2006-11-30 (Rev.2)

陳俊宏 <jollen@jollen.org>

有效學習 Linux 驅動程式

目錄

前言

歡迎到 Jollen's Linux Device Driver 專欄線上學習：
http://www.jollen.org/LinuxDeviceDriver/

Linux 驅動程式是一門進入門檻較高的學科，在整個 Jollen 授課經驗中發現，其自學難度高的主要原因之一是缺乏有效的學習方法；主題本身雖然也有難度，但卻未必是學不通的主因。

技術性課程的教育訓練，是否能帶給同學優質的學習，教材內容是否包含一般化的材料（不能只是 case study），也是重點。

目前雖然有 O'reilly 的 Linux Device Drivers, 3e 聖經本，但對初學者而言，若沒有一個有效的學習方法，想要有效率地學會驅動程式，仍然是比較有難度的。Jollen 依據自己多年的教學經驗，設計了一套有效率的學習課程，Jollen 課程的目的是為了協助工程師快速掌握 Linux 驅動程式的核心觀念。本文的目的則是說明研讀 Linux 驅動程式的正確方法。

Jollen 的課程規劃

在討論如何有效自學前，先來介紹一下 Jollen 自己的驅動程式課程規劃！目前 Jollen 與自強工業科會基金會合作的 Linux 驅動程式系列共分為 3 門主題，課程設計如下：

- Linux Device Driver 入門
- Linux Device Driver 進階
- Embedded Linux / ARM9 人機介面技術

Linux Device Driver 入門

Linux 驅動程式的入門課程，學員能藉此此課程了解 Linux 驅動程式架構與基本的 Linux 驅動程式觀念。Linux 驅動程式的入門關鍵點為「觀念的解析」，有 了觀念後，便能快速看懂 Linux 的驅動程式，這也是許多 Linux 驅動程式初學者的障礙。本課程以觀念導向講解為主。

Linux Device Driver 進階

Linux 驅動程式的進階課程。本課程採前段＋後段的分界法開課（水平切法），學員可以分段學習。

Embedded Linux/ARM9 週邊與人機技術

在理解 Linux 驅動程式的架構與觀念後，需要實際對一個應用案例做探討，本課程的設計目的即是提供學員一個 Samsung S3C2410 平臺，並對此平臺做案例解說與實作。本課程是「Linux Device Driver 入門」與「Linux Device Driver 進階」的案例實作課程（Case Study）。

修習本課程後，學員將具備「在 Embedded Linux 平臺使用與設計 ARM9 週邊與人機介面」的能力。本課程將由「驅動程式架構面」切入，並講解如何設計應用程式來存取 ARM9 的週邊與人機介面；本課程為「講解」導向課程，由「驅動程式」切入「人機程式設計」。學員將能了解如何設計程式並透過驅動程式來存取 ARM9 介面的 coding 方法與整體架構。本課程使用 Jollen 自製的 Jollen-Kit! 2006 Training Board。

教材與課程設計

O'reilly, "Linux Device Driver, 3rd edition" - 學 Linux 驅動程式的主要教科書，全球公認的經典，學驅動程式怎能不用此書！

Jollen 自製講義 - 課程全程使用 O'reilly 的投影片。這是 Jollen 地毯式自 Linux Device Drivers,3e 一書所整理重點摘要，想要學習 Linux 驅動程式，這本書絕對是唯一教材。Jollen 所整理的投影片完整掌握此書的脈絡，別家使用的投影片，無法與此書配合，結果當然就是學員想要在家使用此書自習時，又要花上一段肮枉時間來了解此書的組織邏輯。

本講義 2 年多來，歷經數十次大調整小修改，找出了一個最佳的教學模式。這份講義是跟著 O'Reilly 的書所整理的，因此也可以做為初學者研讀此書的閱讀順序建議。

Jollen 的「Linux 驅動程式入門：架構、觀念與實作」一書 - 這是一本未出版的 Jollen 著作，此書定位為 O'reilly

一書的「前傳」，寫這本書的目的是為衝著教育訓練而來的。這本書可是由 Jollen 的 Linux 驅動程式學習心得所整理而成的。

Jollen 建議有志學習（來上課或自修）的朋友，必須找到對的教材與對的方法，才能有效學習 Linux 驅動程式。若以外觀（大架構）與內觀（詳細流程）來討論，正確的 Linux 驅動程式學習觀念應該是：

