

**ĐẠI HỌC QUỐC GIA HÀ NỘI**  
**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ**



**BÁO CÁO THỰC HÀNH MÔN LẬP**  
**TRÌNH ĐIỀU KHIỂN THIẾT BỊ**  
**TUẦN 2**

**Học phần:** Lập trình điều khiển thiết bị – ELT3297  
**Giảng viên:** Hoàng Gia Hưng (LT), Dương Minh Ngọc (TH)  
**Sinh viên:** Đoàn Đức Mạnh  
**Ngành:** Kỹ thuật máy tính  
**Khóa:** QH-2022-I/CQ-E-CE2  
**Mã sinh viên:** 22022167  
**Email:** 22022167@vnu.edu.vn

**HÀ NỘI, 2025**

# Contents

<b>Chương 1.Lý thuyết về Linux Kernel Module (LKM)</b>	<b>3</b>
I.    Khái niệm và vị trí của LKM	3
II.   Cấu trúc cơ bản của một LKM	4
III.  Ghi log trong kernel	4
IV.  Biên dịch module ngoài kernel tree	5
V.    Nạp và gỡ module	5
VI.  Tham số module	6
VII.  Ngữ cảnh tiến trình trong kernel	6
VIII. Chia sẻ symbol giữa các module	6
IX.  Nguyên tắc lập trình module an toàn	7
X.    Kết luận	7
<b>Chương 2.Thực hành</b>	<b>7</b>
I.    Bài 1: Module <code>hello</code>	7
II.   Bài 2: Module <code>hello_param</code>	9
III.  Bài 3: Module in dãy số và tổng bình phương	11
IV.  Bài 4: Module <code>hello_array</code>	13
V.    Bài 5: Module <code>hello_multi</code>	16
VI.  Bài 6: Module <code>hello_proc</code>	18
VII.  Bài 7: Module <code>hello_sysfs</code>	20
VIII. Bài 8: Module Provider và Consumer	23

## Danh sách hình

1	Mối quan hệ giữa User Space và Kernel Space thông qua System Calls . .	3
2	Kiến trúc giao tiếp giữa User Space, Linux Kernel và phần cứng (HW) . .	3
3	Hai dạng xây dựng kernel . . . . .	3
4	Cấu trúc cơ bản của một LKM . . . . .	4
5	Kết quả chạy module <code>hello</code> sau khi nạp và gỡ. . . . .	8
6	Kết quả chạy module <code>hello_param</code> với tham số <code>whom=Jerry N=5</code> . . . . .	10
7	Kết quả chạy module <code>hello_ex3</code> với tham số <code>n=5</code> . . . . .	13
8	Kết quả chạy module <code>hello_array</code> với tham số <code>nums=1,2,3,4,5,6</code> . . . . .	15
9	Kết quả chạy module <code>hello_multi</code> . . . . .	17
10	Kết quả chạy module <code>hello_proc</code> . . . . .	20
11	Kết quả chạy module <code>hello_sysfs</code> với tham số <code>val</code> . . . . .	22
12	Kết quả thực thi của Provider và Consumer . . . . .	28

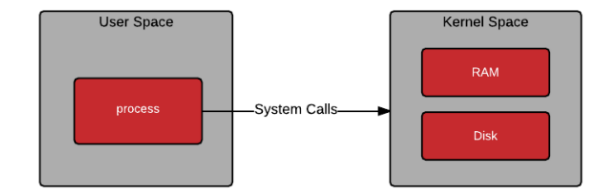
## Danh sách bảng

## Chương 1.: Lý thuyết về Linux Kernel Module (LKM)

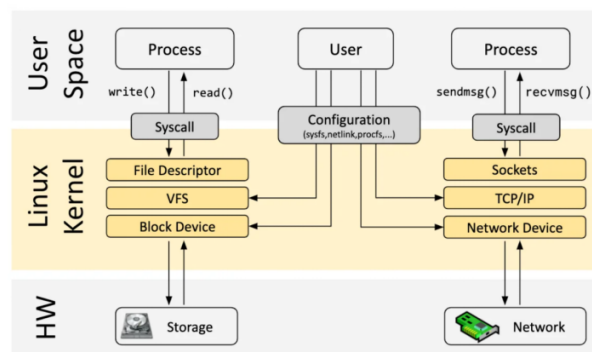
### I. Khái niệm và vị trí của LKM

Một hệ điều hành hiện đại thường được chia thành hai không gian chính:

- **User Space**: nơi chạy các chương trình ứng dụng của người dùng.
- **Kernel Space**: nơi chạy mã nhân (kernel) và quản lý tài nguyên hệ thống.



Hình 1: Mối quan hệ giữa User Space và Kernel Space thông qua System Calls



Hình 2: Kiến trúc giao tiếp giữa User Space, Linux Kernel và phần cứng (HW)

Trong Linux, kernel có thể được xây dựng theo hai dạng:

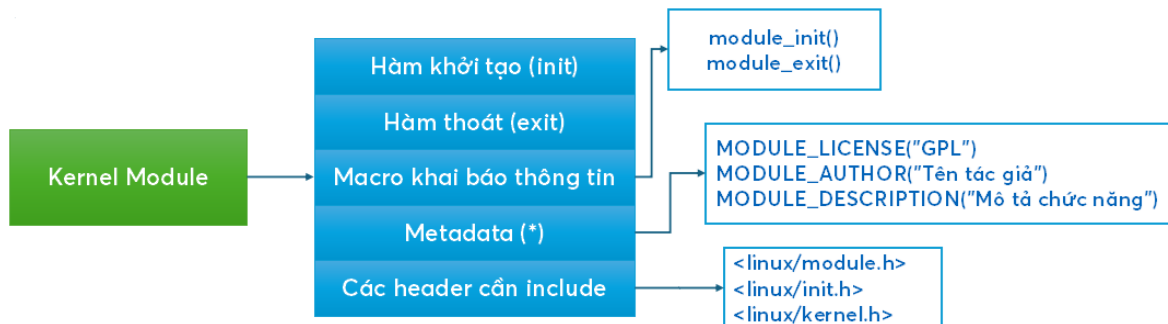
- Kernel Image Built-in**: tất cả chức năng (bao gồm cả driver) được biên dịch trực tiếp vào nhân, tạo thành một khối nguyên khối.
- Kernel Module (LKM)**: một số chức năng được tách thành các mô-đun độc lập, có thể nạp (load) hoặc gỡ (unload) động trong khi hệ thống đang chạy mà không cần khởi động lại.



