網路驅動程式分類與介面標準

魏煥雲、林盈達

國立交通大學資訊科學研究所
新竹市大學路1001號
gis87517@cis.nctu.edu.tw
ydlin@cis.nctu.edu.tw

摘要

在網路的世界裡，網路卡及網路驅動程式是我們最不可缺少的配備。到底我們使用者在面對林林總總的網路驅動程式時，要如何瞭解其間的關係呢？本文將從「網路驅動程式的分類」切入，瞭解「介面的需求與標準」，並研究「DOS 網路卡驅動程式」、「Windows 網路卡驅動程式」的實作細節與原理，以透視網路奧妙又複雜的世界。

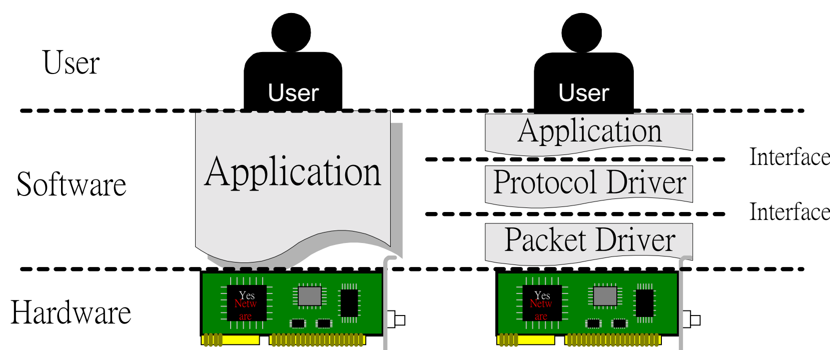
關鍵字：網路卡驅動程式、Protocol Driver、Adapter Driver、Packet Driver、ODI、NDIS、VxD、Windows、DOS。

1．網路驅動程式分類

網路技術，和其他的電腦技術有一個很大的不同點，就是多台電腦通訊。電腦上其他的硬體驅動程式（如顯示卡、音效卡），只要管好自己機器上的事，鮮少有人會限制你要怎麼做；網路驅動程式要考慮到的不只是管好自己，還要跟其他電腦通訊！因此網路驅動程式就會形成一層一層的協定堆疊：最底層控制硬體的「網路卡驅動程式（Adapter Driver）」、中間層與其他電腦搭起溝通橋樑的「協定驅動程式（Protocol Driver）」及最上層爲各種用途設計的「應用程式（Application）」。粗略來說，我們就

可以分類網路驅動程式爲這三類：Adapter Driver、Protocol Driver、Application。其中 Protocol Driver 通常會包含 OSI model 裡的數層（如 Network layer、Transport layer）。在網路演進的過程中，這幾個 Driver 之間的介面也浮現出「標準化」的需求。

