

Linux kernel module

Giới thiệu

Linux được thiết kế để làm việc với hàng tỉ thiết bị. Nhưng ta không thể đưa tất cả các driver vào trong kernel được, vì sẽ làm cho kích thước kernel rất lớn. Giải pháp cho vấn đề này đó là: thiết kế các driver dưới dạng module tách rời với kernel. Trong quá trình hoạt động, driver nào cần thiết sẽ được lắp vào kernel, còn driver nào không cần thiết sẽ bị tháo ra khỏi kernel (dynamic loading).

Như vậy, để trở thành lập trình viên Linux device driver, ta cần hiểu về Linux kernel module. Bài học này sẽ tập trung làm rõ các vấn đề liên quan tới Linux kernel module:

- Linux kernel module là gì? Linux kernel module và Linux driver có mối quan hệ gì?
- Cách viết một Linux kernel module.
- Cách biên dịch (build) một Linux kernel module.
- Cách lắp (load) module này vào và cách tháo (unload) module này ra khỏi kernel.

Linux kernel module

Linux kernel module là một file với tên mở rộng là (.ko). Nó sẽ được lắp vào hoặc tháo ra khỏi kernel khi cần thiết. Chính vì vậy, nó còn có một tên gọi khác là loadable kernel module. Một trong những kiểu loadable kernel module phổ biến đó là driver. Đến đây, các bạn có thể có một số thắc mắc:

- Thiết kế driver theo kiểu loadable module có ưu điểm gì?
- Có phải tất cả các driver đều là các loadable module không?
- Quá trình đưa kernel module vào trong kernel space diễn ra như thế nào?

Chúng ta sẽ lần lượt giải đáp các thắc mắc trên. Việc thiết kế driver theo kiểu loadable module mang lại 3 lợi ích:

- Giúp giảm kích thước kernel. Do đó, giảm sự lãng phí bộ nhớ và giảm thời gian khởi động hệ thống.
- Không phải biên dịch lại kernel khi thêm mới driver hoặc khi thay đổi driver.
- Không cần phải khởi động lại hệ thống khi thêm mới driver. Trong khi đối với Windows, mỗi khi cài thêm driver, ta phải khởi động lại hệ thống, điều này không thích hợp với các máy server.

Phần lớn các driver đều là các loadable kernel module, nhưng không phải là tất cả. Vẫn có một số driver được tích hợp luôn vào trong kernel, đặc biệt là các bus driver. Chúng được gọi là built-in driver. Các device driver thường sẽ là các loadable kernel module.

Ngược lại, không phải loadable kernel module nào cũng là driver, ví dụ kvm.ko là loadable kernel module nhưng không phải là driver. Trên thực tế, loadable kernel module được chia làm 3 loại chính: device driver, system call và file system.

Trong khóa học này, các thuật ngữ module, device driver, loadable kernel module, Linux kernel module, loadable module, kernel module đều được hiểu là một.

Khi cần một module nhưng nó lại chưa có trong kernel space, kernel sẽ đưa module ấy vào. Quá trình này có thể diễn ra một cách tự động, với trình tự sau:

