NetBSD 網路卡驅動程式：追蹤與分析

馮若華 曹世強 林盈達

國立交通大學資訊科學系

新竹市大學路 1001 號

TEL：(03)5712121 EXT. 56667

E-MAIL：{gis92510, weafon, ydlin}@cis.nctu.edu.tw

主要聯絡人：馮若華

摘要

一個電腦系統包含硬體與軟體部分，驅動程式(Driver)主要是負責硬體裝置與系統軟體間的溝通與控制；網路卡驅動程式即是負責作為網路卡與作業系統間溝通的橋樑。一方面，驅動程式接受作業系統的要求，透過 I/O port 控制網路卡上的暫存器和緩衝區，將封包傳送出去，另一方面，處理卡片所觸發之硬體中斷 (hardware interrupt)，以得知收到封包的消息，並利用直接記憶體存取(DMA，Directly Memory Access)機制搬進系統記憶體，觸發軟體中斷，通知作業系統，以減少處理器工作的負擔。在本文中，我們主要以 Intel 網路卡(使用 i82557 相容晶片)為追蹤對象，介紹其中作為傳送封包時用的 ifnet 結構和接收封包用到的 fxp_softc 結構。此外也將深入描述驅動程式使用 DMA 和 interrupt 來完成傳送接收封包的過程。

關鍵字：NetBSD、網路卡驅動程式、直接記憶體存取(DMA)、中斷(Interrupt)。

1.驅動程式簡介

作業系統主要的功能之一就是控制週邊的設備，而驅動程式就是扮演幫助作業系統控制週邊裝置的角色。我們藉由圖 1 描述驅動程式扮演軟體和硬體間溝通管道的概念。驅動程式從作業系統那邊收到的一般讀寫要求，轉換成週邊設備能了解的細部命令，依序加以執行。由於驅動程式的存在，作業系統想使用週邊裝置的時候，只需要對驅動程式作要求即可，不需要對於每個裝置的細節都加以了解，因此簡化系統設計的複雜度，並減輕運作時的負擔。若以網路卡驅動程式(Network Adaptor Driver)的例子來看，它要接收系統核心的要求，並透過網路卡上的 MAC 控制器(Controller)的暫存器作訊息溝通，達到兩個主要的任務：一是將要傳送的封包傳給網路卡，二是從網路卡抓取封包並將封包往上層送。

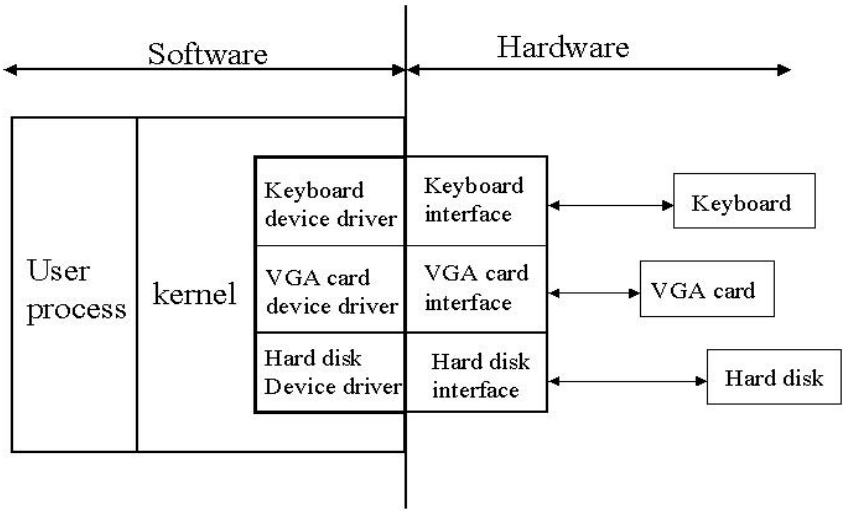

圖 1：驅動程式扮演作業系統與週邊設備溝通橋樑的概念圖

若以作業系統溝通方式來區分，在 Unix 中，驅動程式被區分成四大類：block、character、terminal、STREAMS[1]。但在 NetBSD[2]中，則只區分成兩類：區塊型(block)、字元型(character)。其中區塊型驅動程式的特性是經由許多固定長度的緩衝區(buffer)和作業系統進行資料交換，作業系統需要維護這些緩衝區，將資料裝入特定的資料結構中，使用緩衝區管理常式來包裝這些資料。見圖 2。