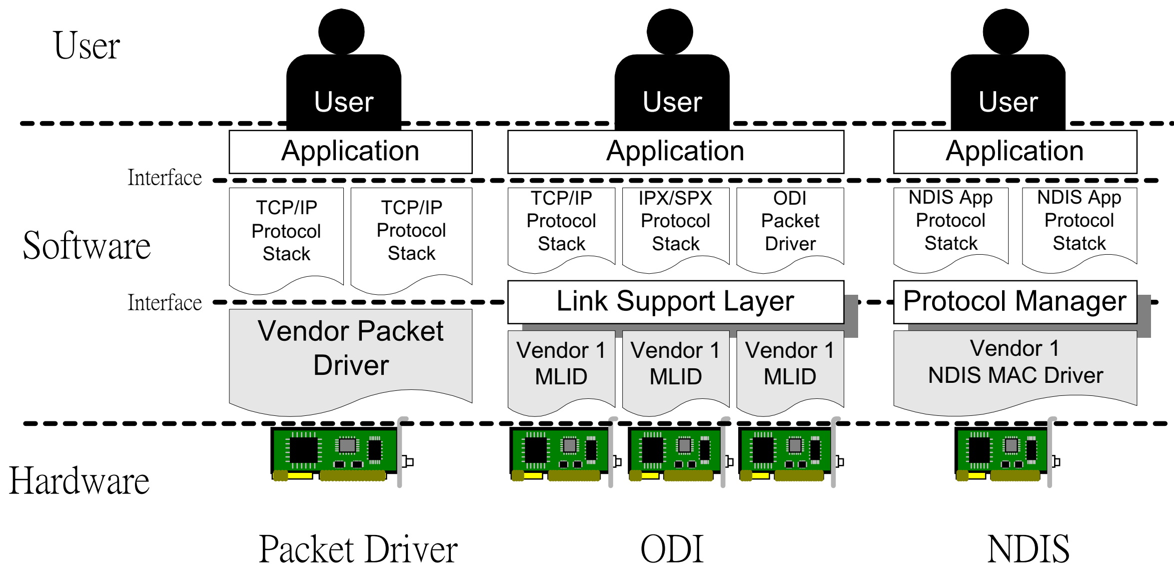
而從過去的演進來看，我們更會發現一項有趣的事情——網路軟體的分工：從早先網路軟體只有 Application，到分工出 Adapter Driver，再到分工出 Protocol Driver。我們不禁要問起，這樣子的演變對我們的貢獻在哪裡呢？其實，對使用者的貢獻不大，但是對網路卡廠商貢獻可就大了！試想，早期網路卡廠商推出的產品，其提供的軟體不但得控制最低層的硬體，尙須具備與其他電腦連線的協定堆疊，甚至還要涵蓋最上層各種用途的應用程式，這對廠商來說可不是件輕鬆的工作（圖一左）。在網路軟體分層分工後，Application 就比較容易撰寫了（圖一右）。然而由於各層之間的介面（Interface）是廠商內部私定的，而非標準，因此各層的 software 幾乎還是得由同一個廠商來撰寫。



（圖一）網路軟體分工前與後（灰色部分爲廠商所需撰寫的程式）

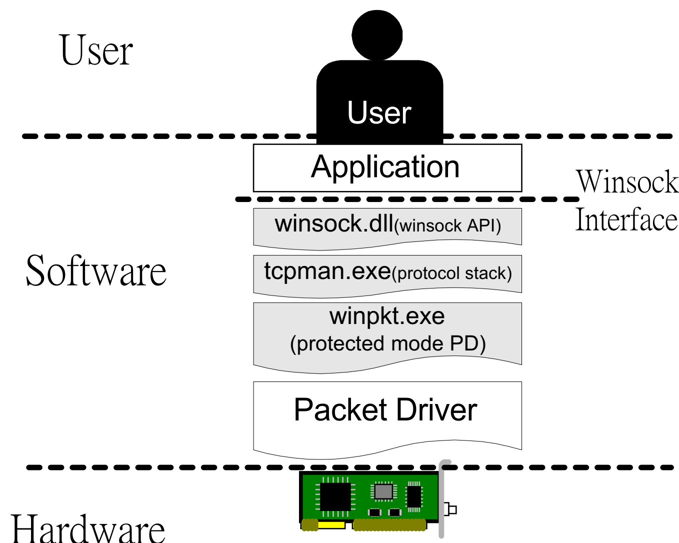
2. 標準介面

爲因應上述問題的需求，各層軟體間的介面「標準化」成爲當務之急。1987 年 FTP software 推出的 Packet Driver 介面（圖二左）首先出現，訂出 Adapter Driver 和其上層之間的介面，之後網路卡廠商終於可以喘口氣了。隨後 Novell 的 ODI（Open Data-Link Interface）及 Microsoft、3Com 的 NDIS（Network Driver Interface Specification）標準介面（見圖二中、右）也分別在 1989 及 1991 年問世。值得注意的是後兩者中更出現了「協定管理員（Protocol Manager）」的角色，用以管理、配對、連結網路卡驅動程式及協定驅動程式。這些標準讓網路卡廠商可以專心研發網路卡的 Adapter Driver，不用花心力去研發各種協定堆疊驅動程式了。其他廠商若提供支援這些標準介面的 Protocol Driver，就可連結（Bind）在一起工作。



