# **Linux Kernel 設備驅動**

Reference :

Billibili 一口Linux

<https://beta.hackfoldr.org/linux>

<http://wiki.csie.ncku.edu.tw/linux/schedule>

<https://hackmd.io/@sysprog/gnu-linux-dev/https%3A%2F%2Fhackmd.io%2Fs%2Fr1Psrf0KW>

<https://ithelp.ithome.com.tw/users/20001007/ironman/958>

<https://ithelp.ithome.com.tw/users/20133148/ironman/4785>

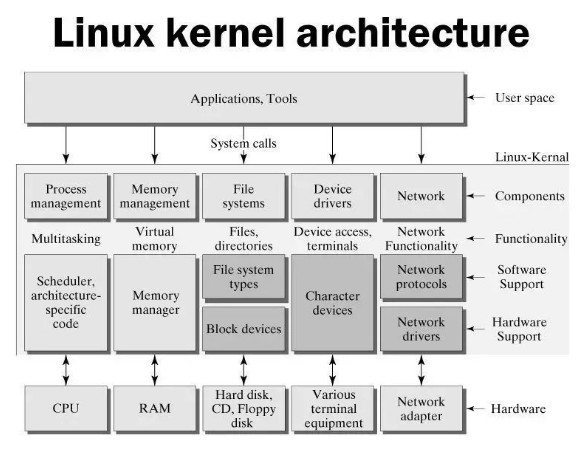
<https://ithelp.ithome.com.tw/users/20129818/ironman/3852>

<https://man7.org/training/lusp/index.html>

## Linux Kernel架構

Kernel module 是不同於一般在使用者模式下的執行程式 ( process)，它會載入到 Linux kernel 來執行且通常會是驅動或硬體相關的程式。而它的目的是去減小kernel，來讓Linux更有彈性而且能依據需求載入必要的模組。

架構上它區分成五大子系統 : 行程管理、內存管理、文件系統、設備管理、網路協議



行程管理

程式(process)是Linux中重要的概念，其中包含了選擇下一個要執行的程式，並且在切換運行程式的時候要進行上下文切換(context switch)，更甚至要在多個程式同時運行的狀況下，能夠正常運行。在後續的內容也會介紹基本常見的排程方法，以及O(1)排程、CFS算法，到現今的 EAS 排成器(Energy Awareness Scheduler)。

記憶體管理

記憶體管理在 Linux中較為複雜，其中牽涉到實體記憶體與虛擬記憶體，虛擬記憶體讓多個程式可以同時運行，讓程式實際的大小能比記憶體的大小更大。還有頁面回收(page reclamation) 、分頁錯誤(page fault)、 記憶體共享(memory share)等等議題。

中斷管理

中斷管理中包含例外(exception)，與中斷(interrupt)，例外包含程式錯誤例外、軟體產生例外、機械錯誤例外。中斷則通常是外部設備透過中斷信號告知處理器，處理器會停下目前執行的指令處理中斷。而這在Linux中提供了 softirq, tasklet 等等作法。

檔案系統

Linux 核心為了統合各個檔案系統實作，引入一層虛擬檔案系統 ((Virtual File System, VFS)，後者是一組檔案操作的抽象介面，於是依循 VFS 界面開發的任何的檔案系統，可在執行時期掛載到 Linux 核心，在 FUSE (Filesystem in Userspace) 出現後，更給檔案系統開發者極大的彈性，能夠快速且多樣地延展作業系統的特徵

以上節錄自 jserv老師 [Linux 核心設計: 檔案系統概念及實作手法](https://ithelp.ithome.com.tw/QVjOIv-WRzSC6pxc-jh3QQ)。

## 2. 基本內容

(1) C、Linux系統編程 (文件IO)

(2) 硬體電路圖(訊號線、數據線、SPI/I2C/UART)