Hình 3: Hai dạng xây dựng kernel

## II. Cấu trúc cơ bản của một LKM

Một LKM là một đoạn mã C đặc biệt, được biên dịch thành file đối tượng có phần mở rộng `.ko`, và có thể được nạp vào kernel khi cần. Thành phần cơ bản:



Hình 4: Cấu trúc cơ bản của một LKM

- **Hàm khởi tạo (init):** gọi khi module được nạp, thường để cấp phát tài nguyên, đăng ký driver, tạo device file. Trả về 0 nếu thành công, giá trị âm nếu lỗi.
- **Hàm thoát (exit):** gọi khi module bị gỡ, dùng để giải phóng tài nguyên, huỷ đăng ký driver, dọn dẹp bộ nhớ.
- **Macro đăng ký:** `module_init()` và `module_exit()` liên kết hàm init/exit với cơ chế nạp/gỡ module.
- **Metadata:** mô tả thông tin về module, ví dụ:

```

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Tên tác giả");
MODULE_DESCRIPTION("Mô tả chức năng");
  
```

- **Các header thường dùng:**
  - `<linux/module.h>` : khai báo macro init/exit, license.
  - `<linux/init.h>` : định nghĩa từ khoá `__init`, `__exit`.
  - `<linux/kernel.h>` : các hàm tiện ích như `printk`.
  - `<linux/fs.h>`, `<linux/uaccess.h>`: dùng cho driver thao tác file.
  - `<linux/sched.h>`: truy xuất thông tin tiến trình (`current`).

## III. Ghi log trong kernel

Kernel không sử dụng `printf`, mà dùng `printk()` với mức độ ưu tiên:

- KERN\_EMERG: sự cố nghiêm trọng, hệ thống không thể dùng.
- KERN\_ALERT: cảnh báo cần xử lý ngay.
- KERN\_ERR: lỗi nghiêm trọng.
- KERN\_WARNING: cảnh báo.
- KERN\_INFO: thông tin chung.
- KERN\_DEBUG: thông tin debug chi tiết.

## IV. Biên dịch module ngoài kernel tree

Một module không thể biên dịch trực tiếp bằng `gcc`, mà cần tận dụng hệ thống build của kernel. Do đó cần Makefile đặc biệt:

```
obj-m := hello.o
KDIR := /lib/modules/$(shell uname -r)/build
PWD := $(shell pwd)
```

```
all:
    $(MAKE) -C $(KDIR) M=$(PWD) modules
```

```
clean:
    $(MAKE) -C $(KDIR) M=$(PWD) clean
```

Cấu trúc này cho phép build module ngoài cây mã nguồn kernel (out-of-tree module).

## V. Nạp và gỡ module

Sau khi biên dịch, module tạo ra file `.ko`. Các lệnh cơ bản:

- `insmod <module>.ko [tham_số=giá_trị]` : nạp trực tiếp module, không kiểm tra phụ thuộc.
- `modprobe <module> [tham_số=giá_trị]` : nạp module và tự động xử lý phụ thuộc.
- `rmmod <module>` : gỡ module.
- `lsmod` : liệt kê các module đang nạp.

## VI. Tham số module

Module có thể nhận tham số để điều chỉnh hành vi:

- Khai báo cơ bản:

```
module_param(name, type, perm);  
MODULE_PARM_DESC(name, "Mô tả tham số");
```

- Đặt tên tham số khác tên biến:

```
module_param_named(param_name, variable, type, perm);
```

- Tham số có thể đọc/ghi qua sysfs: `/sys/module/<module>/parameters/`.

## VII. Ngữ cảnh tiến trình trong kernel

Khi một module chạy, nó hoạt động trong ngữ cảnh tiến trình gọi. Một số trường phổ biến:

- `current->pid`: PID của tiến trình.
- `current->tgid`: Thread group ID.
- `current->comm`: tên tiến trình (tối đa 16 ký tự).
- `current->state`: trạng thái tiến trình.
- `current->prio`: độ ưu tiên.

## VIII. Chia sẻ symbol giữa các module

- **Symbol**: tên đại diện cho hàm/biến toàn cục, ánh xạ tới địa chỉ trong bộ nhớ.
- Kernel duy trì **Kernel Symbol Table** chứa các symbol được export.
- Xem symbol: `cat /proc/kallsyms`.
- Export symbol:

```
EXPORT_SYMBOL(symbol_name);  
EXPORT_SYMBOL_GPL(symbol_name);
```

- Import symbol bằng `extern` và gọi trực tiếp trong module khác.

## IX. Nguyên tắc lập trình module an toàn

- Luôn kiểm tra giá trị trả về của hàm kernel: `kmalloc`, `register_chrdev`, `kthread_run`, v.v.
- Trả về mã lỗi chuẩn: `-ENOMEM`, `-EINVAL`, `-EFAULT`, ...
- Giải phóng tài nguyên đúng cách: tài nguyên cấp phát trong `init` phải được thu hồi trong `exit`.
- Nếu `init` thất bại giữa chừng, cần giải phóng tất cả tài nguyên đã cấp phát trước đó.

## X. Kết luận

Linux Kernel Module là nền tảng quan trọng để mở rộng kernel và viết device driver. Việc hiểu cấu trúc cơ bản, quy trình biên dịch, cách nạp/gỡ, sử dụng tham số và chia sẻ symbol giúp sinh viên xây dựng được các module thực tiễn và tiến tới lập trình driver hoàn chỉnh.

## Chương 2.: Thực hành

Mã nguồn thực hành có thể tham khảo tại:



### I. Bài 1: Module hello

#### Đề bài

Viết module tên **hello** với các yêu cầu sau:

- Khi nạp module: in ra log `hello`: `Hello, Kernel!`.
- Khi gỡ module: in ra log `hello`: `Goodbye, Kernel!`.