（圖二）PD、ODI與NDIS介面（灰色部分為網路卡廠商所需撰寫的程式）

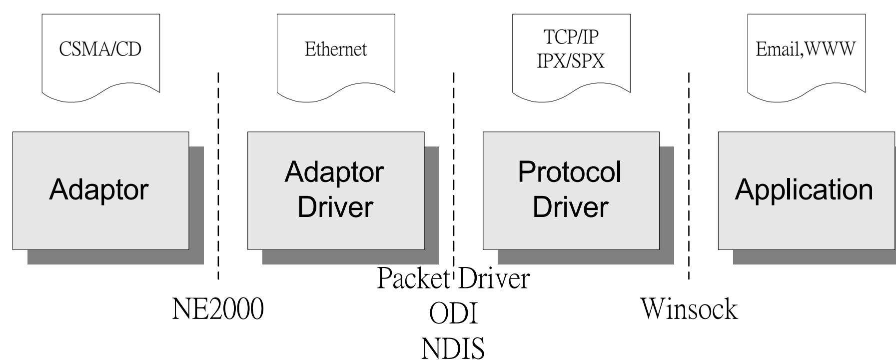
另外一個浮現的問題是：Application 和 Protocol Driver 之間的介面也需要標準化。1993 年幾個大廠商共同推行的 Winsock 就是一個「協定驅動程式」和「應用程式」之間的一個標準介面（見圖三）。在早期的 Windows 環境下，要撰寫一個 TCP/IP 的應用程式，和在 DOS 下一樣，是沒有標準的。A 寫的應用程式得配合 A 廠商寫的協定驅動程式才能跑；B 寫的應用程式得配合 B 廠商寫的協定驅動程式才能跑。兩家的應用程式或協定驅動程式不能互換，只因為「應用程式介面」（API）不一樣，所以雖然是做一樣的事，也要各自寫各自的程式；現在大家只要按照 Winsock 的介面撰寫應用程式，就可暢快地在有安裝 Winsock 的系統上執行了！



（圖三）Windows 3.1下的winsock介面
（灰色部分為Protocol Driver廠商所需撰寫的程式）

在上文提到在「Adapter Driver 與 Protocol Driver 之間的標準介面」，也看到了「Application 與 Protocol Driver 之間的標準介面」，也許你您不禁會懷疑，是否在 OSI model 中其它層之間也有標準呢？由於大多數盛行的網路協定並不完全符合 OSI model，缺乏 session 和 presentation 層的定義，使得這幾層的網路協定沒有活絡起來。因此還有可能標準化的界面只剩下「Adapter Driver 和 Hardware」之間的介面了。那這兩層之間有沒有標準介面呢？答案當然是 —— 沒有！ 網路卡廠商在設計硬體時爲了達到最好的效能，採用的技術會有所不同，自然在線路，邏輯，控制方法也會不同，因此廠商都要爲自己設計的硬體奉上能駕御它的「網路卡驅動程式（Adpter Driver）」。在實際使用上，我們很少可以把廠商 A 的網路卡驅動程式拿到廠商 B 的網路卡上使用，就是這個道理。因此這一界面除了是「不需要」，更是「不能要」，否則廠商便無從競爭了。然而在商業上，除了「標準」，更有「主流」！在音效卡領域裡，廠商幾乎一面倒要與 Sound Blaster 相容；在網路卡的領域裡，Novell 出的的 NE2000 網路卡就是一個例子。廠商若推出與其相容的卡，在 driver 的寫作上會有較多的資源，網管人員也可很方便地管理（網路卡插了就可以跑，甚至連 driver 都不用換）。

NE2000（Novell Ethernet，2KB memory）這張網路卡所採用的 MAC controller 是相容於一顆叫 DP8390 的 chip，所以一張網路卡若要相容於 NE2000（NE2000 Compatible），它的 MAC controller 也得要相容於 DP8390 才行。所謂「相容」不是指 controller 的 chip 接腳功能相同，而是指 controller 內部 register 的功能排列相同。例如現在有一張 NE2000 網路卡 base I/O address 是 0x300，那它的 ISR，TSR，RSR 等 register 的 I/O address 必定是 0x307，0x304，0x312 等（每一個 I/O address 對應一個 register）；一張卡若想要與 NE2000 相容，想當然爾它的 controller 裡 register 不但要與上述排列相同，而且 register 裡的哪個 bit 代表什麼意義也要一樣！Adapter Driver 所關心的是 register，它和 controller 之間以 register 溝通的，在下文會詳細剖析網路卡（Adapter）與 Adapter driver 之間的細節。縱合以上所述，Adapter 與 Adapter Driver 之間的雖然沒有標準界面，卻有有主流。 「網路驅動程式」與「界面標準（或主流） 」之間的關係也就可以圖四來釐清:



（圖四）「網路驅動程式」與「界面標準(或主流) 」關係圖

3 . DOS及Windows網路驅動程式實作

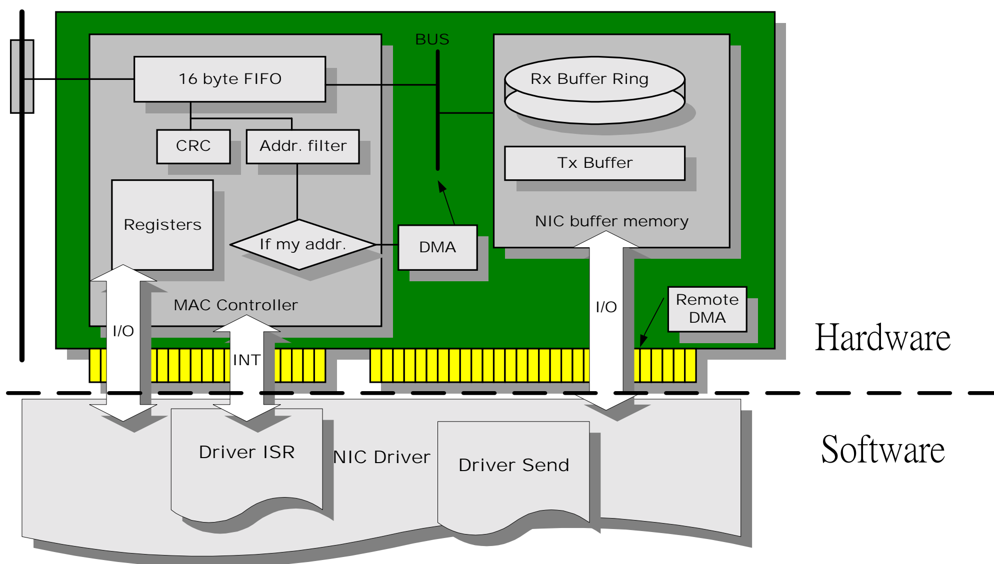
歷經標準介面的洗禮，現在網路卡廠商終於只要專心寫網路卡的驅動程式就可以了。因此我們買網路卡裡面附的驅動程式，幾乎只提供合乎標準介面的 Adapter Driver。好景不常，多工的 windows 環境出來攪局了。不同的網路程式透過同一張卡要收送資料，要如何才做得到呢？底下我們就先研究 DOS 下的網路卡驅動程式的架構，再介紹在多工的 WINDOWS 下要如何因應，才能讓我們在多工的環境中享受到網路的便利。

3.1　DOS網路驅動程式實作

雖然我們已經有了驅動程式概念的巨觀，但是詳細地去檢視細部的動作，可能會混淆哪些是軟體（driver）做的事、哪些是硬體（controller）做的事。是故我們以最常見的 NE2000 網路卡爲例，看看在 DOS 下軟體和硬體如何分工合作：

3.1.1軟體（driver）vs. 硬體（controller）

在圖五中，上面的部分是硬體，下面的部分是軟體，其間透過 IRQ 與 I/O port 來溝通訊息。以下就先分別簡介軟體和硬體的各部分模組（module），再以傳送（Transmit）及接收（Receive）的流程來說明軟體和硬體如何分工合作：