(3) 瞭解program、process、thread

### **程式(program)**

它是能夠完成特定任務的指令集合 (可執行文件)，然後是放在硬碟尚未讀入記憶體的狀態。

### **行程(process)**

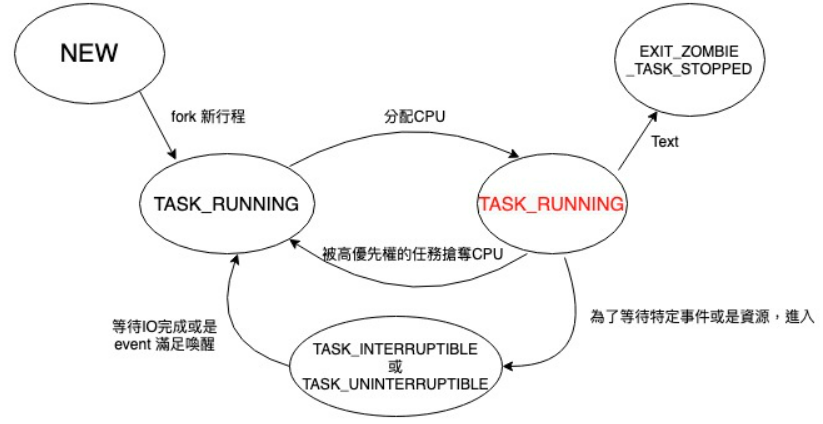
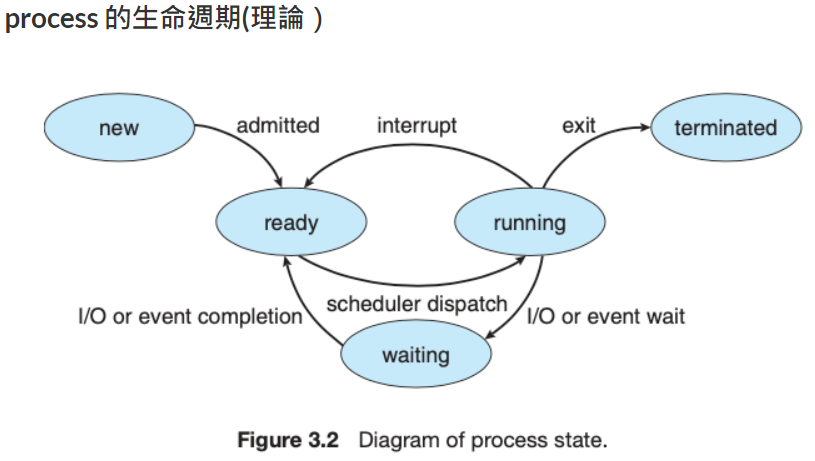
它是一段執行中的程序，內容會被移入記憶體，而在一個行程裡面會有可執行的指令。

另外在作業系統中，會用行程控制塊(process control block, PCB)，記錄行程所需要的資訊，常見的有以下幾種

1. 行程目前的狀態(state)：像是Running、Ready、Zombie state等
2. 程序計數器（Program Counter, PC):紀錄目前行程執行到哪條指令。
3. CPU暫存器(CPU registers)：在上下文切換時保存CPU當下的寄存器狀態，以便切換回來的時候能夠繼續執行。
4. CPU排程資訊(CPU scheduling info)：包含行程的優先權等級、排程的序列等等。
5. 記憶體管理(memory management info):行程使用的記憶體資訊，像是使用的分頁表(page table)
6. 文件資訊：行程打開的文件等等。

#### process 的行程狀態

<理論> <實際>



在作業系統的聖經上，理論的行程是下面的各種狀態

new: 創立新的新行程  
 ready: 行程所需的資料已經移入記憶體，等待獲得CPU資源  
 waiting: 行程因為等待某個資源暫時移出CPU  
 running: 行程正在CPU運行中  
 terminated: 行程結束

但在Linux實際情況又不同

**task\_running:**

Linux將理論的ready與running合成為單獨的task\_running，上圖中右邊的task\_running代表實際運行的行程，左邊則是正在等待被指定的CPU行程，只是對Linux來說這兩種狀態並沒有特別區分，而是統一用task\_running標記。

**task\_interruptible / task\_uninterruptible:**

當行程需要執行IO或需要等待條件滿足時，這時的等待時間是無法預估的，因此行程的狀態會轉移成task\_interruptible / task\_uninterruptible，而這兩個狀態差別是， task\_interruptible在條件滿足/收到未被屏蔽的訊號時能夠返回等待隊列中，反之task\_uninterruptible並不能被訊號喚醒，只能夠在條件滿足時返回運行狀態。