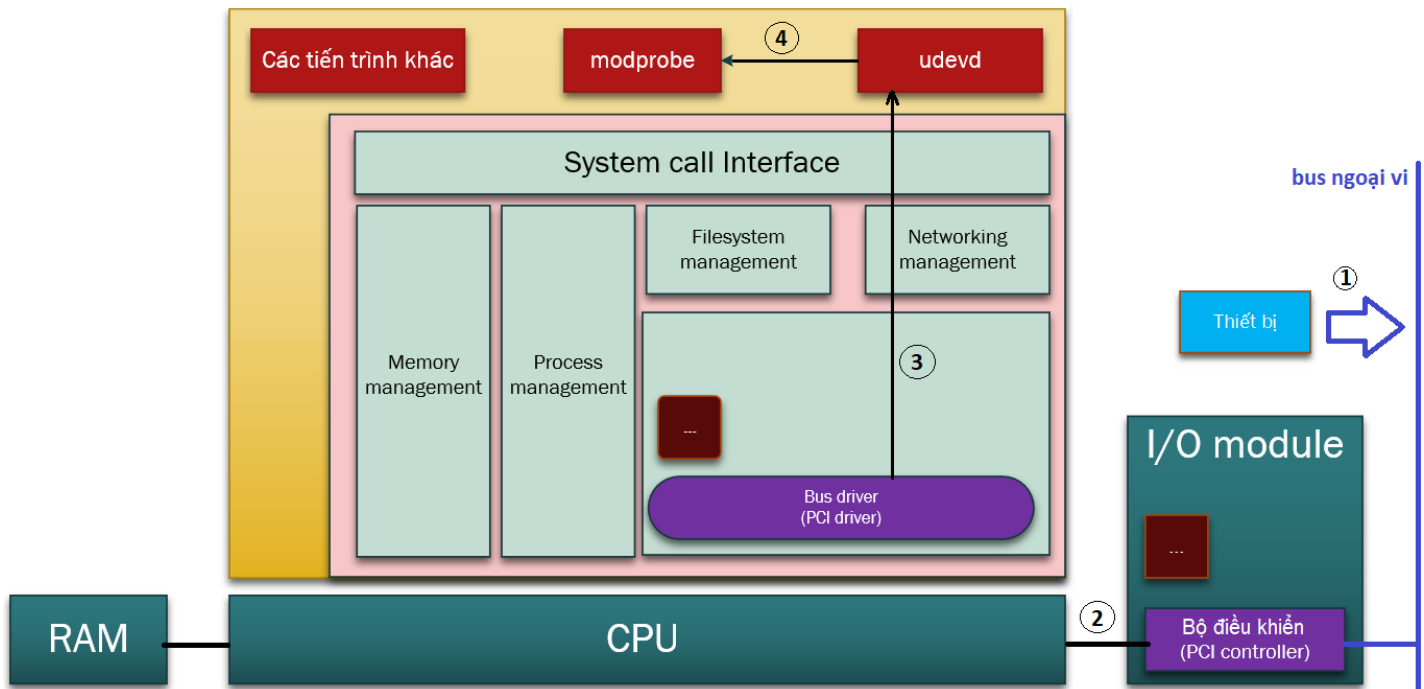
- Bước 1: Kernel kích hoạt tiến trình **modprobe** cùng với tham số truyền vào là tên của module (ví dụ xxx.ko).
- Bước 2: Tiến trình modprobe kiểm tra file `/lib/modules/<kernel-version>/modules.dep` xem xxx.ko có phụ thuộc vào module nào khác không. Giả sử xxx.ko phụ thuộc vào module yyy.ko.
- Bước 3: Tiến trình modprobe sẽ kích hoạt tiến trình **insmod** để đưa các module phụ thuộc vào trước (yyy.ko), rồi mới tới module cần thiết (xxx.ko).

Như vậy, các module được đưa vào kernel space dưới sự giúp đỡ của tiến trình modprobe. Câu hỏi đặt ra là: kernel kích hoạt tiến trình modprobe bằng cách nào?

- Cách 1 là sử dụng **kmod**. Đây là một thành phần của Linux kernel, hoạt động trong kernel space. Khi một thành phần nào đó của kernel cần đưa một module vào trong kernel space, nó sẽ truyền tên module cho

hàm **request_module** của kmod. Hàm request_module sẽ gọi hàm **call_usermodehelper_setup** để sinh ra tiến trình modprobe. Các bạn có thể tham khảo mã nguồn của kmod tại [/kernel/kmod.c](https://kernel.kmod.c).

- Cách 2 là sử dụng **udev** (hình 1). Đây là một tiến trình hoạt động trong user space. Nếu một thiết bị cắm vào hệ thống máy tính, thì điện trở trên bus ngoại vi (ví dụ PCI bus hoặc USB bus) sẽ thay đổi và bộ điều khiển (controller) sẽ biết điều này. Khi đó, bus driver sẽ gửi một bản tin lên cho tiến trình udevd. Bản tin này chứa thông tin về thiết bị. Tiến trình udevd sẽ tra cứu file `/lib/modules/<kernel-version>/modules.alias` để tìm ra driver nào tương thích với thiết bị. Sau đó, udevd sinh ra tiến trình modprobe.



Hình 1 Minh họa quá trình kích hoạt modprobe bằng udevd

case study

Bây giờ, chúng ta cùng tìm hiểu cách viết một Linux kernel module.