#### Mã nguồn

```
1 // hello.c
2 #include <linux/module.h>
3 #include <linux/init.h>
4 #include <linux/kernel.h>
```



```

5
6 static int __init hello_init(void) {
7     printk(KERN_INFO "hello:␣Hello,␣Kernel!\n");
8     return 0;
9 }
10
11 static void __exit hello_exit(void) {
12     printk(KERN_INFO "hello:␣Goodbye,␣Kernel!\n");
13 }
14
15 module_init(hello_init);
16 module_exit(hello_exit);
17
18 MODULE_LICENSE("GPL");
19 MODULE_AUTHOR("Jerry");
20 MODULE_DESCRIPTION("Exercise␣1␣-␣Hello␣Kernel␣Module");

```

```

1 // Makefile
2 obj-m := hello.o
3 KDIR := $(HOME)/linux-5.15.163
4 PWD  := $(shell pwd)
5
6 all:
7     $(MAKE) -C $(KDIR) M=$(PWD) modules
8
9 clean:
10    $(MAKE) -C $(KDIR) M=$(PWD) clean

```

## Kết quả chạy

```

student@uetsys:~/modules$ sudo insmod hello.ko
student@uetsys:~/modules$ sudo rmmod hello
student@uetsys:~/modules$ dmesg | tail -5
[ 193.792835] hello: loading out-of-tree module taints kernel.
[ 193.810942] Hello, Kernel!
[ 201.092898] Goodbye, Kernel!
[ 242.804346] Hello, Kernel!
[ 250.708006] Goodbye, Kernel!
student@uetsys:~/modules$

```

Hình 5: Kết quả chạy module hello sau khi nạp và gỡ.

## Giải thích

- Hàm `hello_init` được đánh dấu `__init` và đăng ký bằng `module_init`, được gọi khi nạp module. Nó in thông báo “Hello, Kernel!”.
- Hàm `hello_exit` được đánh dấu `__exit` và đăng ký bằng `module_exit`, được gọi khi gỡ module. Nó in “Goodbye, Kernel!”.

- `printk` là hàm in log trong kernel, ở mức `KERN_INFO`.
- Metadata: `MODULE_LICENSE`, `MODULE_AUTHOR`, `MODULE_DESCRIPTION` cung cấp thông tin cho module.

## Kết luận

Module `hello` minh họa cơ chế cơ bản của một LKM: định nghĩa hàm khởi tạo, hàm hủy, đăng ký chúng với kernel và in thông điệp vào log. Đây là bước khởi đầu để làm quen với lập trình nhân Linux.

## II. Bài 2: Module `hello_param`

### Đề bài

Viết module kernel tên `hello_param` với các yêu cầu:

- Có 2 tham số truyền từ người dùng:
  - `whom` – chuỗi ký tự, mặc định là `"world"`.
  - `howmany` (trong mã gọi là `N`) – số nguyên, mặc định là 1.
- Khi nạp module, in ra log `howmany` lần thông điệp:

```
hello_param: Hello, <whom>! [i/N]
```

- Khi gỡ module, in ra log:

```
hello_param: Bye, <whom>!
```

### Mã nguồn C

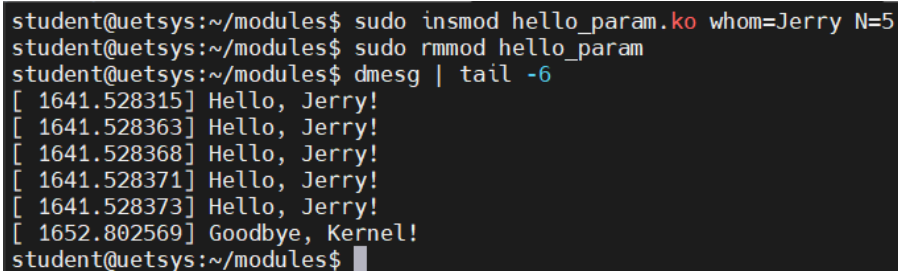
```
1 // hello_param.c
2 #include <linux/module.h>
3 #include <linux/init.h>
4 #include <linux/kernel.h>
5
6 static int N = 1;
7 module_param(N, int, 0644);
8 MODULE_PARM_DESC(N, "Number_of_iterations");
9
10 static char *whom = "world";
11 module_param(whom, charp, 0644);
```

```
12 MODULE_PARM_DESC(whom, "Name to greet when the module is loaded");
13
14 static int __init hello_init(void) {
15     int i;
16     for (i = 0; i < N; i++) {
17         printk(KERN_INFO "Hello, %s!\n", whom);
18     }
19     return 0;
20 }
21
22 static void __exit hello_exit(void) {
23     printk(KERN_INFO "Goodbye, Kernel!\n");
24 }
25
26 module_init(hello_init);
27 module_exit(hello_exit);
28
29 MODULE_LICENSE("GPL");
30 MODULE_AUTHOR("Jerry");
31 MODULE_DESCRIPTION("Exercise 2 - Hello Parameter Module");
```

## Makefile

```
1 obj-m := hello_param.o
2 KDIR := $(HOME)/linux-5.15.163
3 PWD := $(shell pwd)
4
5 all:
6     $(MAKE) -C $(KDIR) M=$(PWD) modules
7
8 clean:
9     $(MAKE) -C $(KDIR) M=$(PWD) clean
```

## Kết quả chạy



```
student@uetsys:~/modules$ sudo insmod hello_param.ko whom=Jerry N=5
student@uetsys:~/modules$ sudo rmmod hello_param
student@uetsys:~/modules$ dmesg | tail -6
[ 1641.528315] Hello, Jerry!
[ 1641.528363] Hello, Jerry!
[ 1641.528368] Hello, Jerry!
[ 1641.528371] Hello, Jerry!
[ 1641.528373] Hello, Jerry!
[ 1652.802569] Goodbye, Kernel!
student@uetsys:~/modules$
```

Hình 6: Kết quả chạy module hello\_param với tham số whom=Jerry N=5.

## Giải thích chi tiết

- Hai tham số được khai báo bằng macro `module_param`:
  - `N` (int, quyền 0644) → cho phép đọc/ghi giá trị số vòng lặp từ sysfs.
  - `whom` (charp, quyền 0644) → tham số chuỗi, chỉ định tên sẽ chào.
- Hàm `hello_init()` được gọi khi nạp module bằng `insmod`. Trong hàm, vòng lặp chạy từ 1 đến `N` và in ra log thông điệp:

```
hello_param: Hello, Jerry! [i/N]
```

với `i` là số thứ tự hiện tại.

- Hàm `hello_exit()` được gọi khi gỡ module bằng `rmmod`, in ra thông điệp chia tay với tên đã truyền.
- Lệnh kiểm thử:

```
sudo insmod hello_param.ko whom=Jerry N=5
sudo rmmod hello_param
dmesg | tail -6
```

cho kết quả 5 dòng chào và 1 dòng tạm biệt.