\_TASK\_STOPPED: SIGSTOP、SIGTSTP、SIGTTIN、SIGTTOUT等信號會將行程暫時停止，進入\_TASK\_STOPPER的狀態。  
**exit\_zombie:**

行程已經不在運行隊列中，只是記憶體仍然保有該行程的資料，尚未處理。

以上的定義出現在 /include/linux/sched.h 中

### **執行緒(thread)**

作業系統分配記憶體、CPU時間等等資源的基本單元。但是Linux 裡面並沒有區分行程與執行緒，所以當使用 task\_struct的時候可以代表process 或 thread。  
從實作的角度看來  
建立user process的方法是有三個系統呼叫: fork, vfork, clone  
建立kernel thread的方法有兩種系統呼叫: kernel\_thread, kthread\_create

fork、vfork、clone、kernel\_thread

一張含有 文字 的圖片

自動產生的描述 一張含有 文字 的圖片

自動產生的描述

一張含有 文字 的圖片

自動產生的描述 一張含有 文字 的圖片

自動產生的描述

## 3. 環境安裝

透過VMware安裝Ubuntu

sudo apt-get autoremove open-vm-tools (不一定要做)

(1) 安裝 Open VM Tools Desktop：$sudo apt install open-vm-tools-desktop

(2) 安裝 Open VM Tools Server：$sudo apt install open-vm-tools

(3) 重新開機: $Reboot

(4) 驗證安裝: $apt-cache policy open-vm-tools\*

可直接在VMware、Windows拖動檔案來傳輸 🡨 但需要先修改資料夾權限

chmod o+w [foldername\]

* < Ctrl + Alt + t 打開命令終端>
* 必備工具: Source Insight

### Linux 開機流程

#### Step 1 硬體設備引導

1.1 POST (Power On Self-Test) 上電檢查

1.2 BIOS 🡪 1.初始化設備、2.尋找開機位置 (通常是用硬碟來開機)

HDD : 查找啟動硬碟的第一個扇區(MBR/BootSector）

1.3 MBR 🡪 1.3.1 Bootloader（啟動裝載程序）GRUB、分區表、開機管理程式(LILO)

#### Step 2 BootLoader啟動

2.1 執行BootLoader主程序(位於MBR前446個bytes)