（圖五）網路卡與驅動程式的各部分模組

Ⅰ. 硬體模組（**Hardware Module**）：

一張網路卡（NIC, Network Interface Card）上最容易看到的不外乎 MAC controller、NIC memory，簡單分述如下：

MAC controller

網路卡上最大的那一塊晶片，通常應該就是所謂的 MAC controller。這傢伙裡面藏有許多邏輯單元（logic unit），以控制網路卡對網路線存取的運作，使其按照標準協定（Ethernet 上即為 CSMA/CD 協定）。對程式設計師比較有關係的是裡面的暫存器（register），寫 Driver 的人可以用 I/O port 與 register 溝通。底下就列舉一些 controller 裡面比較重要的東西：

Registers —— Controller 裡有許多的暫存器，供程式設計師讀取收發狀況（receive & transmit status）、下達命令、管理記憶體（memory management）等。

DMA —— 在 NE2000 的 Controller 中有兩組 DMA channel。Local DMA 專門負責搬動「NIC memory」與「MAC controller 的 FIFO」之間的資料；Remote DMA 則專司搬動「NIC memory」與「主記憶體」之間的資料。

FIFO —— 資料收發的最前線戰地，一送一收都是在這緩衝區暫存的。

NIC memory

Receive —— 網路卡在接收資料的時候，工作方式為被動的，因此其 buffer 要有特殊的設計，才能因應網路資料忽多忽少（burst）的特性。DP8390 這個 controller 將 Receive 進來的資料存到像 ring 狀的資料結構裡，並用一些 status register 來更新 ring 的參數、狀態，以利 driver 將資料搬到電腦的記憶體中處理。

Transmit —— 相對地，「傳送」是扮演主動的角色。Driver 一次送一個封包，網路卡送完封包發出中斷通知 Driver 後，Driver 再送下一個封包。因此，沒有必要如 Receive Buffer Ring 一樣的設計。

Ⅱ. 軟體模組（**Software Module**）：

由上面的知識，我們可以推論最簡化的網路卡驅動程式會有兩個基本模組，分別是中斷服務程式以及 API，分述如下：

Driver ISR（中斷服務程式）：用來處裡中斷要求的副程式，如 Receive 或 Transmit 的 Error/Success 處理等。

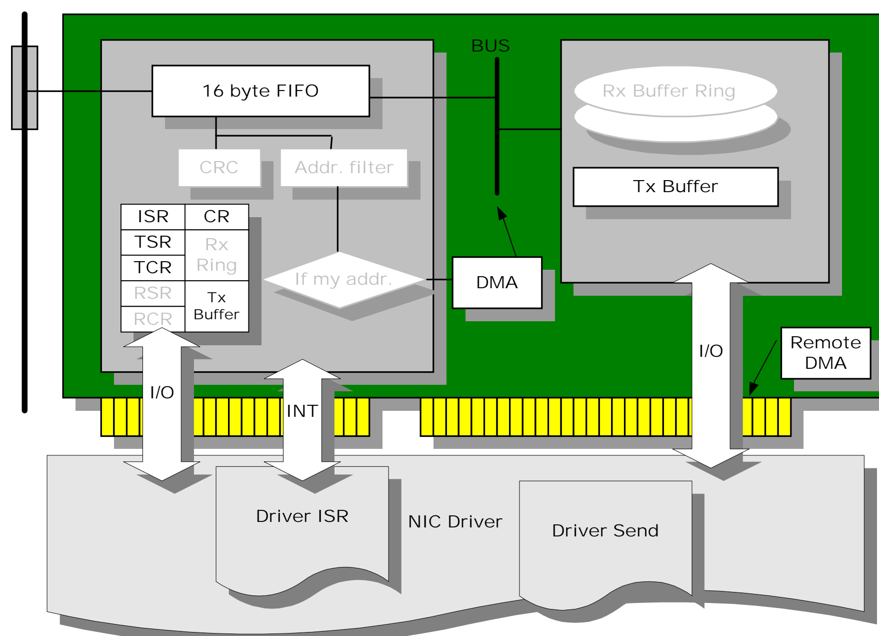
Driver Send（API）：提供給上層軟體呼叫的 API

3.1.2 網路卡與驅動程式的互動

看過了軟體、硬體的模組後，可以從資料收發來一窺軟硬、體互動的模式：

I．資料傳送

應用程式有資料要傳送時會呼叫 Driver Send 模組，把資料從主記憶體搬到網路卡記憶體，而後 Driver Send 模組會利用 I/O port 下達「送」的指令給 MAC Controller（寫入 Command Register, CR），DMA 就會乖乖地將資料搬到 FIFO 上送出去，並回報傳送的狀況於 TSR（Transmit Status Register），最後觸發中斷訊號通知 Driver ISR 來處理例外情況，如 FIFO underrun、Collision 等（見圖六）。