- 外觀：正確學習 Linux 驅動程式的方法。
- 內觀：以一個流程來初步認識 Linux 驅動程式。

在後文 Jollen 會就這二個層面來做說明。

給初次接觸 Linux Device Driver 的學員

首次接觸 Linux device driver 應該要如何著手？

驅動程式本身是屬於「軟體硬介面」的程式設計技術，要學好這門技術，正確的學習方法是相當重要的。正確的驅動程式學習方法有幾個重點：

- 使用適合自己的教材，重點是必須配合 Linux kernel 原始碼來學驅動程式的設計
- 確實去瞭解硬體的規格與微處理器架構
- 要能分得清楚哪些東西是介面（interfacing）也就是與硬體無關的程式（machine-independent）；以及哪些是站在第一線做硬體控制的程式（machine-dependent）。
- 相關名詞。
- 軟體硬介面。
- 使用 kernel 2.4 來入門

Linux kernel 2.4.x 才能編譯獨立的 kernel module，初學 Linux 驅動程式的讀者，應先準備一台 Linux kernel 2.4.x 的 Linux 工作電腦（推薦安裝 Red Hat Linux 9），以利學習。

錯誤的學習方法：

- 只看驅動程式的書，就想把驅動程式學好。
- 誤以為驅動程式與 kernel 原始碼八杆子打不著。
- 誤以為不用瞭解硬體與 CPU 架構就可以精通驅動程式。

一開始學驅動程式，務必瞭解以下這些東西：

- Linux kernel 的基本資料結構，請先從 *linux/fs/devices.c* 與 *linux/fs.h* 下手。
- 先知道什麼是 **file operations**，什麼是 **fops**。
- 瞭解*/dev* 的用途。

像典型的驅動程式書籍，要將裝置分成字元型（character devices）、區塊區（block devices）以及網路裝置（network）來分別討論驅動程式的寫法。

必懂：Linux Kernel Module

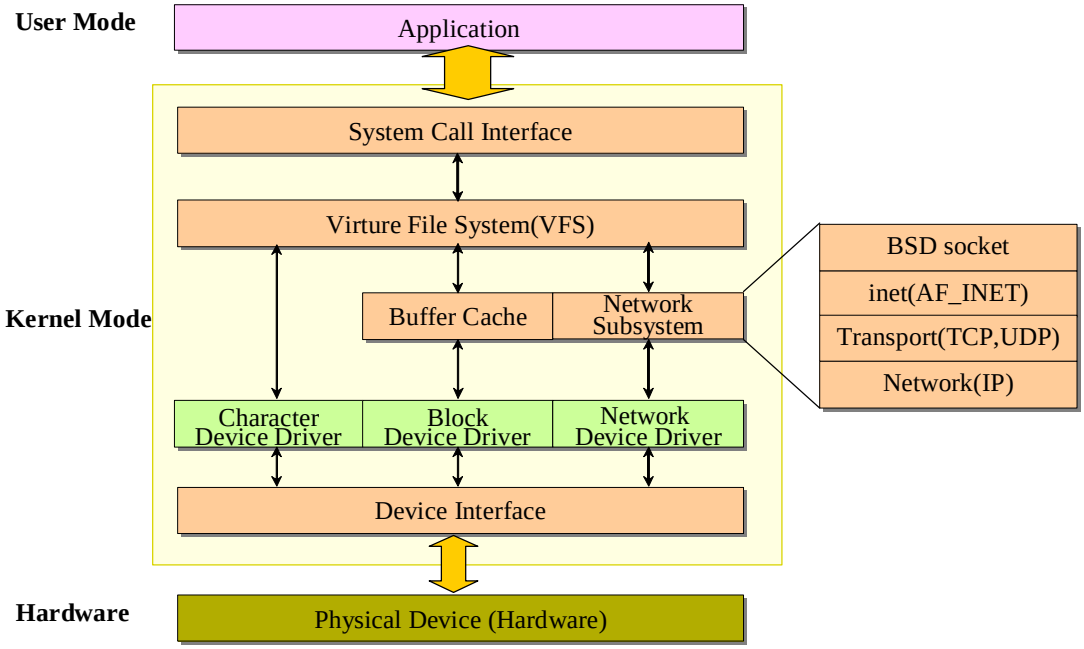
Linux kernel module 並不等於驅動程式，也就是並非所有的 kernel module 都是驅動程式，但我們大多會將驅動程式寫成 kernel module。所以，要學會寫驅動程式，就要先會寫 kernel module。