2.2 啟動GRUB的核心畫面並解析grub.conf 配置文件，加載Kernel到memory中

#### Step 3 Kernel引導階段

3.1 /boot/kernel及內核參數 內核初始化，加載基本的硬件驅動

3.2 /boot/initrd 引導initrd解壓加載

3.2.1 階段一：在內存中釋放供內核使用的根文件系統

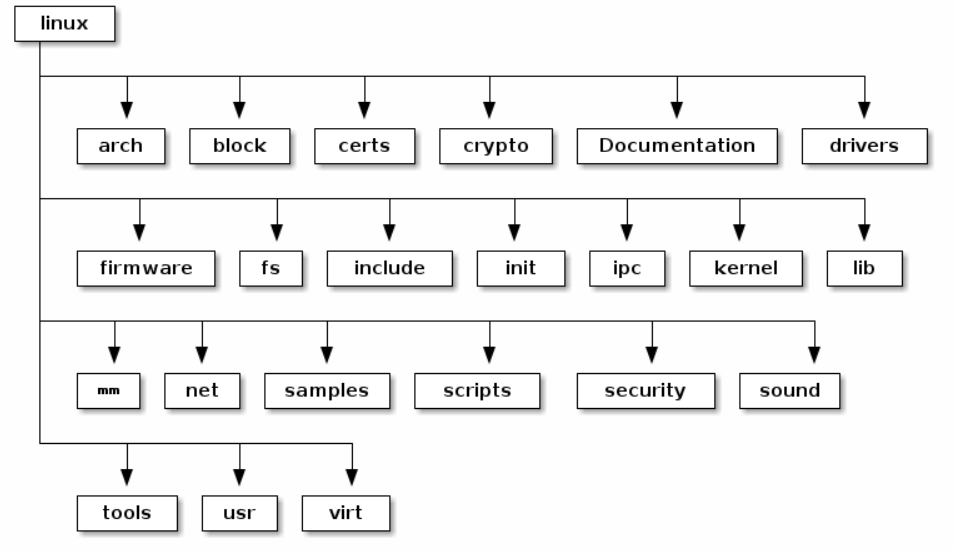
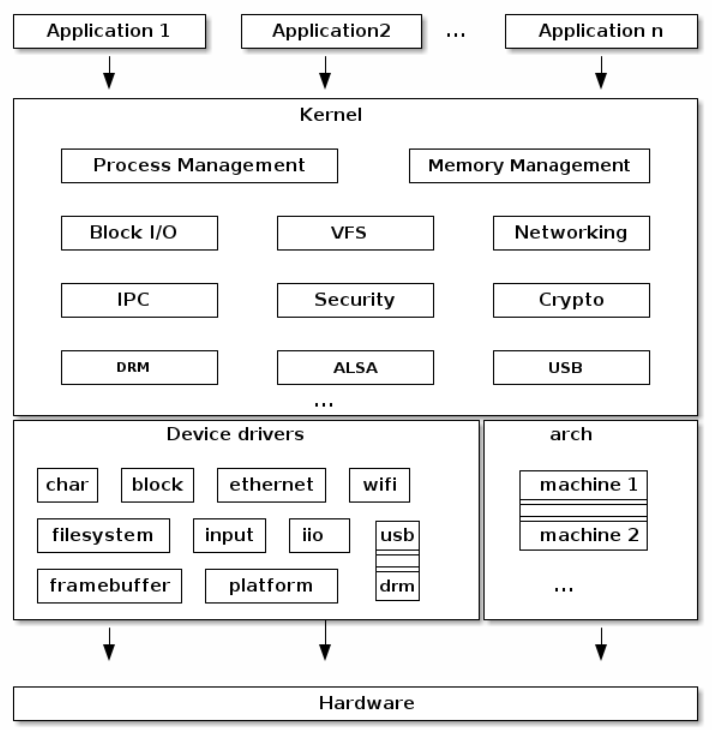
執行initrd文件系統中的init，完成加載其他驅動模塊

3.2.2 階段二：真正的根文件系統中的 /sbin/init 進程執行中

1. 載入kernel
2. kernel執行第一支程式 /sbin/init
3. init 執行第一支 script程式/etc/rc.d/rc.sysinit
4. init 執行預設的 run-level 目錄中的 scripts
5. init 執行 /etc/rc.d/rc.local
6. 執行 /bin/login 程式
7. 執行shell程式(Command Interpreter)

## 4. Linux目錄結構

### Linux source code layout

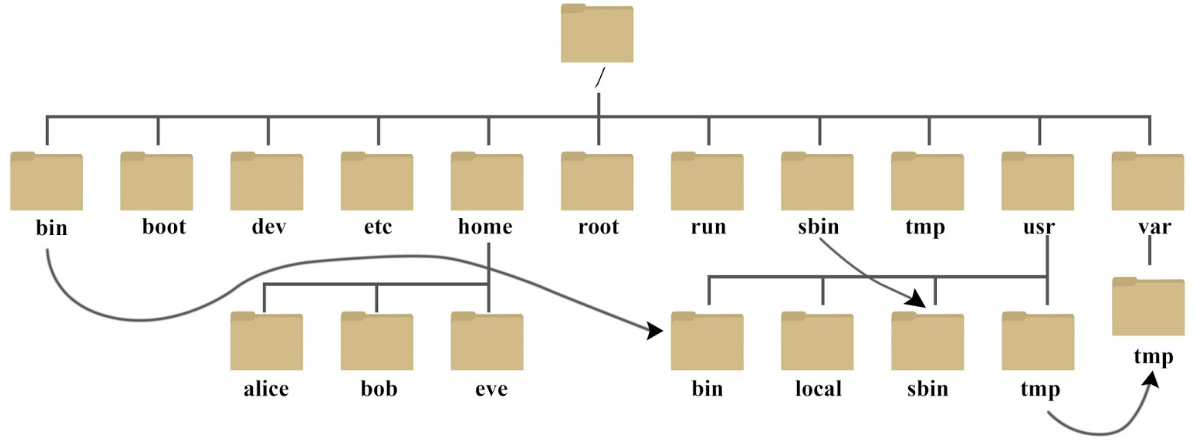
平台無關Init、kernel、mm、ipc、lib、fs、net、drivers