## Kết luận

Bài tập minh họa cách sử dụng `module_param` để truyền tham số từ người dùng vào module kernel. Nhờ đó, hành vi của module (số lần lặp, tên cần chào) có thể thay đổi linh hoạt khi nạp, thay vì cố định trong mã nguồn.

## III. Bài 3: Module in dãy số và tổng bình phương

### Đề bài

Viết một module kernel nhận tham số `n` (số nguyên, mặc định = 1):

- a. Khi nạp module: in ra các số nguyên từ 1 đến `n`.
- b. Khi nạp module: tính tổng bình phương các số từ 1 đến `n` và in kết quả ra log.

Khi gỡ module: in ra thông điệp "Done with `n=<giá trị>`".

## Mã nguồn C

```
1 // hello_ex3.c
2 #include <linux/module.h>
3 #include <linux/init.h>
4 #include <linux/kernel.h>
5
6 static int n = 1;
7 module_param(n, int, 0644);
8 MODULE_PARM_DESC(n, "Number_of_iterations");
9
10 static int __init hello_init(void) {
11     int i;
12     int sum = 0;
13     for (i = 1; i <= n; i++) {
14         printk(KERN_INFO "i=%d\n", i);
15         sum += i * i;
16     }
17     printk(KERN_INFO "Sum=%d\n", sum);
18     return 0;
19 }
20
21 static void __exit hello_exit(void) {
22     printk(KERN_INFO "Done_with_n=%d\n", n);
23 }
24
25 module_init(hello_init);
26 module_exit(hello_exit);
27
28 MODULE_LICENSE("GPL");
29 MODULE_AUTHOR("Jerry");
30 MODULE_DESCRIPTION("Excercise_3");
```

## Makefile

```
1 obj-m := hello_ex3.o
2 KDIR := $(HOME)/linux-5.15.163
3 PWD  := $(shell pwd)
4
5 all:
6     $(MAKE) -C $(KDIR) M=$(PWD) modules
7
8 clean:
9     $(MAKE) -C $(KDIR) M=$(PWD) clean
```

## Kết quả chạy

```
student@uetsys:~/modules$ sudo insmod hello_ex3.ko n=5
[sudo] password for student:
student@uetsys:~/modules$ sudo rmmod hello_ex3
student@uetsys:~/modules$ dmesg | tail -7
[ 3493.801853] i = 1
[ 3493.803372] i = 2
[ 3493.804110] i = 3
[ 3493.804132] i = 4
[ 3493.804138] i = 5
[ 3493.804159] Sum = 55
[ 3514.414360] Done with n = 5
student@uetsys:~/modules$
```

Hình 7: Kết quả chạy module `hello_ex3` với tham số `n=5`.

## Giải thích

- Tham số `n` được khai báo bằng `module_param`, cho phép truyền từ dòng lệnh khi nạp module.
- Trong hàm `hello_init()`, vòng lặp từ 1 đến `n` được thực hiện:
  - In ra từng giá trị `i`.
  - Tính tổng bình phương bằng cách cộng `i*i`.
- Kết quả tổng bình phương được in ra sau vòng lặp.
- Trong hàm `hello_exit()`, khi gỡ module bằng `rmmod`, in ra thông điệp xác nhận kết thúc với giá trị `n`.

## Kết luận

Bài tập minh họa cách sử dụng tham số module và thao tác với vòng lặp trong kernel module. Nó cũng cho thấy cách module có thể tính toán giá trị và in log ra `dmesg`, giúp kiểm chứng việc truyền tham số từ người dùng vào nhân Linux.

## IV. Bài 4: Module `hello_array`

### Đề bài

Viết một module kernel có tham số `nums` là một mảng số nguyên (tối đa 10 phần tử).

- Khi nạp module: in ra từng giá trị trong `nums` và tính tổng của chúng.
- Khi gỡ module: in ra dòng `"hello_array: unloaded"`.

## Mã nguồn C

```
1 // hello_array.c
2 #include <linux/module.h>
3 #include <linux/init.h>
4 #include <linux/kernel.h>
5
6 static int nums[10];
7 static int nums_count;
8 module_param_array(nums, int, &nums_count, 0444);
9 MODULE_PARM_DESC(nums, "Array of up to 10 integers");
10
11 static int __init hello_init(void) {
12     int i;
13     int sum = 0;
14     int count = nums_count <= 10 ? nums_count : 10;
15     for (i = 0; i < count; i++) {
16         printk(KERN_INFO "nums[%d]=%d\n", i, nums[i]);
17         sum += nums[i];
18     }
19     printk(KERN_INFO "Sum=%d\n", sum);
20     return 0;
21 }
22
23 static void __exit hello_exit(void) {
24     printk(KERN_INFO "hello_array: unloaded\n");
25 }
26
27 module_init(hello_init);
28 module_exit(hello_exit);
29
30 MODULE_LICENSE("GPL");
31 MODULE_AUTHOR("Jerry");
32 MODULE_DESCRIPTION("Excercise 4 - Hello Array");
```

## Makefile

```
1 obj-m := hello_array.o
2 KDIR := $(HOME)/linux-5.15.163
3 PWD := $(shell pwd)
4
5 all:
6     $(MAKE) -C $(KDIR) M=$(PWD) modules
7
8 clean:
9     $(MAKE) -C $(KDIR) M=$(PWD) clean
```

## Kết quả chạy

```
student@uetsys:~/modules$ sudo insmod hello_array.ko nums=1,2,3,4,5,6
[sudo] password for student:
student@uetsys:~/modules$ sudo rmmod hello_array
student@uetsys:~/modules$ dmesg | tail -8
[ 4776.948133] nums[0] = 1
[ 4776.949819] nums[1] = 2
[ 4776.949867] nums[2] = 3
[ 4776.949874] nums[3] = 4
[ 4776.949877] nums[4] = 5
[ 4776.949880] nums[5] = 6
[ 4776.949890] Sum = 21
[ 4786.691911] hello_array: unloaded
student@uetsys:~/modules$
```

Hình 8: Kết quả chạy module `hello_array` với tham số `nums=1,2,3,4,5,6`.