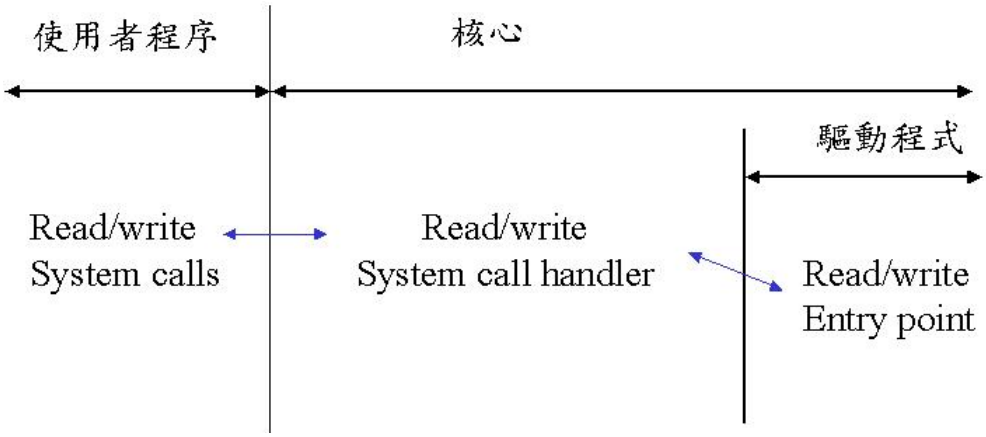

圖 2：區塊型驅動程式與核心及使用者程序溝通之示意圖

而字元型驅動程式則用來處理長度不固定的 I/O 要求。也就是說這種驅動程式可以用在一次處理一個 byte 的裝置(例如：line printer)，或著用來處理持續性的大量資料，（例如：analog-to-digital converter)。見圖 3。

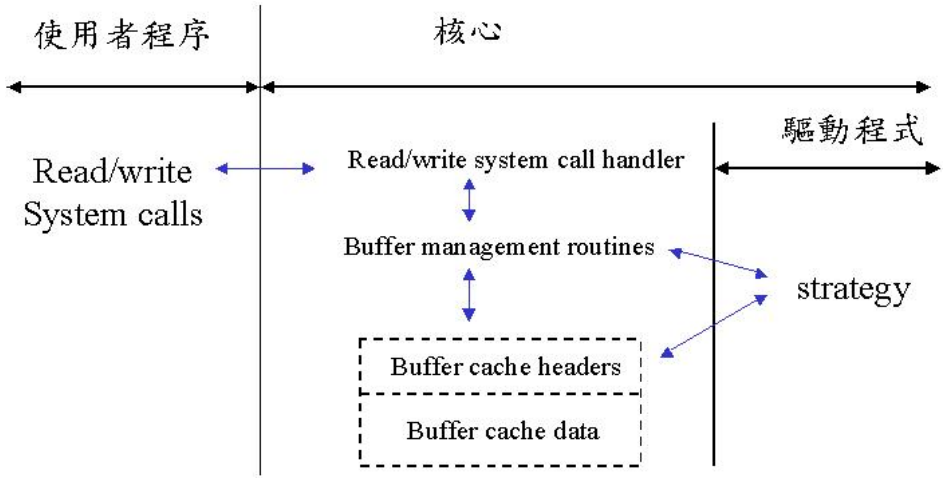

圖 3：字元型驅動程式與核心及使用者程序溝通之示

區塊型和字元型驅動程式最大的差別是，使用者程序和區塊型的互動是經由緩衝區來做資料交換，通常用在需要快速交換資料的設備上，例如儲存設備或網路設備。而和字元型驅動程式的互動則是直接將控制需求傳遞直接傳遞下來或是資料接收，也因此比較適用於低速的週邊設備。

在下一節，我們將介紹網路卡驅動程式撰寫時的五個基本作業，這包括掛上硬體、初始化、中斷處理，及透過 I/O port 讀取資料，然後是如何在 NetBSD 上設定加入新的裝置。第三節則追蹤相容於 Intel i82557 這個晶片的網路卡驅