必看：linux/fs.h

在學習 Linux 驅動程式，必須先瞭解 *file_operations* 結構。故名思意，*file_operations* 結構是用來操作檔案的重要資料結構，透過此結構才能對檔做開啟、讀取等操作，而指向此結構的指標則稱為 **fops**。

除了 *file_operations* 外，對其它資料做操作也有相對應的操作結構，例如要操作 inode 就會有 *inode_operations*。這些重要的操作結構均定義於 *linux/fs.h* 標頭檔裡。此標頭檔應該要好好看一看。

大架構：



此圖不在 O'Reilly 的書裡，但若能徹底理解如圖，則有助於了解 Linux 驅動程式的整體架構。User application 與驅動程式之間的互動關係也能由此圖進行說明，可參照本文的「結語」部份。

正確的 Linux 驅動程式學習方法

Linux 驅動程式的重點在於「觀念」而非程式碼的語法法，Linux 驅動程式觀念的核心精神為如何設計機制良好的 kernel 程式。善用 Linux 的 APIs 來設計機制（mechanism）與行為（behavior）良好的驅動程式，是學習 Linux 驅動程式的重點。

對 device driver 設計師而言，我們所要做的工作可分成 2 個層面來討論：virtual device driver 與 physical device driver。

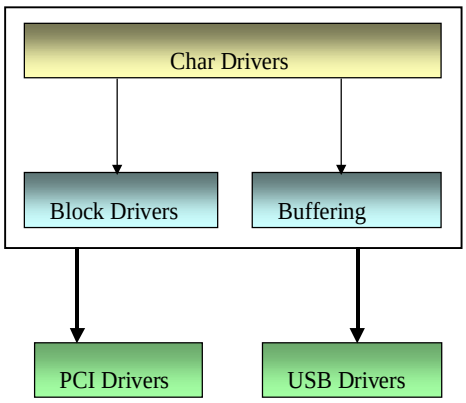
1. Virtual Device Driver: 往上層支援 Linux kernel 所提供的 Virtual File System 層，並藉此實作 system calls。使用者可透過 system call interface 與 device driver 溝通。

2. Physical Device Driver: 往下層使用 Linux kernel 所提供的 device interface 來存取並控制實體硬體裝置。

Virtual device driver 的主題重要性大於 physical device driver，如何善用 Linux 所提供的介面 (interface) 來設計驅動程式，並配合 user application 來設計應用程式是這個主題的重點。與 user application 如何互動，是撰寫驅動程式時所要考慮的重要一環，只考量驅動程式本身的設計，而忽略或輕忽 user application 的設計，是錯誤的觀念。

Virtual device driver 的目的在於善用 Linux 的 APIs 來設計機制 (mechanism) 與行為 (behavior) 良好的驅動程式，因此「觀念」的重要性遠大於「語法」的討論。本書秉持這樣的信念撰寫而成，仔細閱讀絕對是多多益善的。Physical device driver 則是討論「如何透過 I/O port 或 I/O memory」來控制裝置，也就是與晶片組的溝通。

Linux device driver 的撰寫必須多了解一些更低階或低層的硬體知識，才能融會貫通。站在軟體與韌體的角度而言，我們不必要太深入硬體的實作細節，但了解其原理對增進功力是絕對有幫助的。



學習 Linux device driver 應由 character device driver 起步，因為許多重要的入門觀念均可藉由 character device driver 學得。