平台相關arch 🡪 i386、arm、ppc、sh (每個又包含kernel、mm、lib)

<https://linux-kernel-labs.github.io/refs/heads/master/lectures/intro.html>

* arch - contains architecture specific code; each architecture is implemented in a specific sub-folder (e.g. arm, arm64, x86不同CPU架構有不同目錄)
* block - contains the block subsystem code that deals with reading and writing data from block devices: creating block I/O requests, scheduling them (there are several I/O schedulers available), merging requests, and passing them down through the I/O stack to the block device drivers
* certs - implements support for signature checking using certificates
* crypto - software implementation of various cryptography algorithms as well as a framework that allows offloading such algorithms in hardware
* Documentation - documentation for various subsystems, Linux kernel command line options, description for sysfs files and format, device tree bindings (supported device tree nodes and format)
* drivers - driver for various devices as well as the Linux driver model implementation (an abstraction that describes drivers, devices buses and the way they are connected)
* firmware - binary or hex firmware files that are used by various device drivers
* fs - home of the Virtual Filesystem Switch (generic filesystem code) and of various filesystem drivers
* include - header files
* init - the generic (as opposed to architecture specific) initialization code that runs during boot
* ipc - implementation for various Inter Process Communication system calls such as message queue, semaphores, shared memory
* kernel - process management code (including support for kernel thread, workqueues), scheduler, tracing, time management, generic irq code, locking
* lib - various generic functions such as sorting, checksums, compression and decompression, bitmap manipulation, etc.
* mm - memory management code, for both physical and virtual memory, including the page, SL\*B and CMA allocators, swapping, virtual memory mapping, process address space manipulation, etc.
* net - implementation for various network stacks including IPv4 and IPv6; BSD socket implementation, routing, filtering, packet scheduling, bridging, etc.
* samples - various driver samples
* scripts - parts the build system, scripts used for building modules, kconfig the Linux kernel configurator, as well as various other scripts (e.g. checkpatch.pl that checks if a patch is conform with the Linux kernel coding style)
* security - home of the Linux Security Module framework that allows extending the default (Unix) security model as well as implementation for multiple such extensions such as SELinux, smack, apparmor, tomoyo, etc.
* sound - home of ALSA (Advanced Linux Sound System) as well as the old Linux sound framework (OSS)
* tools - various user space tools for testing or interacting with Linux kernel subsystems
* usr - support for embedding an initrd file in the kernel image
* virt - home of the KVM (Kernel Virtual Machine) hypervisor

### Linux 系統目錄



**/bin :** 存放最常用的命令。(cp、ls)

**/boot：**存放啟動Linux時的核心文件，包括作業系統的核心以及導引程序Grub。

**/dev ：**存放Linux的外部設備，在Linux中訪問設備和訪問文件的方式是相同。

**/etc：**存放所有系統管理需要的配置文件和子目錄。

**/home**：主目錄，在Linux中每個用戶都有一個自己的目錄，一般該目錄名是以用戶帳號命名的，如上圖中的alice、bob 和eve。

**/lib**：存放系統最基本的動態連結資料庫和Kernel module，其作用類似於Windows裡的.dll。幾乎所有的應用程序都需要用到這些共享庫。

**/lib64 :** 64位元有這個資料夾是64位元的程式庫

**/lost+found**：這個目錄一般情況下是空的，當系統非法關機後，這裡就會存放一些文件。

**/media**：Linux會自動識別一些設備，然後再把識別的設備掛載到這個目錄下。

**/mnt**：臨時掛載其他文件系統。

**/opt**：opt 是optional，這是給主機額外安裝軟體存放的目錄。比如你安裝ORACLE數據庫就可以放到這個目錄下。但默認下是空的。

**/proc**：proc是Processes，它是一種虛擬文件系統，儲存的是當前Kernel運行狀態的一系列特殊文件，這個目錄是一個虛擬目錄，它是記憶體的映射，我們可以通過直接訪問這個目錄來獲取系統信息。如/proc/cpuinfo保存了有關CPU的訊息。  
另外這個目錄內容不在硬碟上而是在記憶體裡，所以透過修改裡面的文件，可以達到一些功能