動程式實例，先認識硬體與軟體的資源及工作分配和重要資料結構，以瞭解封包收送的細部動作。最後是我們的結論。

2.網路卡驅動程式之基本程序

撰寫網路卡驅動程式有五個基本程序。首先需撰寫，硬體掛上函式以及初始化函式，讓系統在初始的時候可以呼叫，使網路卡可以正常運作。接著必須撰寫中斷處理函式，以處理系統或裝置產生的中斷。此外，當驅動程式要接收和傳送封包時，必須利用 I/O port 來作控制裝置上的暫存器；最後在撰寫完驅動程式之後，必須修改系統設定檔，註冊這個裝置的存在。接下來，我們將分別詳述這五個程序。

2.1 掛上硬體 (attach)

掛上(attach)函式是在系統初始的時候，會被呼叫一次。驅動程式要和裝置溝通之前，需要先建立起溝通的管道，所以必須先作一些預備的工作。所謂準備的工作包含配置變數以便給稍後網路卡驅動程式運作時呼叫使用，或者是要求配置核心的記憶體建立和載入 DMA map，以便作為送收的封包暫存使用。系統提供底下幾個巨集來幫助驅動程式完成 DMA buffer 管理的功能。

int bus_dmamem_alloc()：要求 DMA 記憶體的控制資料

int bus_dmamem_map()：對應 DMA 記憶體的控制資料

int bus_dmamap_create()：產生控制資料的 DMA 對應

int bus_dmamap_load()：載入控制資料的 DMA 對應

掛上硬體一般來說是 NetBSD 驅動程式第一件工作。和 Linux 不同的是，NetBSD 中被沒有所謂的硬體探測(probe hardware)。所謂的硬體探測是指探測硬體所需使用的 I/O ports 和 IRQ 號碼。有了正確的 I/O port，才能和裝置上暫存器溝通，而有了 IRQ 號碼才能註冊中斷處理常式。但這些工作在 NetBSD 中，在進入網路卡驅動程式前，已經由作業系統準備就緒了。

2.2 初始化(initialize)

在掛上硬體、取得記憶體空間之後，隨之要做的就是要將介面初始化，這

包括記憶體空間、緩衝區結構及變數值的設定，而後才能啟動中斷，讓驅動程式能開始運作。在網路卡驅動程式中，會被初始的有：敘述封包傳輸的鏈結、收取封包的緩衝區、儲存收到封包的區域、群播過濾器並且啟動 config command/DMA。

2.3 中斷處理(interrupt handling)

系統核心與周邊裝置之間的資料交換通常會需要較多的時間，因此為了避免驅動程式需要不斷詢問工作是否完成，周邊裝置會利用觸發硬體中斷，來主動的呼叫驅動程式中的一個程序，來執行後續的工作，這便是所謂的中斷處理。而這個程序通常被稱為中斷服務常式。

在網路卡驅動程式利用中斷作資料交換時使用的機制稱為"interrupt-driven I/O"[3]，這個方法中輸出緩衝區(output buffer)被某個驅動程式填滿資料後， 在中斷處理時再將資料送給裝置；輸入緩衝區(input buffer)在中斷處理時會被填入資料，然後被需要這些資料的驅動程式所取出，而所謂中斷處理的部分就是交給中斷服務常式來負責。

2.3.1 註冊與註銷 ISR

一個裝置通常只需使用一個 ISR。而 ISR 通常由驅動程式提供，並在系統核心啟動時，利用核心提供的「ISR 註冊 API」來進行註冊工作。以 PCI 匯流排類型的裝置而言，NetBSD 提供給驅動程式使用之註冊和註銷 ISR 的 API 為 pci_intr_establish 及 pci_intr_distablish。這兩個函式的原始碼在 arch/i386/pci/pci_machdep.c 中，我們分別介紹如下：

ISR 註冊用 API：

```
void * pci_intr_establish(pc, ih, level, func, arg)
    pci_chipset_tag_t pc;
    pci_intr_handle_t ih;
    int level, (*func) __P((void *));
    void *arg;
```

其中的 func 就是要註冊的中斷處理常式，arg 為其參數，而 level 表示中斷的優先權，主要是以裝置的種類來區分優先權的高低。

ISR 註銷用 API：

```
void pci_intr_disestablish(pc, cookie)
    pci_chipset_tag_t pc;
    void *cookie;
```

這種做法和 Linux 不同，Linux 是在驅動程式中註冊和註銷中斷服務常式，跟裝置有關的運作都是在驅動程式中負責；但是 NetBSD 中註冊中斷服務常式是在核心啟動時一併進行，驅動程式中和裝置相關的硬體偵測方面是在核心執行。

