the structure
    loff_t (*llseek) (struct file *, loff_t, int); // Change current read/write position in a file
    ssize_t (*read) (struct file *, char __user *, size_t, loff_t *); // Used to retrieve data from the device
    ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *); // Used to send data to the device
    ssize_t (*aio_read) (struct kiocb *, const struct iovec *, unsigned long, loff_t); // Asynchronous read
    ssize_t (*aio_write) (struct kiocb *, const struct iovec *, unsigned long, loff_t); // Asynchronous write
    ssize_t (*read_iter) (struct kiocb *, struct iov_iter *); // possibly asynchronous read
    ssize_t (*write_iter) (struct kiocb *, struct iov_iter *); // possibly asynchronous write
    int (*iterate) (struct file *, struct dir_context *); // called when VFS needs to read the directory contents
    unsigned int (*poll) (struct file *, struct poll_table_struct *); // Does a read or write block?
    long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long); // Called by the ioctl system call
    long (*compat_ioctl) (struct file *, unsigned int, unsigned long); // Called by the ioctl system call
    int (*mmap) (struct file *, struct vm_area_struct *); // Called by mmap system call
    int (*mremap) (struct file *, struct vm_area_struct *); // Called by memory remap system call
    int (*open) (struct inode *, struct file *); // first operation performed on a device file
    int (*flush) (struct file *, fl_owner_t id); // called when a process closes its copy of the descriptor
    int (*release) (struct inode *, struct file *); // called when a file structure is being released
    int (*fsync) (struct file *, loff_t, loff_t, int datasync); // notify device of change in its FASYNC flag
    int (*aio_fsync) (struct kiocb *, int datasync); // synchronous notify device of change in its FASYNC flag
    int (*fasync) (int, struct file *, int); // asynchronous notify device of change in its FASYNC flag
    int (*lock) (struct file *, int, struct file_lock *); // used to implement file locking
};
```

Hình 4 Cấu trúc quản lý dữ liệu file

### d. Source code cho driver thiết bị

Source code này được đính kèm và có file README hướng dẫn đi kèm.