像是下面命令可以屏蔽主機的ping命令，讓別人無法ping到機器：

echo 1 > /proc/ sys / net / ipv4 / icmp\_echo\_ignore\_all

**/root**：該目錄為系統管理員，也稱作超級權限者的用戶主目錄。

**/sbin**：Superuser Binaries，這裡存放的是系統管理員使用的系統管理程序。

**/selinux**： 這個目錄是Redhat/CentOS 所特有的目錄，Selinux 是一個安全機制類似於windows 的防火牆，但是這套機制比較複雜，這個目錄就是存放selinux相關的文件的。

**/srv**： 該目錄存放一些服務啟動之後需要提取的數據。

**/sys**：這是Linux2.6的大的變化。該目錄下安裝了2.6中新出現的文件系統 sysfs 。sysfs文件系統包含下面3種文件系統的信息：

進程訊息的proc 文件系統、設備的devfs 文件系統、偽終端的devpts 文件系統

該文件系統是內核設備樹的一個直觀反映。當一個內核對像被創建的時候，對應的文件和目錄也在內核對象子系統中被創建。

**/tmp**：tmp 是temporary(臨時) 的縮寫這個目錄是用來存放一些臨時文件的。

**/usr**： usr 是unix shared resources(共享資源) 的縮寫，這是一個非常重要的目錄，用戶的很多應用程序和文件都放在這個目錄下，類似於windows 下的program files 目錄。

**/usr/bin：**系統用戶使用的應用程序。

**/usr/sbin：**超級用戶使用的比較高級的管理程序和系統守護程序。

**/usr/include :** 存放C/C++頭文件的目錄

**/usr/src：**Kernel source code默認的放置目錄。

**/var**：variable存放著不斷擴充的東西，一些經常被修改的目錄會放在這個目錄下。包括各種日誌文件。

**/run**：臨時文件系統，儲存系統啟動以來的訊息。當系統重啟時這個目錄下的文件會被刪掉或清除。如果你的系統上有/var/run 目錄，應該讓它指向run。

## 5. Kernel Module

對於放在Kernel執行的程序會和一般的程序明顯的不同，這點可以從範例程式碼hello.c看到。

而主要造成不同的原因是Kernel是透過上電後由bootloader去啟動的，它會是常駐執行的。而一般的程序是放在user space，這和kernel space兩個的地址是不同的。

* user space的地址通常是虛擬的地址
* Kernel module的編譯需要創建makefile然後透過make的方式編譯，而不是gcc

在做Kernel編譯時需要兩份檔案: hello.c、Makefile。

一張含有 文字 的圖片

自動產生的描述

一張含有 文字 的圖片

自動產生的描述

那這裡的順序是Makefile會在內部執行三次

1. 進到else執行 🡪 make -C $(KDIR) M=$(PWD) modules

2. 進到ifneq執行 🡪 obj-m:=hello.o (編譯產生目標文件)

3. 進到ifneq執行 🡪 將object file和kernel中會用到的module合併產生出hello.ko

ifneq ($(KERNELRELEASE),)

如果$(KERNELRELEASE) != 空 (因為還沒編譯空對空是相同的，所以會能進到else; 不相同則進到hello.o)

else 底下有兩個還境變數

(1) KDIR:=/lib/modules/$(shell uname -r)/build [指定到生成目標文件所依賴的內核]

(2) PWD :=$(shell pwd) 儲存當前目錄到PWD

* make all (默認)🡪 執行make -C $(KDIR) M=$(PWD) modules
* make clean 🡪 執行rm -f \*.ko \*.o \*.mod.o \*.symvers \*.cmd \*.mod.c \*.order
* obj-m 表示要編譯成模組的檔案名稱集合
* all 為預設的 make 動作
* clean 為清除的動作