2.3.2 撰寫 ISR 之注意事項

ISR 的工作通常包含中斷原因的判斷及後續工作的處理。以網路卡觸發之中斷為例，其處理常式要判斷該中斷究竟是因為完成封包傳送還是接收完封包，或甚至是運作錯誤所觸發的。此外由於 ISR 是在中斷期間執行，因此執行時有許多限制，例如：不能和使用者空間互相傳遞資料、不能進入睡眠狀態等事情。在實作 ISR 時，可以使用的資訊即是傳入的參數，這個參數通常以無特定型別之指標(void*)方式傳入，而後在 ISR 中才被轉型為 xxx_softc *的型態，這參數中記錄著表示裝置狀態的旗標，以及匯流排讀寫的方法，經由這些資訊，得到關於如先前設定好的 I/O Port 或 IRQ 號碼，才能進一步的和裝置溝通，以判斷中斷產生的原因，或執行其他需要執行的事情。

2.4 使用 I/O port 讀取及寫入資料

要達到 I/O 的運作有三種可行的技術:programmed I/O、interrupt-driver I/O 和 Direct Memory Access(DMA)。Programmed I/O 裝置傳送資料從系統到裝置有兩種方法：I/O port 和 I/O memory mapped。而驅動程式所使用的即是：interrupt、DMA、I/O port，前面對於中斷已經加以介紹，以下針對於 I/O port 加以說明，以便對於 I/O port 的概念及使用方式更加清楚。

目前硬體 I/O port 的寬度大多有 8,16 及 32 bits 三種，讀取不同的寬度的 port 需使用不同的函式。在 NetBSD 核心中，定義以下的函式用來存取 I/O port (在 arch/i386/include/pio.h)。

```
u_int8_t inb (int port);
void outb (int port, u_int8_t data);
```

inb( )從 8 位元寬度的 port 讀取 8 位元的資料，而 outb( )在寬度 8 位元寬度的 port 寫入 8 位元。兩個函式中的 b 也可換成 w 和 l，這時則分別支援 16 及 32 bits。這些函式在驅動程式中，通常被進一步包裝成成 CSR_READ_1 或 CSR_WRITE_1 來使用。最後面的數字即是 port 的寬度，單位是 byte，所以可能是 1、2 或 4。

除了以上單一讀取的方式外，NetBSD 還支援了字串的操作:

```
void insb (int port, void *addr, int cnt);
void outsb (int port, void *addr, int cnt);
```

insb( )從 8 bits 寬度的 port 讀取 cnt bytes 的資料, 然後將這些資料儲存到記憶體位址為 addr 的地方，而 outsb( )將記憶體位址為 addr 的資料，寫入 cnt bytes 到 8 bits 寬度的 port。兩個函式中的 b 也可換成 w 和 l，這時則分別支援 16 及 32 bits。

2.5 掛上新裝置

當我們要為一個裝置撰寫驅動程式的時候，要先決定兩件事：裝置的名

稱、裝置的種類。在寫好驅動程式之後，程式碼基本上是放在/usr/src/sys/dev/下。並且要通知核心有此一個新的裝置的存在，所謂的通知會牽涉到的三個檔案：conf.h、conf.c、files[4]。

/usr/src/sys/sys/conf.h

在 conf.h 中有兩個 table：bdevsw(block device switch)、cdevsw(chharactor device switch)，此時將裝置加入所屬的 table 中，這 table 中的 entry 主要是要讓核心知道，這個裝置會用到哪些由系統提供的函式，例如：open、close，好讓系統準備好提供給裝置使用。其格式如下：

#define cdev_xxx_init(c,n) cdev__oci_init(c,n)

/usr/src/sys/arch/i386/i386/conf.c

在這個檔案中加入 include file 並且宣示裝置的名稱，其中 xxx.h 是由系統產生的，即是加入下段程式碼：

#include "xxx.h"

cdev_decl(xxx)

接下來是把裝置加入 bdevsw/cdevsw table 中，通常是加在 table 的最下面，以避免其他裝置的混淆，他的寫法是：

cdev_xxx_init(NXXX, xxx) / 71: ... */*

/usr/src/sys/conf/files

這個檔案是說明哪些是合法的裝置名稱：

define xxx 或 *device xxx*

哪些檔案和這些裝置是相關的，一個相關的檔案寫成一行，第二個欄位皆是在/usr/src/sys 下：

file dev/xxx.c xxx needs-flag

有寫 needs-flag 者系統會幫他產生 xxx.h 檔，即是 conf.c 中所引用的。如果要研究驅動程式，由此三個檔案中可以觀察出一些資訊。

3. NetBSD 網路卡驅動程式實例

我們以使用 Intel i82557 相容晶片的網路卡，作為驅動程式追蹤的主要對象。在 NetBSD 中，這張網路卡的裝置名稱為 fxp，使用區塊型的裝置驅動程式。以下就其硬體與軟體架構、驅動程式中的重要資料結構和封包流程加以說明。