## Giải thích chi tiết

- `static int nums[10];` khai báo một mảng 10 phần tử kiểu số nguyên để lưu giá trị người dùng truyền vào.
- `static int nums_count;` lưu số lượng phần tử thực tế được truyền.
- `module_param_array(nums, int, &nums_count, 0444);` cho phép truyền một mảng số nguyên khi nạp module (ví dụ: `insmod hello_array.ko nums=1,2,3,4,5,6`).
- Trong `hello_init()`:
  - Biến `count` được xác định là số phần tử hợp lệ (không vượt quá 10).
  - Vòng lặp duyệt qua từng phần tử của mảng, in ra giá trị và cộng dồn vào biến `sum`.
  - Sau vòng lặp, in ra tổng các phần tử.
- Trong `hello_exit()`: khi gỡ module bằng `rmmod`, in ra thông điệp "hello\_array: unloaded".
- Khi chạy thử:

```
sudo insmod hello_array.ko nums=1,2,3,4,5,6
sudo rmmod hello_array
dmesg | tail -8
```

kết quả in ra các giá trị từ 1 đến 6 và tổng bằng 21.

## Kết luận

Bài tập minh họa cách truyền tham số dạng mảng cho kernel module. Nó giúp làm quen với `module_param_array`, cách xử lý số phần tử thực tế và cách in log ra kernel thông qua `dmesg`.



## V. Bài 5: Module hello\_multi

### Đề bài

Viết một module kernel gồm hai file nguồn:

- `main.c`: định nghĩa hàm `init/exit`, gọi hàm từ file phụ.
- `helper.c`: định nghĩa hàm `helper_greet()` in log.
- Khi nạp module, in ra thông điệp `"hello_multi: module loaded"` và gọi `helper_greet()`.
- Khi gỡ module, in ra `"hello_multi: module unloaded"`.

### Mã nguồn C

File `helper.c`

```
1 #include <linux/kernel.h>
2 #include <linux/module.h>
3
4 void helper_greet(void)
5 {
6     printk(KERN_INFO "hello_multi: Greetings from helper_greet()!\n");
7 }
```

File `main.c`

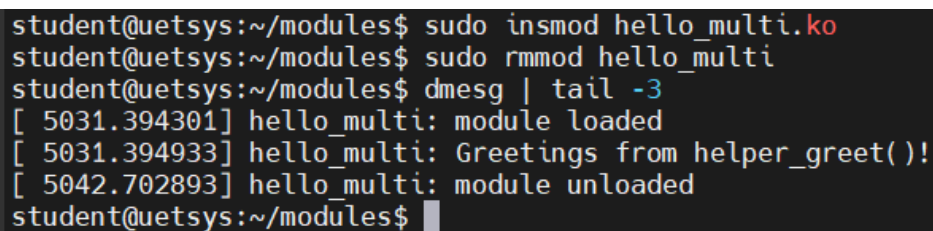
```
1 #include <linux/module.h>
2 #include <linux/init.h>
3 #include <linux/kernel.h>
4
5 /* prototype of helper_greet() defined in helper.c */
6 void helper_greet(void);
7
8 static int __init hello_multi_init(void)
9 {
10     printk(KERN_INFO "hello_multi: module loaded\n");
11
12     /* call function from helper.c */
13     helper_greet();
14
15     return 0;
16 }
17
```

```
18 static void __exit hello_multi_exit(void)
19 {
20     printk(KERN_INFO "hello_multi: module unloaded\n");
21 }
22
23 module_init(hello_multi_init);
24 module_exit(hello_multi_exit);
25
26 MODULE_LICENSE("GPL");
27 MODULE_AUTHOR("Jerry");
28 MODULE_DESCRIPTION("Exercise 5 - Hello Multi-file Module");
```

## Makefile

```
1 obj-m := hello_multi.o
2 hello_multi-objs := main.o helper.o
3 KDIR := $(HOME)/linux-5.15.163
4 PWD := $(shell pwd)
5
6 all:
7     $(MAKE) -C $(KDIR) M=$(PWD) modules
8
9 clean:
10    $(MAKE) -C $(KDIR) M=$(PWD) clean
```

## Kết quả chạy



```
student@uetsys:~/modules$ sudo insmod hello_multi.ko
student@uetsys:~/modules$ sudo rmmod hello_multi
student@uetsys:~/modules$ dmesg | tail -3
[ 5031.394301] hello_multi: module loaded
[ 5031.394933] hello_multi: Greetings from helper_greet()!
[ 5042.702893] hello_multi: module unloaded
student@uetsys:~/modules$
```

Hình 9: Kết quả chạy module hello\_multi.

## Giải thích chi tiết

- Module này được chia thành hai phần:
  - helper.c định nghĩa hàm `helper_greet()`, in ra thông báo khi được gọi.
  - main.c quản lý vòng đời module với hai hàm `hello_multi_init()` và `hello_multi_exit()`.
- Khi nạp module bằng `insmod`, hàm `hello_multi_init()` chạy trước:

- In ra thông điệp `"hello_multi: module loaded"`.
- Gọi hàm `helper_greet()` từ file `helper.c` để in ra thông báo bổ sung.
- Khi gỡ module bằng `rmmod`, hàm `hello_multi_exit()` chạy và in ra `"hello_multi: module unloaded"`.
- Makefile sử dụng biến `hello_multi-objs` để biên dịch nhiều file nguồn (`main.o` và `helper.o`) thành một module `hello_multi.ko`.
- Thực nghiệm với:

```
sudo insmod hello_multi.ko
sudo rmmod hello_multi
dmesg | tail -3
```

cho ra ba dòng log: loaded → greetings từ helper → unloaded.

## Kết luận

Bài tập này minh họa cách xây dựng một kernel module từ nhiều file nguồn khác nhau. Qua đó, ta thấy rõ cách tổ chức code thành các phần riêng biệt (logic chính và hàm phụ), giúp chương trình dễ bảo trì và mở rộng hơn.

## VI. Bài 6: Module `hello_proc`

### Đề bài

Viết module kernel `hello_proc` với yêu cầu:

- Khi nạp module, in ra:
  - Tên tiến trình hiện tại (`current->comm`).
  - PID của tiến trình hiện tại (`current->pid`).
- Khi gỡ module, in ra thông điệp `"hello_proc: unloaded"`.