Bước 1: tạo thư mục cho bài học hôm nay

```
cd /home/ubuntu
mkdir ldd
cd ldd
mkdir phan_1
mkdir phan_1/bai_1_3
```

Bước 2: dùng lệnh "cd" di chuyển vào thư mục phan_1/bai_1_3, rồi dùng lệnh "vim hello.c" để mở một file có tên hello.c. Sau đó, sao chép đoạn mã sau vào file hello.c

```
/*
 * hello.c - ví dụ về linux kernel module
 */

#include <linux/module.h> /* thư viện này định nghĩa các macro như module_init và module_exit */

#define DRIVER_AUTHOR "Nguyen Tien Dat <dat.a3cbq91@gmail.com>"
#define DRIVER_DESC "A sample loadable kernel module"

static int __init init_hello(void)
{
    printk("Hello Vietnam\n");
    return 0;
}

static void __exit exit_hello(void)
```

```
{
    printk("Goodbye Vietnam\n");
}

module_init(init_hello);
module_exit(exit_hello);

MODULE_LICENSE("GPL"); /* giấy phép sử dụng của module */
MODULE_AUTHOR(DRIVER_AUTHOR); /* tác giả của module */
MODULE_DESCRIPTION(DRIVER_DESC); /* mô tả chức năng của module */
MODULE_SUPPORTED_DEVICE("testdevice"); /* kiểu device mà module hỗ trợ */
```

Do module hello cần dùng một số hàm hoặc macro của Linux kernel, nên chúng ta sẽ sử dụng từ khóa **#include** để chỉ rõ các file cần dùng. Điều này cũng tương tự như khi viết các chương trình ứng dụng trên user space, chúng ta cũng dùng **#include** cùng với tên của các thư viện. Do đó, có thể nói rằng, Linux kernel chính là một thư viện, cung cấp các hàm, các macro để chúng ta phát triển kernel module.

Trong ví dụ trên, ta chỉ cần tham chiếu tới file của Linux kernel là `<linux/module.h>`. File này chứa 2 macro quan trọng, là: **module_init()** và **module_exit()**. Do đó, dù viết bất cứ kernel module nào, ta cũng cần tham chiếu tới `<linux/module.h>`.

- **module_init** giúp xác định hàm nào sẽ được thực thi ngay sau khi lắp module vào kernel.
- **module_exit** giúp xác định hàm nào được thực thi ngay trước khi tháo module ra khỏi kernel.

Trong ví dụ trên, **init_hello()** là hàm được gọi ngay sau khi module hello được lắp vào, và **exit_hello()** là hàm được gọi ngay trước khi module hello bị tháo ra khỏi kernel.

Macro **__init** thường đi kèm với hàm khởi tạo. Trong ví dụ trên, macro **__init** xuất hiện trước tên hàm **init_hello**. Macro này giúp kernel biết rằng, hàm **init_hello()** chỉ phải thực thi lúc khởi tạo, nên vùng nhớ chứa hàm này có thể được giải phóng sau khi nó thực thi xong mà không ảnh hưởng gì.

Tương tự, macro **__exit** thường đi kèm với hàm kết thúc. Trong ví dụ trên, **__exit** xuất hiện trước tên hàm **exit_hello**. Macro này cho kernel biết, khi lắp module vào kernel thì chưa cần đưa hàm **exit_hello** vào trong bộ nhớ RAM. Chỉ khi chuẩn bị tháo module ra khỏi kernel, hàm **exit_hello** này mới cần được đưa vào RAM và thực thi.

Trong quá trình viết kernel module, các lập trình viên thường sử dụng hàm **printk** để ghi lại quá trình hoạt động của module. Việc này được gọi là logging. Mục đích của việc logging là để phục vụ quá trình gỡ lỗi sau này (debug). Ta có thể sử dụng lệnh **dmesg** để xem quá trình hoạt động của kernel kể từ lúc nó khởi động. Chúng ta sẽ tìm hiểu kỹ hơn về hàm **printk** trong [bài 3_1](#).