3.1 硬體和軟體

網路卡(Networking Interface Card)的硬體裝置和軟體驅動程式之間主要透過 I/O port 及 DMA 機制來搬運資料，底下，我們以圖 4 來簡介硬體及軟體之分工與其各自負責的部分。

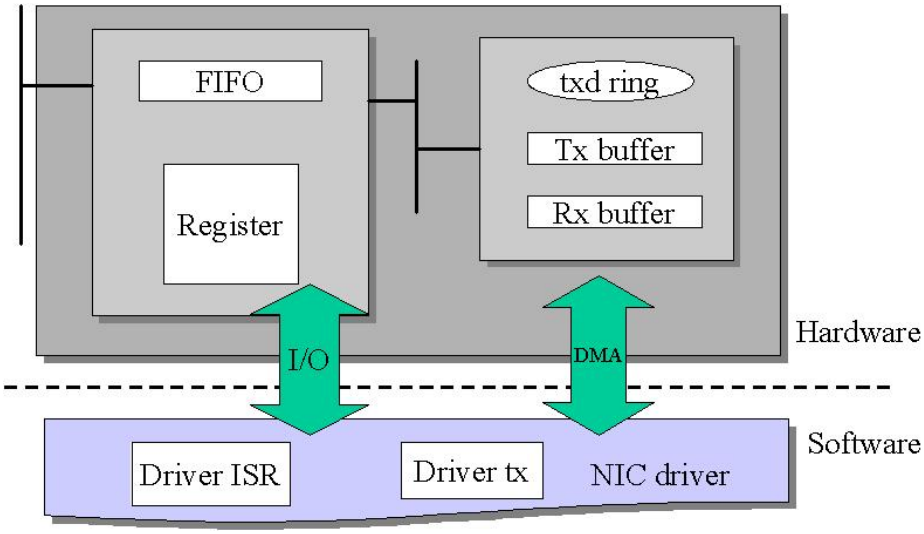

圖 4：網路卡軟硬體架構圖

在硬體方面，可以分成 MAC 控制器、NIC 記憶體兩部分。前者裡頭有：

- 暫存器(Register)：一方面提供驅動程式瞭解目前封包送收的狀態(transmit status & receive status)，另一方面則用來作為下達命令、管理記憶體等動作的控制介面。
- FIFO：資料收發最前線的封包儲存區。網路卡硬體控制的封包送收，都是在這類型可提供快速存取的記憶體暫存。

而硬體中的 NIC 記憶體也可分成接收和傳送兩類型：

- 接收用記憶體：網路卡在接收封包的時候，工作方式為被動的，i82557 的控制器將收到的封包資料存在 receive buffer，再使用 DMA 在 NIC

memory 和主記憶體之間搬運資料。

- 傳送用記憶體：網路卡在傳送資料上扮演著主動的角色，驅動程式一次從作業系統管理的記憶體，搬移一個封包到網路卡的 transmit buffer 上，並配合著一個 transmit descript ring(TDR)，紀錄傳送的相關資訊，例如：傳送門檻、狀態、編號。

至於軟體方面，則是指驅動程式。其中運作時最主要的程序有兩個：

- 中斷服務常式(interrupt service routine, ISR)：負責判斷網路卡所觸發中斷的種類，例如傳送完成通知及接收完成通知。並開始進行觸發後的陸續工作。
- 傳送用 API：負責傳送封包的工作，這個是提供給上層呼叫使用的。

3.2 重要資料結構(data structure)

在 NetBSD 中，網路卡驅動程式最常使用到的資料結構有兩個：ifnet 和 xxx_softc。其中 xxx 是裝置名稱；以 Intel i82557 為例，即為 fxp_softc。

■ struct ifnet

此資料結構定義在<net/if.h>中，表 1 中列出其重要欄位，其中 if_snd 指向 output queue 上的一個封包，可經由這個欄位拿到一個輸出封包。