另外要查看它跑的過程，可以加入顯示訊息去看$(info “顯示內容”)，然後結果可以在make後看到。

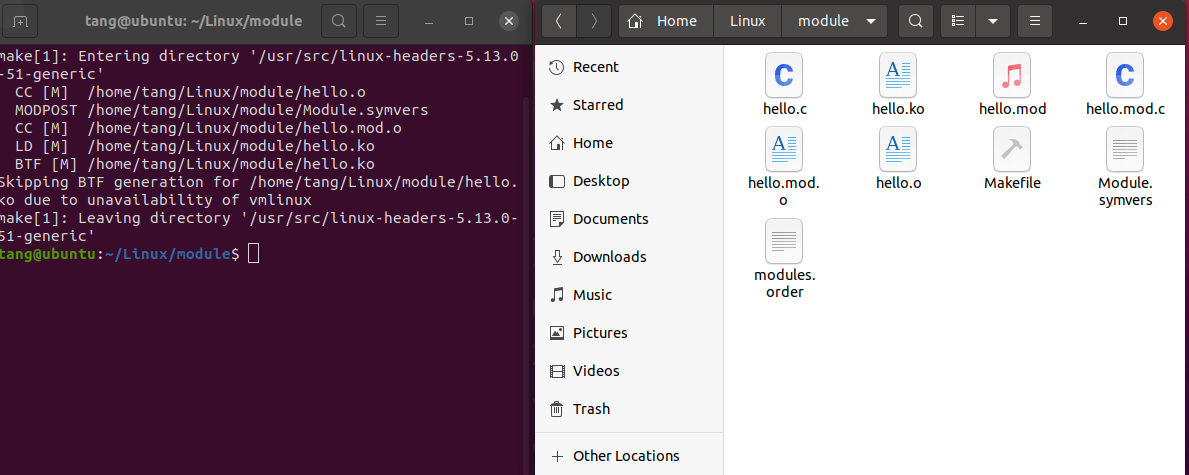
### Kernel Module編譯加載過程

常用指令

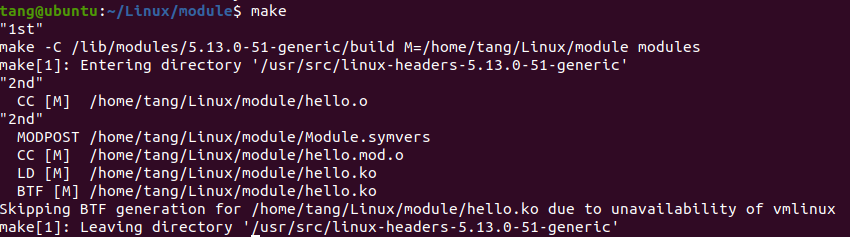
* make 編譯
* lsmod 顯示模塊
* insmod/rmmod 安裝卸載模塊
* dmesg 列出log訊息

#### 編譯檔案

要執行module編譯在使用上只要將Makefile和hello.c放到同一個資料夾再到terminal執行make就行了。那下圖中除了Makefile和hello.c其他都是編譯後產生的檔案。另外在做make的過程中CC [M]代表編譯、LD [M]代表鏈結，其他有編譯錯誤的情況會出現紅字，成功的話則會出現leaving directory代表編譯完成了



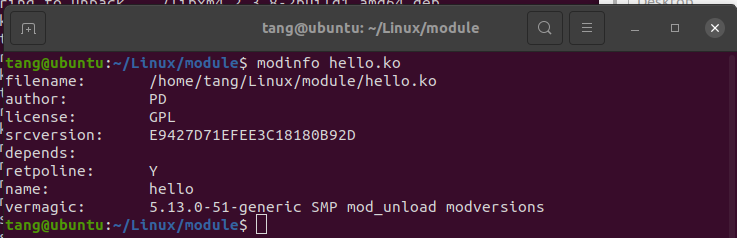
這邊是顯示剛才說明Makefile跑三次的情形



#### 加載模塊

要將module插入到kernel使用，會需要用到hello.ko這份剛編譯完的檔案。

可以先用modinfo hello.ko查看模塊資訊



接著依序輸入指令完成，清除紀錄，然後用dmesg查看模塊加載、卸載時的訊息。

另外lsmod可以用grep來搜索加載的模塊 🡪 lsmod | grep hello

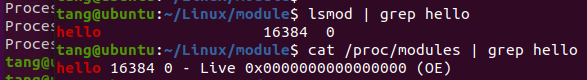
tang@ubuntu:~/Linux/module**$ dmesg –c**

tang@ubuntu:~/Linux/module**$ dmesg**

tang@ubuntu:~/Linux/module**$ sudo insmod hello.ko**

tang@ubuntu:~/Linux/module**$ lsmod**

tang@ubuntu:~/Linux/module**$ rmmod**



### Kernel Module參數傳遞

在Linux kernel中要藉由module\_param()的巨集才能在載入module的同時帶入參數 (linux/moduleparam.h)。根據定義module\_param(name, type, perm) 有三個參數，分別是變數名稱、變數的資料型態，以及對應sysfs檔案的權限。