Các macro nằm ở cuối ví dụ trên cung cấp các thông tin về module. Ta có thể sử dụng lệnh **modinfo** để xem các thông tin của một module:

- Macro **MODULE_AUTHOR** cho biết ai là người tạo ra module.
- Macro **MODULE_DESCRIPTION()** cho biết module làm được những gì.
- Macro **MODULE_SUPPORTED_DEVICE()** cho biết module này hỗ trợ làm việc với những thiết bị nào.
- **MODULE_LICENSE** cho biết người dùng có cần phải trả phí nếu sử dụng module hay không. Trong ví dụ trên, giấy phép sử dụng module thuộc loại GPL. Với giấy phép sử dụng GPL, người dùng có thể sử dụng module miễn phí. Ngoài GPL, còn có các loại license như GPL v2, BSD/GPL, MIT/GPL, MPL/GPL.

Biên dịch kernel module

Để biên dịch kernel module, ta sử dụng phương pháp Kbuild. Theo phương pháp này, chúng ta cần tạo ra 2 file: một file có tên là **Makefile**, file còn lại có tên là **Kbuild**. Đầu tiên, ta sẽ tạo ra **Makefile**.

```
#cd /home/ubuntu/ldd/phan_1/bai_1_3
#vim Makefile

KDIR = /lib/modules/$(uname -r)/build

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

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

Trong Makefile trên:

- Thẻ **all** chứa câu lệnh để biên dịch các module trong thư mục hiện tại.
- Thẻ **clean** chứa lệnh xóa tất cả các object file có trong thư mục hiện tại.

Tiếp theo, ta tạo ra file Kbuild nằm trong cùng thư mục với Makefile:

```
#cd /home/ubuntu/ldd/phan_1/bai_1_3
#vim Kbuild

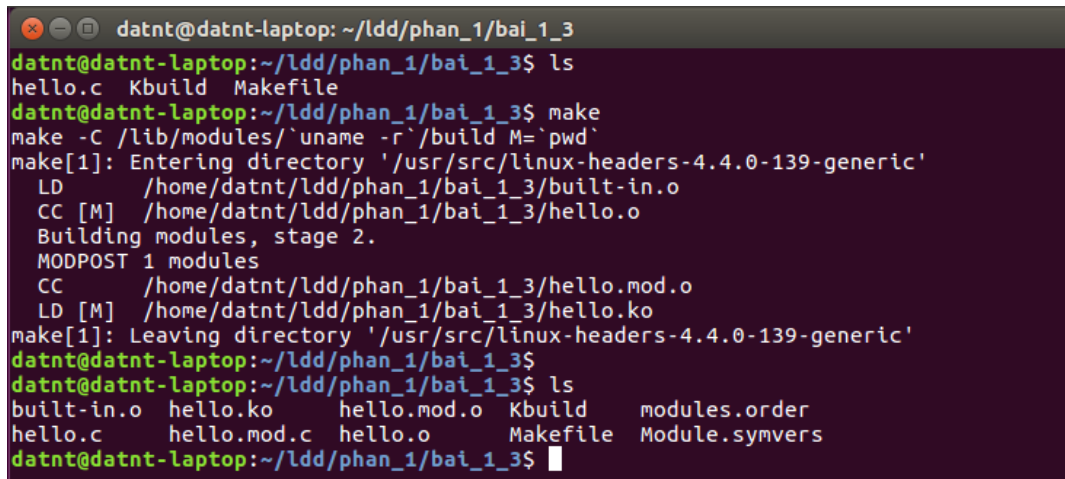
EXTRA_CFLAGS = -Wall

obj-m      = hello.o
```

Trong file Kbuild trên:

- Biến **obj-m** chỉ ra rằng: object file sẽ được biên dịch theo kiểu kernel module.
- Cờ **-Wall** cho phép trình biên dịch hiển thị tất cả các bản tin cảnh báo trong quá trình biên dịch.

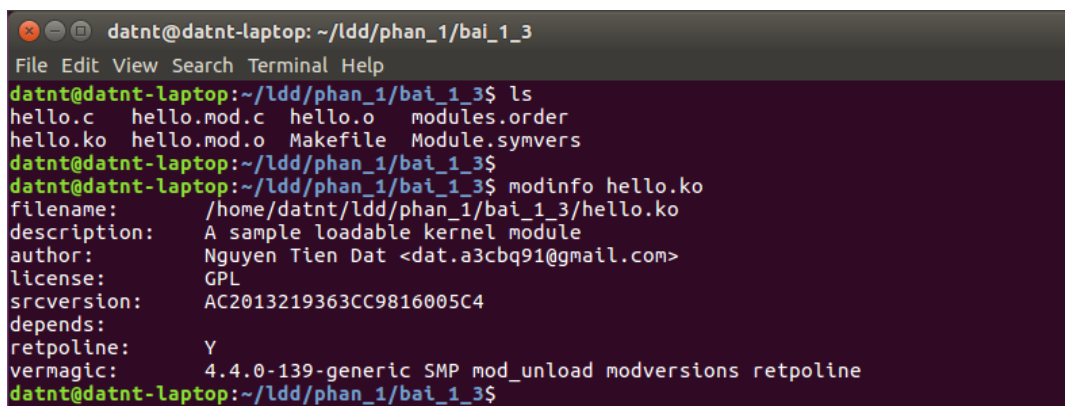
Để tạo ra kernel module, ta gõ lệnh **make** hoặc **make all** (hình 2). Khi ta gõ lệnh "make", tiến trình **make** sẽ dựa vào Makefile và Kbuild để biên dịch mã nguồn, tạo ra kernel module.