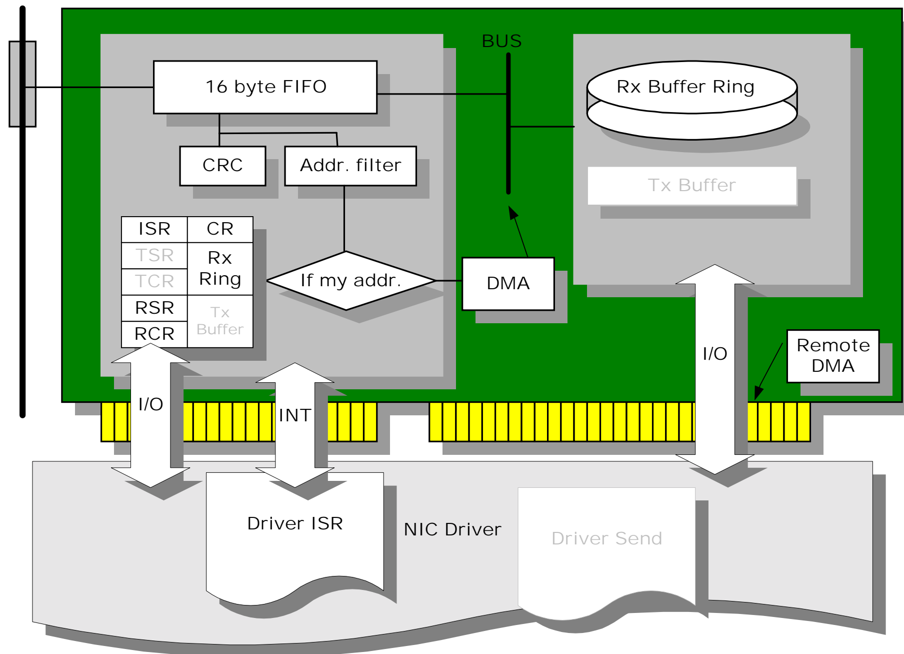


（圖六）網路卡傳送封包時會用到的模組（用不到者以淺色表示）

II．資料接收

網路卡在偵測到網路線上有屬於自己的封包後，DMA 會自動把封包存入 Rx Buffer Ring、更新與 ring 有關的 register、並回報接收狀況於 RSR（Receive Status Register）後，發出中斷通知 Driver ISR。Driver ISR 被動地被中斷後，由讀取 RSR 來得知接收的結果並處理例外狀況，例如接收成功就搬動 ring 裡的資料到主記憶體內；發生 Rx buffer ring overflow 就啟動一些程序去 recover 等。另外值得留意就是圖中看來似乎只接收與自己 address 相同的 frame，其

實這可以經由設定 RCR（Receive Configuration Register）的位元來決定是否收取 broadcast 封包，或甚至網路上全部的封包（見圖七）。



（圖七）網路卡在接收封包時會用到的模組（用不到者以淺色表示）

由撰寫網路卡驅動程式過程中，會得到一些很有趣的心得。譬如說以前若用韌體儲存設定值的網路卡，在設定程式找不到後，可能就要試很多不同 I/O port 及 IRQ 的排列組合。而心得告訴我只要用反組譯程式 debug 的 in 指令去讀取某些 I/O port（例如進入 debug 後下達 in 300），如果有回應 21/22 即可得知此網路卡的 I/O port 設定在此，接著只要試幾個 IRQ 就可以試出來了。現今流行的 Windows 95 系統在自動偵測硬體的時候，亦多是利用偵測 I/O port 來觀察硬體回應的訊息，以判斷硬體的型態。