OS 是設計良好的軟硬體介面，Linux 驅動程式設計即是在學習如何使用 Linux 提供的介面來設計驅動程式。

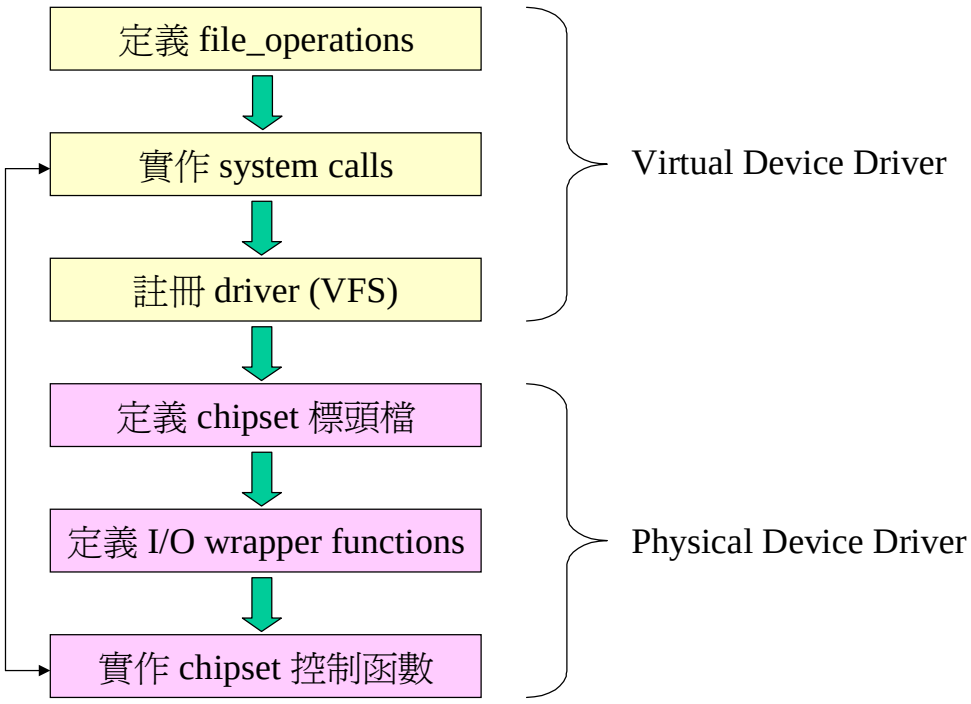
Linux kernel 所提供的 API 均經過良好設計，因此使用 kernel 所提供的 API 可以確保系統運行的安全性與穩定性。例如，有許多 API 內部都有 spinlock 來解決 race condition 的問題。

以一個觀念流程做為開場

要如何才能理解「如何設計 Linux 驅動程式」與「什麼是驅動程式」這二個初學者最主要的盲點呢？

利用一個流程來說明如何設計 Linux 驅動程式是最有效的教學方法。流程是一個系統化知識的結論，為了讓 Linux 驅動程式的初學者認識什麼是 Linux 驅動程式，並了解撰寫 Linux 驅動程式的方法，本課程一開始便以一個 Linux 驅動程式的觀念流程來做說明。此流程配合本講義特有的 Debug Card 驅動程式來進行實作課程。

Linux 驅動程式的設計雖然沒有一定的標準流程，但是由「觀念」層面可以歸納出一個一般化的流程。Jollen 的課程將介紹一個設計 Linux 驅動程式的基本流程，並以一個範例來說明實作過程。



此流程為一個觀念流程，實際撰寫驅動程式時，並不會完全以這個流程來設計。但初學 Linux 驅動程式時，則應以此流程為主循序學習，才能理解基本的 Linux 驅動程式實作。

對 Linux 驅動程式而言，virtual device driver 的重要性遠在 physical device driver 之上，乍聽之下這或許不太能理解，因為沒有 physical device driver 是無法真正驅動硬體的。但實作上，physical device driver 是一成不變的程式寫法，能不能寫出好的驅動程式，關鍵是在 virtual device driver 的部份。

Linux 驅動程式是建構在 file_operations 裡。file_operations 定義驅動程式的 system call 與實作 system call 的函數，我們把 file_operations 任何一個部份拿出來討論的話，都能