```
datnt@datnt-laptop: ~/ldd/phan_1/bai_1_3
datnt@datnt-laptop:~/ldd/phan_1/bai_1_3$ ls
hello.c  Kbuild  Makefile
datnt@datnt-laptop:~/ldd/phan_1/bai_1_3$ make
make -C /lib/modules/`uname -r`/build M=`pwd`
make[1]: Entering directory '/usr/src/linux-headers-4.4.0-139-generic'
LD      /home/dant/ldd/phan_1/bai_1_3/built-in.o
CC [M]  /home/dant/ldd/phan_1/bai_1_3/hello.o
Building modules, stage 2.
MODPOST 1 modules
CC      /home/dant/ldd/phan_1/bai_1_3/hello.mod.o
LD [M]  /home/dant/ldd/phan_1/bai_1_3/hello.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.4.0-139-generic'
datnt@datnt-laptop:~/ldd/phan_1/bai_1_3$ ls
built-in.o  hello.ko  hello.mod.o  Kbuild  modules.order
hello.c     hello.mod.c  hello.o  Makefile  Module.symvers
datnt@datnt-laptop:~/ldd/phan_1/bai_1_3$
```

Hình 2. sử dụng công cụ make để biên dịch kernel module

Sau khi biên dịch xong, ta sẽ thấy xuất hiện một file có tên mở rộng là **ko** (ko là viết tắt của kernel object). Đây chính là kernel module. Để biết được các thông tin về module, ta sử dụng lệnh **modinfo** (hình 3).



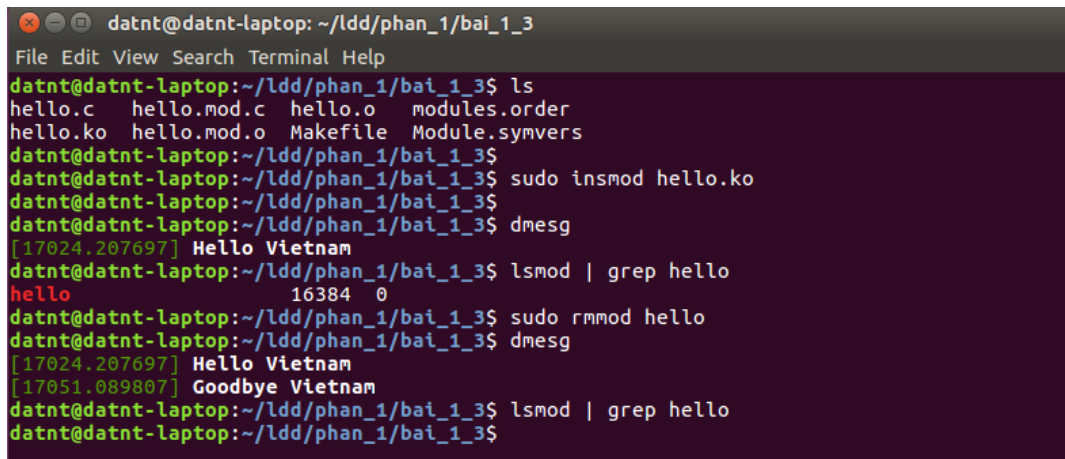
```
datnt@datnt-laptop: ~/ldd/phan_1/bai_1_3
File Edit View Search Terminal Help
datnt@datnt-laptop:~/ldd/phan_1/bai_1_3$ ls
hello.c  hello.mod.c  hello.o  modules.order
hello.ko  hello.mod.o  Makefile  Module.symvers
datnt@datnt-laptop:~/ldd/phan_1/bai_1_3$ modinfo hello.ko
filename:       /home/dant/ldd/phan_1/bai_1_3/hello.ko
description:    A sample loadable kernel module
author:        Nguyen Tien Dat <dat.a3cbq91@gmail.com>
license:       GPL
srcversion:     AC2013219363CC9816005C4
depends:
retpoline:     Y
vermagic:      4.4.0-139-generic SMP mod_unload modversions retpoline
datnt@datnt-laptop:~/ldd/phan_1/bai_1_3$
```

Hình 3. sử dụng công cụ modinfo để biết thông tin về module

Lắp/tháo kernel module

Để lắp module vào trong kernel, ta có thể thực hiện thủ công bằng cách gõ lệnh **insmod**. Sau khi lắp xong, ta sẽ dùng lệnh **lsmod** để kiểm tra xem module đã được load thành công chưa. Tiếp theo, ta sẽ dùng lệnh **hdmesg**

để theo dõi quá trình hoạt động của module (hình 4). Cuối cùng, chúng ta sẽ dùng lệnh **rmmod** để tháo module ra khỏi kernel.