Ví dụ log mong muốn:

```
hello_proc: loaded by process "insmod" (pid=1234)
hello_proc: unloaded
```

## Mã nguồn C

```
1 // hello_proc.c
2 #include <linux/module.h>
3 #include <linux/init.h>
4 #include <linux/kernel.h>
5 #include <linux/sched.h>
6 #include <linux/sched/task.h>
7
8 static int __init hello_init(void) {
9     printk(KERN_INFO "hello_proc: loaded by process \"%(pid=%d
10         )\n",
11         current->comm, current->pid);
12     return 0;
13 }
14
15 static void __exit hello_exit(void) {
16     printk(KERN_INFO "hello_proc: unloaded\n");
17 }
18
19 module_init(hello_init);
20 module_exit(hello_exit);
21
22 MODULE_LICENSE("GPL");
23 MODULE_AUTHOR("Jerry");
24 MODULE_DESCRIPTION("Excercise 6");
```

## Makefile

```
1 obj-m := hello_proc.o
2 KDIR := $(HOME)/linux-5.15.163
3 PWD := $(shell pwd)
4
5 all:
6     $(MAKE) -C $(KDIR) M=$(PWD) modules
7
8 clean:
9     $(MAKE) -C $(KDIR) M=$(PWD) clean
```

## Kết quả chạy

```
student@uetsys:~/modules$ sudo insmod hello_proc.ko
student@uetsys:~/modules$ sudo rmmod hello_proc
student@uetsys:~/modules$ dmesg | tail -2
[ 5638.352433] hello_proc: loaded by process "insmod" (pid=477)
[ 5647.440971] hello_proc: unloaded
student@uetsys:~/modules$ █
```

Hình 10: Kết quả chạy module `hello_proc`.

## Giải thích chi tiết

- Biến `current` là một con trỏ toàn cục trong kernel, luôn trỏ đến cấu trúc `task_struct` của tiến trình hiện tại.
- Trường `current->comm` lưu tên tiến trình (ví dụ: "insmod" khi ta nạp module).
- Trường `current->pid` chứa Process ID của tiến trình đó.
- Trong hàm `hello_init()`, khi chạy lệnh `insmod hello_proc.ko`, kernel in ra:

```
hello_proc: loaded by process "insmod" (pid=XXX)
```

- Trong hàm `hello_exit()`, khi gọi `rmmod hello_proc`, kernel in ra:

```
hello_proc: unloaded
```

- Makefile biên dịch file `hello_proc.c` thành module `hello_proc.ko`.

## Kết luận

Bài tập minh họa cách truy cập thông tin về tiến trình hiện tại trong kernel (`current`), bao gồm tên và PID. Đây là bước cơ bản để hiểu cách kernel theo dõi các tiến trình và quản lý trạng thái khi thực thi module.

## VII. Bài 7: Module `hello_sysfs`

### Đề bài

Viết module kernel `hello_sysfs` với yêu cầu:

- Có tham số `val` (số nguyên, mặc định = 0).

- Khai báo `module_param(val, int, 0644)` để cho phép đọc/ghi.
- Khi nạp module, in ra `"hello_sysfs: loaded with val=<giá trị>"`.
- Khi gỡ module, in ra `"hello_sysfs: unloaded"`.
- Cho phép thay đổi `val` trong khi module đang chạy thông qua sysfs:

```
echo 42 | sudo tee /sys/module/hello_sysfs/parameters/val
cat /sys/module/hello_sysfs/parameters/val
```

## Mã nguồn C

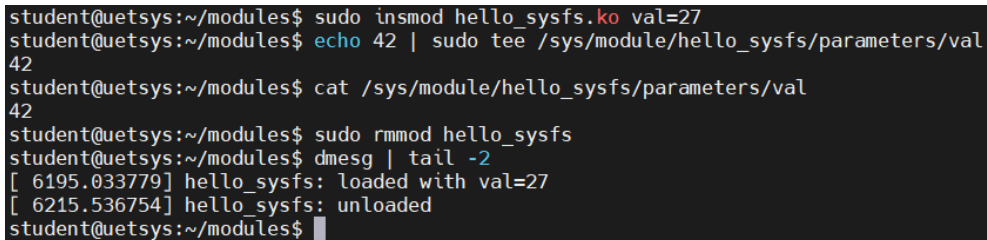
```
1 // hello_sysfs.c
2 #include <linux/module.h>
3 #include <linux/init.h>
4 #include <linux/kernel.h>
5 #include <linux/sched.h>
6 #include <linux/sched/task.h>
7
8 static int val = 0;
9 module_param(val, int, 0644);
10 MODULE_PARM_DESC(val, "Input parameter");
11
12 static int __init hello_init(void) {
13     printk(KERN_INFO "hello_sysfs: loaded with val=%d\n", val);
14     return 0;
15 }
16
17 static void __exit hello_exit(void) {
18     printk(KERN_INFO "hello_sysfs: unloaded\n");
19 }
20
21 module_init(hello_init);
22 module_exit(hello_exit);
23
24 MODULE_LICENSE("GPL");
25 MODULE_AUTHOR("Jerry");
26 MODULE_DESCRIPTION("Excercise 7 - sysfs");
```

## Makefile

```
1 obj-m := hello_sysfs.o
2 KDIR := $(HOME)/linux-5.15.163
3 PWD := $(shell pwd)
4
5 all:
6     $(MAKE) -C $(KDIR) M=$(PWD) modules
```

```
7
8 clean:
9      $(MAKE) -C $(KDIR) M=$(PWD) clean
```

## Kết quả chạy



```
student@uetsys:~/modules$ sudo insmod hello_sysfs.ko val=27
student@uetsys:~/modules$ echo 42 | sudo tee /sys/module/hello_sysfs/parameters/val
42
student@uetsys:~/modules$ cat /sys/module/hello_sysfs/parameters/val
42
student@uetsys:~/modules$ sudo rmmod hello_sysfs
student@uetsys:~/modules$ dmesg | tail -2
[ 6195.033779] hello_sysfs: loaded with val=27
[ 6215.536754] hello_sysfs: unloaded
student@uetsys:~/modules$
```

Hình 11: Kết quả chạy module `hello_sysfs` với tham số `val`.

## Giải thích chi tiết

- `static int val = 0;` khai báo biến toàn cục `val` với giá trị mặc định 0.
- `module_param(val, int, 0644);` cho phép kernel module nhận tham số `val` từ người dùng và ánh xạ nó vào `sysfs` với quyền đọc/ghi.
- Trong hàm `hello_init()`:

- Khi nạp module bằng `insmod hello_sysfs.ko val=27`, log kernel in ra:

```
hello_sysfs: loaded with val=27
```

- Trong hàm `hello_exit()`:

- Khi gỡ module bằng `rmmod`, log kernel in ra:

```
hello_sysfs: unloaded
```

- Nhờ `sysfs`, tham số có thể thay đổi khi module đang chạy:

```
echo 42 | sudo tee /sys/module/hello_sysfs/parameters/val
cat /sys/module/hello_sysfs/parameters/val
```

Điều này cập nhật trực tiếp giá trị `val` trong module.