欄位名	意義
if_softc	lower-level data for this interface
if_bpf	broadcast packet filter structure
if_timer	used for transmit timeout handler
if_flags	interface flag：active, running, etc.
if_snd	output queue (includes altq)

Used by：fxp_start、fxp_init

表 1：ifnet 結構中重要欄位介紹

■ struct fxp_softc

此資料結構定義在<dev/ic/i82557var.h>中，表 2 中列出其重要欄

位，其中 sc_damt 包含許多作 DMA mapping 的方法，而 sc_dmamap 則是描述 DMA mapping 一些細節，例如：要對應的大小。驅動程式可以從 sc_rxq 這個 receive queue 上一次拿到網路卡上的一個封包。

每一個裝置驅動程式都需要有一個資料結構為 xxx_softc，每個裝置的 xxx_softc 內容不一定一樣，其中 xxx 為裝置的名稱，而且驅動程式中的每一個函式都需要以 xxx 作為開頭，例如：xxxattach()。而 xxx_softc 的第一個元素必須要為 struct device 的型態，是因為 autoconfig 系統需要 softc 資料結構被宣告並且其第一個元素為 struct device。還有如果其他較小的裝置要求資訊的話，softc 資料結構可以記載比 device 資料結構更多的資訊和元素。

欄位名	意義	欄位名	意義
sc_dev	generic device structures	sc_dmamap	describes a DMA mapping
sc_dmat	DMA mapping methods	sc_rxq	receive buffer queue
sc_ethercom	ethernet common part	sc_rxmaps	free receive buffer DMA maps
sc_sdhook	shutdown hook	sc_txdirty	first dirty TX descriptor
sc_powerhook	power hook	sc_txlast	last used TX descriptor

Used by：fxp_attach、fxp_intr、fxp_txintr、fxp_rxintr

表 2：fxp_softc 結構重要欄位介紹

3.3 封包發送(Packet Transmit)

關於封包的發送，我們分成控制流程和資料搬移流程來介紹。圖 5 為網路卡驅動程式在發送封包時的控制流程，灰色部分為驅動程式。

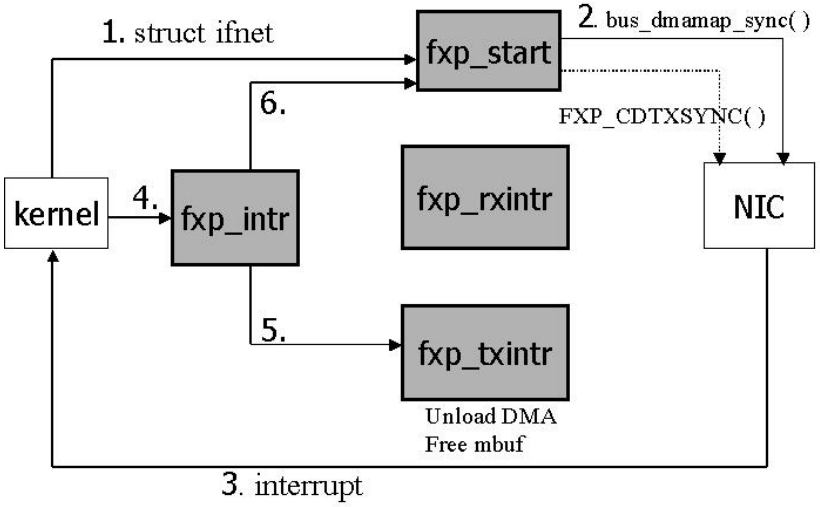

圖 5：封包發送控制流程圖

當核心想要透過某張網路卡發送封包的時候，會呼叫 fxp_start()，並傳入用以描述該網路卡相關設定的 ifnet 當參數。fxp_start()從 ifnet 所指示的 output queue 上將要傳送的封包搬到事前和網路卡講好的 transmit DMA buffer 上，再使用 bus_dmamap_sync()把資料搬到網路卡上，等到 output queue 上的封包傳送完了或者 transmit DMA buffer 已經滿了，fxp_start()會呼叫 FXP_CDTXSYNC()以觸發網路卡將封包送出，等封包送完後會產生中斷來通知核心，於是核心會呼叫適當的中斷處理常式，在此驅動程式中，該常式便是指 fxp_intr()，當中斷處理常式判斷這個中斷是封包傳送完所送的中斷後，他會先呼叫 fxp_txintr()來將 DMA buffer 卸下以及傳送完的 mbuf 鏈結釋放，然後再呼叫 fxp_start()來檢查是否還有需要傳送的封包。