```
datnt@datnt-laptop: ~/ltd/phan_1/bai_1_3
File Edit View Search Terminal Help
datnt@datnt-laptop:~/ltd/phan_1/bai_1_3$ ls
hello.c  hello.mod.c  hello.o  modules.order
hello.ko  hello.mod.o  Makefile  Module.symvers
datnt@datnt-laptop:~/ltd/phan_1/bai_1_3$
datnt@datnt-laptop:~/ltd/phan_1/bai_1_3$ sudo insmod hello.ko
datnt@datnt-laptop:~/ltd/phan_1/bai_1_3$ dmesg
[17024.207697] Hello Vietnam
datnt@datnt-laptop:~/ltd/phan_1/bai_1_3$ lsmod | grep hello
hello                16384  0
datnt@datnt-laptop:~/ltd/phan_1/bai_1_3$ sudo rmmod hello
datnt@datnt-laptop:~/ltd/phan_1/bai_1_3$ dmesg
[17024.207697] Hello Vietnam
[17051.089807] Goodbye Vietnam
datnt@datnt-laptop:~/ltd/phan_1/bai_1_3$ lsmod | grep hello
datnt@datnt-laptop:~/ltd/phan_1/bai_1_3$
```

Hình 4. Đưa module vào/ra khỏi kernel

Kết luận

Driver có thể được tích hợp luôn vào trong kernel hoặc được thiết kế dưới dạng module tách rời. Driver cho các thiết bị cố định, ví dụ các thiết bị trong smartphone, sẽ được tích hợp luôn vào trong kernel. Driver cho các thiết bị hay phải thay đổi, sẽ được thiết kế dưới dạng loadable module. Thông thường, các bus driver sẽ được tích hợp vào trong kernel, còn các device driver được thiết kế dưới dạng loadable module. Ngoài device driver, thì system call và file system cũng được thiết kế theo kiểu loadable module.

Kernel module có thể được đưa vào trong kernel một cách tự động, hoặc thủ công.

- Đối với trường hợp tự động, kernel kích hoạt tiến trình modprobe thông qua kmod hoặc udevd. Sau đó, modprobe sẽ đưa module cần thiết vào.
- Đối với trường hợp thủ công, ta sẽ sử dụng lệnh insmod hoặc modprobe. Còn để đưa kernel module ra khỏi kernel space, ta sẽ sử dụng lệnh rmmod.

Linux kernel cũng giống như một thư viện giúp lập trình viên xây dựng các kernel module. Khi viết một kernel module, ta cần phải tham chiếu tới các file trong thư mục include/linux. Bất cứ một module nào cũng cần tham chiếu tới file <linux/module.h>. File này chứa 2 macro quan trọng, đó là module_init và module_exit. Hai macro này giúp xác định đâu là hàm khởi tạo module, đâu là hàm kết thúc module. Thông thường, ta nên đặt các macro __init trước hàm khởi tạo, và macro __exit trước hàm kết thúc để tiết kiệm bộ nhớ.