- Lệnh kiểm tra `dmesg` xác nhận hành vi nạp/gỡ hoạt động đúng.

## Kết luận

Bài tập minh họa cách sử dụng `module_param` với quyền 0644 để tham số module có thể được thay đổi trong khi chạy thông qua `sysfs`. Đây là cơ chế quan trọng để giao tiếp giữa kernel module và người dùng, giúp tinh chỉnh hành vi module động mà không cần nạp lại.

## VIII. Bài 8: Module Provider và Consumer

**Đề bài:** Viết một module Provider cung cấp hàm `compute_stats` để tính các thông số thống kê cơ bản của một mảng số nguyên gồm:

- Tổng (`sum`)
- Giá trị trung bình (`avg`)
- Giá trị cực đại (`max`)
- Giá trị cực tiểu (`min`)

Kết quả được trả về thông qua một `struct stats_result`. Hàm phải kiểm tra tính hợp lệ của tham số đầu vào và trả lỗi nếu cần. Viết một module Consumer sử dụng hàm `compute_stats` đã `EXPORT_SYMBOL` từ Provider, gọi thử trên một mảng số nguyên mẫu (ví dụ `{10,20,30,40,50}`), và in kết quả ra log kernel bằng `printk`.

—

### Mã nguồn:

File `stats.h`

```
1 #ifndef _STATS_H_
2 #define _STATS_H_
3
4 #include <linux/types.h>
5
6 struct stats_result {
7     long long sum;
8     long avg;
9     int max;
10    int min;
11 };
12
13 #endif /* _STATS_H_ */
```

### Giải thích:

- Đây là file header dùng chung cho cả Provider và Consumer.



- Sử dụng cơ chế `#ifndef ... #define ... #endif` để tránh việc định nghĩa lặp lại (include guard).
- `#include <linux/types.h>` nhằm đảm bảo các kiểu dữ liệu chuẩn trong kernel như `long long`, `size_t` được định nghĩa đúng.
- Struct `stats_result` chứa 4 trường dữ liệu:
  - `sum`: tổng tất cả các phần tử (dùng `long long` để tránh tràn số).
  - `avg`: giá trị trung bình của mảng (sử dụng số nguyên, làm tròn xuống).
  - `max`: phần tử lớn nhất trong mảng.
  - `min`: phần tử nhỏ nhất trong mảng.
- Struct này sẽ được Provider điền dữ liệu và trả về cho Consumer.

File `provider.c`

```
1 #include <linux/module.h>
2 #include <linux/init.h>
3 #include <linux/kernel.h>
4 #include <linux/errno.h>
5 #include "stats.h"
6
7 int compute_stats(const int *arr, size_t len, struct stats_result
8 *out)
9 {
10     size_t i;
11     long long sum = 0;
12     int cur_max, cur_min;
13
14     if (!arr || !out)
15         return -EINVAL;
16     if (len == 0)
17         return -EINVAL;
18
19     cur_max = arr[0];
20     cur_min = arr[0];
21     sum = arr[0];
22
23     for (i = 1; i < len; i++) {
24         int v = arr[i];
25         sum += v;
26         if (v > cur_max)
27             cur_max = v;
28         if (v < cur_min)
29             cur_min = v;
30     }
31
32     out->sum = sum;
33     out->avg = (long)(sum / (long long)len);
```

```

33     out->max = cur_max;
34     out->min = cur_min;
35
36     return 0;
37 }
38 EXPORT_SYMBOL(compute_stats);
39
40 static int __init provider_init(void)
41 {
42     printk(KERN_INFO "provider: loaded\n");
43     return 0;
44 }
45 static void __exit provider_exit(void)
46 {
47     printk(KERN_INFO "provider: unloaded\n");
48 }
49 module_init(provider_init);
50 module_exit(provider_exit);
51
52 MODULE_LICENSE("GPL");
53 MODULE_AUTHOR("Jerry");
54 MODULE_DESCRIPTION("Provider module exporting compute_stats");

```

### Giải thích:

- File này định nghĩa module **Provider**, nhiệm vụ chính là cung cấp hàm `compute_stats()` để các module khác có thể sử dụng.
- Hàm `compute_stats`:
  - Nhận đầu vào là một mảng số nguyên `arr`, kích thước mảng `len`, và con trỏ tới `struct stats_result` để lưu kết quả.
  - Kiểm tra tính hợp lệ của tham số: nếu `arr` hoặc `out` là `NULL`, hoặc `len = 0`, hàm trả về `-EINVAL` (mã lỗi invalid argument).
  - Khởi tạo giá trị ban đầu: gán `cur_max`, `cur_min`, và `sum` bằng phần tử đầu tiên.
  - Duyệt mảng từ phần tử thứ 2, cập nhật `sum`, `cur_max`, `cur_min`.
  - Sau khi duyệt xong, ghi kết quả vào struct:
    - \* `out->sum`: tổng tất cả phần tử.
    - \* `out->avg`: trung bình số nguyên (chia lấy phần nguyên).
    - \* `out->max`: giá trị cực đại.
    - \* `out->min`: giá trị cực tiểu.
  - Hàm trả về 0 nếu tính toán thành công.
- Lệnh `EXPORT_SYMBOL(compute_stats);` cho phép hàm này được sử dụng bởi các module khác (ở đây là Consumer).

- Hai hàm `provider_init()` và `provider_exit()` in thông điệp khi module được nạp hoặc gỡ bỏ.
- Các macro `MODULE_LICENSE`, `MODULE_AUTHOR`, `MODULE_DESCRIPTION` cung cấp thông tin meta cho module.