切成 virtual device driver 與 physical device driver 二個部份。

Linux 驅動程式採取嚴謹的分層式架構設計

Linux 驅動程式採用分層架構的觀念設計，透過「註冊」與「回呼」的機制來清楚地區分每一層的關係。初學者若無法掌握 Linux 驅動程式分層設計的關念，甚致無法接受分層架構的觀念的話，往往無法聽懂整門課，當然也就無法有效學會 Linux 驅動程式！

Linux kernel 這種分層架構的實作，必須利用以下 2 個步驟方能實現：

- 下層呼叫上層所提供的註冊函數，將自己註冊到上層。
- 上層驅動程式 callback 下層。

分層架構的實作必須在下層將自己註冊給上層，上層再回呼下層；上層的驅動程式必須提供註冊函數供下層呼叫，下層驅動程式所使用的註冊函數也將決定自己的上層架構。

最上層的驅動呼叫的註冊函數是由 Linux kernel 所提供的基本註冊函數，因此最上層的驅動程式是將自己註冊到 Linux kernel 裡，即 virtual filesystem 層。下層註冊到上層的實作只需要使用上層所提供的註冊函數即可，上層的驅動程式必須能處理下層的註冊動作。

研讀 Linux 驅動程式時，必須先將驅動程式的分層關係畫清楚，才能有效率地閱讀原始程式碼。Jollen 的自製中文講義以 Linux 的 video4linux 驅動程式做為實例，來清楚地講解最關鍵的 Linux 驅動程式分層的觀念。

O'Reilly, "Linux Device Drivers" 的讀法

如果您是 Linux 驅動程式的初學者，想要以這本書來自學驅動程式，又要在最短時間內掌握整個 Linux 驅動程式的精神，那麼有幾個方向提供您做參考：

- 不要依照章節順序來閱讀
- 不要地毯式的閱讀（即：不要依照章節與段落順序）

這本書的觀念解釋的非常清楚，但是最主要的 Linux 驅動程式觀念都在字元型驅動程式的章節裡，因此依照字元型驅動程式的主題來閱讀會比較適當。建議順序如下：

1. Virtual File System (VFS)
2. 'open' and 'release' system calls
3. 'open' and 'release' driver function
4. Wait Queues
5. Blocking and Nonblocking read/write
6. kmalloc(), copy_to_user(), copy_from_user()
7. I/O ports
8. ioctl()
9. 'ioctl' driver function
10. Reentrant codes (function)
11. Semaphore
12. Kernel Timer, Asynchronous Notification
13. Pre-Defined Task Queues
14. User-Defined Task Queues
15. Writing an Interrupt Handler
16. PCI Device Driver
17. Spinlocks
18. Bottom-half and Tasklets

此建議順序即是 Jollen 教育訓練課程所使用的投影片主題順序，自學的朋友，應該參考此建議順序，規劃一套屬於自己的閱讀計畫，如此便能加速學習 Linux 驅動程式。

開課

請上 Jollen 的網人網站查詢，建議您至 www.jollen.org 首頁加入 Jollen 在 Google Groups 裡的郵件論壇，以便我們能主動遞送最近的課程資訊給您。

結語

我們條列說明 Linux 驅動程式的核心觀念，以利讀者掌握學習方向：

1. Device file 是 user application 與 Linux 驅動程式溝通的媒介。
2. Device file 有一個不重覆的識別號碼，稱為 major number，並有自己的 minor number 用來表示自己的子裝置。
3. 不同的 Linux 驅動程式，均「支援」一個 device file；更精確的說法是，不同的 Linux 驅動程式，均「對應」一個「major number」。
4. Device file 即是 VFS 層。
5. Linux 驅動程式的核心為 file operations，此即「Linux 驅動程式物件」。
6. User application 透過 system call 與 device file 來與驅動程式溝通，user application 必須先呼叫 open system call 來「開啟驅動程式」。
7. 不同的 system call C 函數，會透過 Linux kernel 對應，並由 Linux kernel callback 驅動程式裡相對應的 system call 實作函數。
8. 驅動程式裡個別的 system call 實作函數，是由 file operations 所定義的。
9. File operations 即 *struct file_operations*，*struct file_operations* 定義 system call 的名稱與實作函數的對應，此對應關係使用「函數指標」來實作。