接著我們來描述網路卡驅動程式在發送封包時的資料流程，如圖 6 所示，下方為核心，上方為網路卡(NIC，Networking Interface Card)。

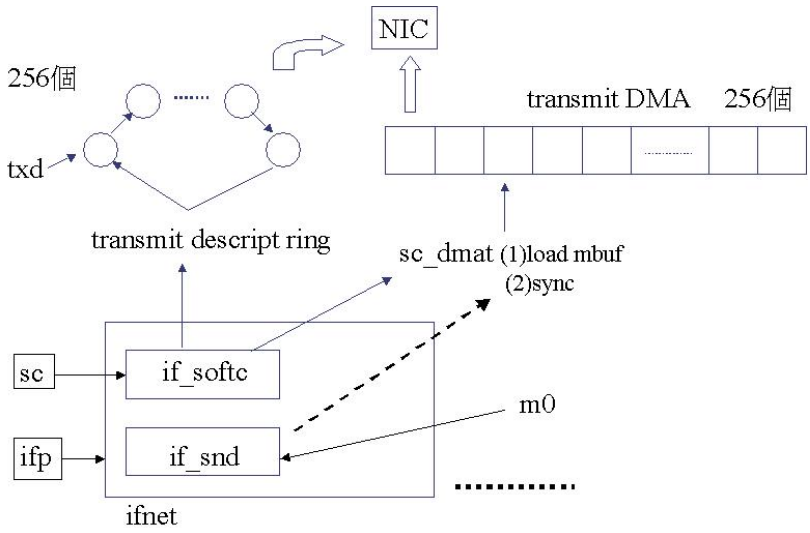

圖 6：封包發送資料流程圖

當網路卡驅動程式得到由核心傳來的 ifnet 資料結構，我們可以從 ifnet ->if_snd 這個 output queue 中得到一個封包 m0，然後再利用 ifp->sc_softc ->sc_dmat 的函式指標執行載入 mbuf 的動作，並將封包 m0 填入；經由 ifnet-> sc_softc 我們可以找到 transmit descript ring，fxp_start 也將傳輸需要的敘述填好，例如：傳輸門檻。等到 output queue 上的封包拿完了或者 transmit DMA buffer 已經滿了，這些封包就會被送到網路上去。

(箭頭為指標，虛線箭頭和白箭頭為資料方向)

3.4 封包接收(Packet Receive)

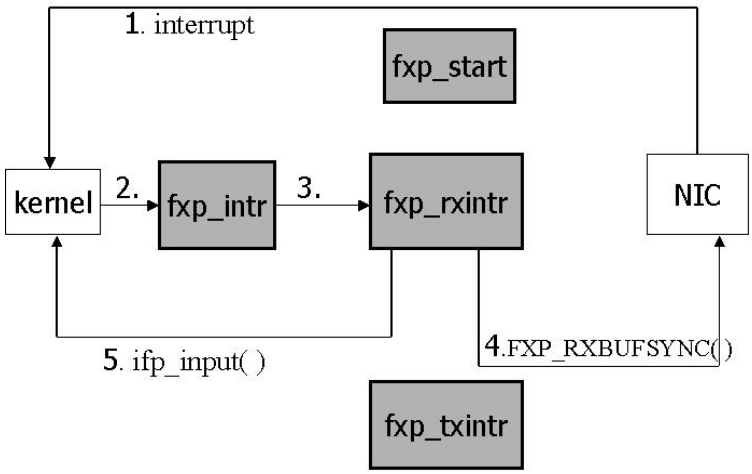

圖 7：封包接收控制流程圖

關於封包的接收，我們也分成控制流程和資料搬移流程來介紹。圖 7 為網路卡驅動程式在接收封包時的控制流程，灰色部分為驅動程式。

當網路卡從網路上收到封包之後，會送一個中斷給核心，告知接收到封包，核心會去呼叫對應的中斷處理常式，在此驅動程式中為 fxp_intr()，當中斷處理常式判斷這個中斷是接收到封包所引起的，就會呼叫 fxp_rxintr()，fxp_rxintr()會從 receive queue 中得到封包，將封包從網路卡上班到系統記憶體中，把封包放到 mbuf 的結構中，並使用 if_input()把收到的這個封包丟往上層處理。