File `consumer.c`

```

1 #include <linux/module.h>
2 #include <linux/init.h>
3 #include <linux/kernel.h>
4 #include "stats.h"
5
6 #define MAX_NUMS 10
7
8 static int nums[MAX_NUMS];
9 static int nums_count = 0;
10 module_param_array(nums, int, &nums_count, 0444);
11 MODULE_PARM_DESC(nums, "Array of integers (max 10) to compute stats on");
12
13 extern int compute_stats(const int *arr, size_t len, struct
14     stats_result *out);
15
16 static int __init consumer_init(void)
17 {
18     struct stats_result res;
19     int ret;
20     int count = nums_count;
21
22     if (count == 0) {
23         printk(KERN_ERR "consumer: no input nums provided\n");
24         return -EINVAL;
25     }
26     if (count > MAX_NUMS) {
27         printk(KERN_WARNING "consumer: nums_count > MAX_NUMS, truncating\n");
28         count = MAX_NUMS;
29     }
30
31     printk(KERN_INFO "consumer: loaded - calling compute_stats() on %d elements\n", count);
32
33     ret = compute_stats(nums, (size_t)count, &res);
34     if (ret) {
35         printk(KERN_ERR "consumer: compute_stats() failed: %d\n", ret);
36         return ret;
37     }

```

```

38     printk(KERN_INFO "consumer: stats->sum=%lld avg=%ld max=%d
        min=%d\n",
39             res.sum, res.avg, res.max, res.min);
40
41     return 0;
42 }
43 static void __exit consumer_exit(void)
44 {
45     printk(KERN_INFO "consumer: unloaded\n");
46 }
47 module_init(consumer_init);
48 module_exit(consumer_exit);
49
50 MODULE_LICENSE("GPL");
51 MODULE_AUTHOR("Jerry");
52 MODULE_DESCRIPTION("Consumer module using compute_stats from
    provider");

```

### Giải thích:

- File này định nghĩa module **Consumer**, nhiệm vụ chính là lấy dữ liệu từ tham số truyền vào, sau đó gọi hàm `compute_stats()` được export bởi Provider.
- Biến `nums` là một mảng số nguyên tối đa 10 phần tử. Biến `nums_count` giữ số lượng phần tử thực tế được truyền khi nạp module bằng `insmod`.
- Macro `module_param_array` cho phép truyền tham số mảng từ không gian người dùng xuống kernel, ví dụ:

```

1     sudo insmod consumer.ko nums=10,20,30,40,50

```

- Trong hàm `consumer_init`:
  - Nếu không có dữ liệu (`nums_count = 0`) thì báo lỗi và từ chối nạp module.
  - Nếu số phần tử vượt quá 10, in cảnh báo và chỉ lấy 10 phần tử đầu tiên.
  - Gọi hàm `compute_stats()` từ Provider để tính toán.
  - Nếu gọi hàm thành công, kết quả thống kê (`sum, avg, max, min`) sẽ được in ra log kernel.
- Hàm `consumer_exit` chỉ đơn giản in thông điệp khi module bị gỡ bỏ.
- Thông qua cách này, Consumer minh họa cơ chế tái sử dụng code giữa các module kernel nhờ `EXPORT_SYMBOL`.

### File Makefile

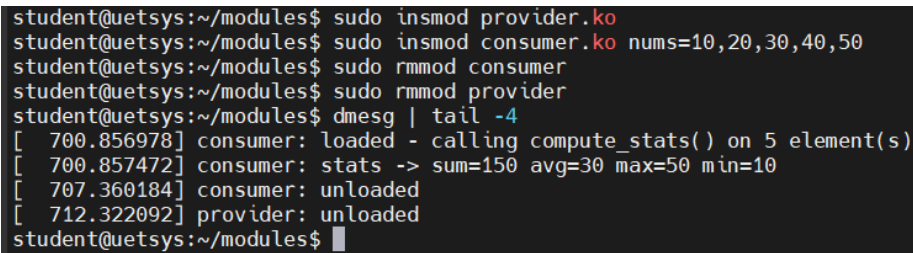
```

1 obj-m := provider.o consumer.o
2 KDIR := $(HOME)/linux-5.15.163
3 PWD := $(shell pwd)

```

```
4 all:
5     $(MAKE) -C $(KDIR) M=$(PWD) modules
6 clean:
7     $(MAKE) -C $(KDIR) M=$(PWD) clean
```

### Kết quả chạy:



```
student@uetsys:~/modules$ sudo insmod provider.ko
student@uetsys:~/modules$ sudo insmod consumer.ko nums=10,20,30,40,50
student@uetsys:~/modules$ sudo rmmod consumer
student@uetsys:~/modules$ sudo rmmod provider
student@uetsys:~/modules$ dmesg | tail -4
[ 700.856978] consumer: loaded - calling compute_stats() on 5 element(s)
[ 700.857472] consumer: stats -> sum=150 avg=30 max=50 min=10
[ 707.360184] consumer: unloaded
[ 712.322092] provider: unloaded
student@uetsys:~/modules$
```

Hình 12: Kết quả thực thi của Provider và Consumer

### Giải thích tổng quan:

- Bài tập được chia thành hai module:
  - **Provider:** định nghĩa và `EXPORT_SYMBOL` hàm `compute_stats()` để tính toán các thông số thống kê (tổng, giá trị trung bình, giá trị cực đại, giá trị cực tiểu) của một mảng số nguyên.
  - **Consumer:** nhận tham số mảng từ người dùng thông qua `module_param_array`, sau đó gọi hàm `compute_stats()` của Provider và in kết quả ra log kernel.
- Hai module minh họa cơ chế **chia sẻ hàm giữa các module kernel** bằng cách sử dụng `EXPORT_SYMBOL`.
- Nếu Consumer không nhận được dữ liệu đầu vào hợp lệ (ví dụ mảng rỗng), module sẽ báo lỗi và không khởi tạo thành công.
- Cấu trúc dự án gồm 4 file chính:
  - `stats.h`: định nghĩa cấu trúc dữ liệu dùng chung.
  - `provider.c`: triển khai hàm thống kê và export symbol.
  - `consumer.c`: sử dụng hàm từ Provider.
  - `Makefile`: biên dịch đồng thời cả hai module.
- Đây là một ví dụ điển hình về **thiết kế module theo mô hình Provider–Consumer**, tách biệt phần cung cấp dịch vụ (hàm xử lý) và phần sử dụng dịch vụ, giúp mã nguồn dễ mở rộng và tái sử dụng.

**Kết luận:** Bài tập minh họa cách chia tách chức năng giữa các module kernel: một module cung cấp hàm (Provider), module khác gọi và sử dụng (Consumer). Đây là cách thiết kế linh hoạt, dễ tái sử dụng và mở rộng, đồng thời làm quen với cơ chế `EXPORT_SYMBOL` trong Linux kernel.

## Tài liệu tham khảo

- a. Jonathan Corbet, Alessandro Rubini, Greg Kroah-Hartman, *Linux Device Drivers, 3rd Edition*, O'Reilly Media, 2005.
- b. Michael Kerrisk, *The Linux Programming Interface – A Linux and UNIX System Programming Handbook*, No Starch Press, 2010.

Hết