例如: module\_param(myint, int, 0); 代表變數名稱是 myinit，其資料型別為 init，而權限設為 0 則代表沒有 sysfs 項目。而sysfs是Linux採用的一種虛擬檔案系統，其功能就是為了要傳遞kernel module的資訊。

這支程式hello-param.c是用來示範如何帶入字串到module中並顯示出來。而變數who的預設值是 "World"，所以如果沒有帶入參數時，預設會輸出 "Hello World"以及"Goodbye World"。

一張含有 文字 的圖片

自動產生的描述

Make 完之後 (記得修改之前的 Makefile)，透過 modinfo 可以看到這個 module 帶有一個參數，變數名稱是 who，變數型態是 charp (who:charp):

### Printk level

#define KERN\_EMERG    "<0>"    /\* system is unusable \*/  
#define KERN\_ALERT    "<1>"    /\* action must be taken immediately \*/  
#define KERN\_CRIT     "<2>"    /\* critical conditions \*/  
#define KERN\_ERR      "<3>"    /\* error conditions \*/  
#define KERN\_WARNING  "<4>"    /\* warning conditions \*/  
#define KERN\_NOTICE   "<5>"    /\* normal but significant \*/  
#define KERN\_INFO     "<6>"    /\* informational \*/  
#define KERN\_DEBUG    "<7>"    /\* debug-level messages \*/

**printk(KERN\_INFO "Hello, world!\n");**

https://huenlil.pixnet.net/blog/post/23271426

頭文件中linux/module.h是每個 kernel module 都需要的 ，

另一個linux/kernel.h是在 printk() 函數中定義 log level 巨集 (KERN\_INFO) 所需要的

## 6. Linux Kernel編譯

1. 下載kernel <https://www.kernel.org/>
2. 放到 Ubuntu後解壓縮
3. 運行以下指令進行kernel編譯  
   sudo make menuconfig  
   sudo apt install libncurses-dev  
   sudo apt install flex  
   sudo apt install bison build-essential  
   sudo apt install libssl-dev  
   sudo apt install libelf-dev

Make error about fatal error: libelf.h: No such file or directory

… 安裝完套件後  
 sudo make (因為會跑很久 所以用sudo make –j8)

<https://wiki.linuxquestions.org/wiki/How_to_build_and_install_your_own_Linux_kernel>

**<報錯1>**

**Make error about fatal error: libelf.h: No such file or directory**

https://github.com/buserror/simavr/issues/170

**<報錯2>**

**No rule to make target ‘debian/canonical-certs.pem‘, needed by ‘certs/x509\_certificate\_list‘**

修改.config文件

> vim .config

> CONFIG\_SYSTEM\_TRUSTED\_KEYS=“debian/canonical-certs.pem“ (改成””)  
> CONFIG\_SYSTEM\_REVOCATION\_KEYS=“debian/canonical-revoked-certs.pem” (改成””)

因為檔案很大，要用搜尋修改的方式

在mode選擇下輸入 /CONFIG\_SYSTEM\_TRUSTED\_KEYS (/+搜尋的關鍵字)，接著按Enter就能用insert的方式修改

https://blog.csdn.net/qq\_36393978/article/details/118157426

https://kaochenlong.com/2011/12/28/vim-tips/  
https://iter01.com/228931.html

**<報錯3>**

[vim 保存提示readonly option is set （add！to override）](https://blog.csdn.net/u010500895/article/details/52815553)

其實這是權限造成的，如何強制保存呢？ :wq!

## 7. Export Symbol

Linux 內核採用模塊化的方式管理內和代碼。不過內核中的模塊是相互獨立的，也就是A模塊的全域變量在B模塊式無法訪問的

使用全域符號表供其他模塊訪問

<全域符號表>

1. Ubuntu 🡪 /usr/src/linux-headers-xxxxx-generic-pae/Module.symvers

一張含有 文字 的圖片

自動產生的描述

1. 獨立編譯的內核則在根目錄底下

<Module.symvers>

一張含有 文字 的圖片

自動產生的描述