圖 8 為網路卡驅動程式在接收封包時的資料流程，下方為核心，上方為網路卡。

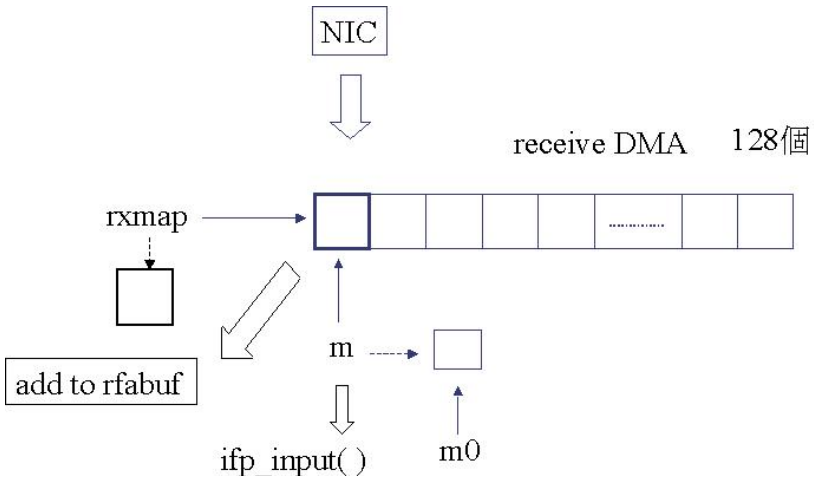

圖 8：封包接收資料流程圖

網路卡從網路上收到的封包會放在已經 mapping 好的 recevice DMA buffer 上，我們可以從 sc_rxq 這個 receive queue 中拿到一個封包，如果封包夠小能放入一個 mbuf 的 header，那就分配一個 header，然後把資料 copy 過去，把原本的記憶體還回給 receive DMA buffer，這種機制在接收大量小封包的時候，能有效的節省記憶體的使用；如果不是小封包就把他加入到 rfa (receive frame area) buffer，並還一塊記憶體給 recevice DMA buffer，將封包搬到記憶體之後，就將指著封包記憶體的指標 m 丟給 if_input()往上層送去。(箭頭為指標，白箭頭為資料方向)

5.結論

網路卡驅動程式在系統中扮演著很重要的角色，處於連結裝置和核心的地位，所以其中牽涉到許多複雜的控制細節和資料搬移處理。在這篇文章中，我們追蹤了 Intel i82557 網路卡在 NetBSD 上的驅動程式。在系統啟動時，驅動程式中的「掛上函式」會先被系統呼叫以掛上網路卡。接著驅動程式會初始，將用來暫存收送封包的緩衝區及卡上相關之暫存器。接著，在網路卡運作時，系統觸發的中斷，會呼叫驅動程式的中斷處理函式，執行接收封包或封包傳送後續處理等動作。一般來說，驅動程式是透過 I/O port 控制網路卡上的暫存器以及 DMA 搬移資料來完成封包收送的這些動作。

基本上，不論是在哪一個系統平台上，網路卡驅動程式的職責是大同小異的，不同的部分主要是因為系統對於封包儲存結構或是 I/O 及 DMA 控制命令的不同，使得驅動程式實作會有些改變。如果想要學習網路卡驅動程式，最快的方法就是直接挑選一個網路卡驅動程式來觀察。當然也可以合併使用核心除錯的工具，例如：kgdb 及 gdb[5]等來追蹤其封包處理的過程，或者 GNU 的 Global[6]套件使追蹤更加容易。

6.參考文獻

[1] George Pajari, *"Writing UNIX Device Drivers"*, Third edition, Amorette Pedersen, September 1992。

[2] *The NetBSD Project*, http://www.netbsd.org/

The NetBSD operating system,
http://residence.educities.edu.tw/rxghome/netbsd/guide/netbsd.html

[3] *Illustrating programmed and interrupt driven I/O,*
http://portal.acm.org/citation.cfm?id=357737&jmp=references&dl=GUIDE&dl=ACM

[4] *Writing a pseudo device*,
http://www.netbsd.org/Documentation/kernel/pseudo/

[5] *Debugging the NetBSD kernel with GDB HOWTO*,

http://www.netbsd.org/Documentation/kernel/kgdb.html

[6] *GNU GLOBAL source code tag system,*

http://www.gnu.org/software/global/