以上的例子是以 NE2000 網路卡來解說，當然現在已經有很多更先進的網路卡出現了，他們的技術也與上述有很大差異，譬如有些 PCI 界面的網路卡採用 memory-mapped 的方式（類似顯示卡存取記憶體的方式）來存取網路卡上的記憶體，而非上述所介紹的 I/O mapped 方式。 但總歸來說，一個網路卡的控制方法、觀念大致上與前述不會差很多的。

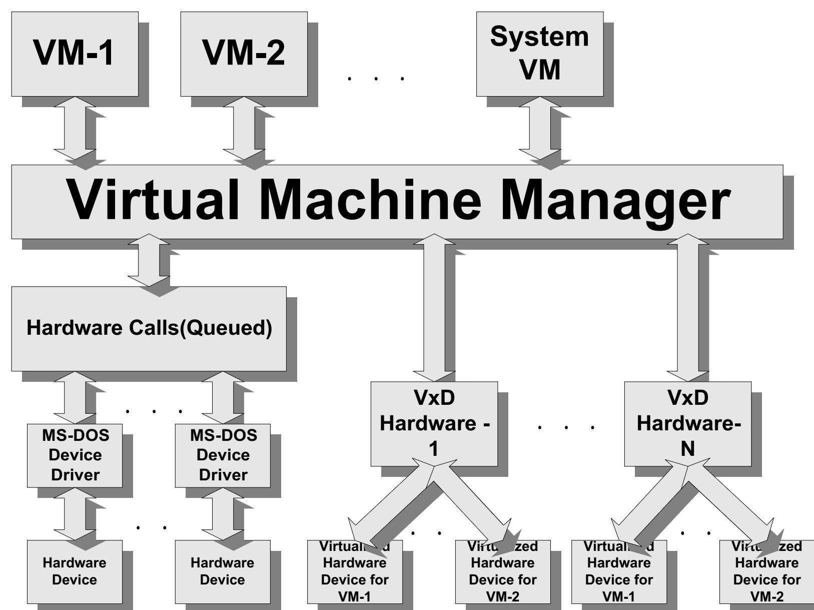
3.2 Windows網路驅動程式實作

在 Windows 多工系統下，多個程式共用系統的資源，若沒有一個仲裁者，必將造成混亂。因此我們若要研究其網路架構，就不得不先看看 Windows 本身的架構：

3.2.1 Windows 架構

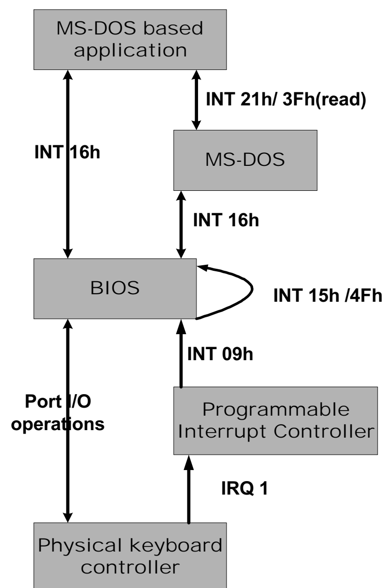
Windows 是一個複雜的作業系統，我們將其功能簡化以後，以圖八來表示。VMM（Virtual Machine Manager）是 Windows 的系統核心，其在初始化的時候建立 System VM 並把 Windows 系統載入 System VM。若在 Windows 中執行視窗程式或 Win32 Console 程式，則他會跑在 System VM 中；若在 Windows 中執行傳統 DOS 程式，VMM 就會再製造 VM 出來模擬 DOS 環境，使 DOS 程式「感覺」他完全擁有這台機器。

Windows 能做到這樣的工作，是仰賴著一種特殊的驅動程式——Virtual Device Driver（VxD）。VxD 的 x 是泛指電腦上的 device，所以 VKD 指的是 Virtual Keyboard Driver、VDD 指的是 Virtual Display Driver。不同於以往 DOS 的驅動程式（圖八左），Virtual Device Driver 必須做到讓「每一個 VM」都認爲，「自己獨佔此 Driver 描述的硬體」（圖七八）。網路卡亦是電腦上的 device，想當然爾也會需要有對應的 VxD，才能提供多工的網路環境給不同 VM、不同 Process。



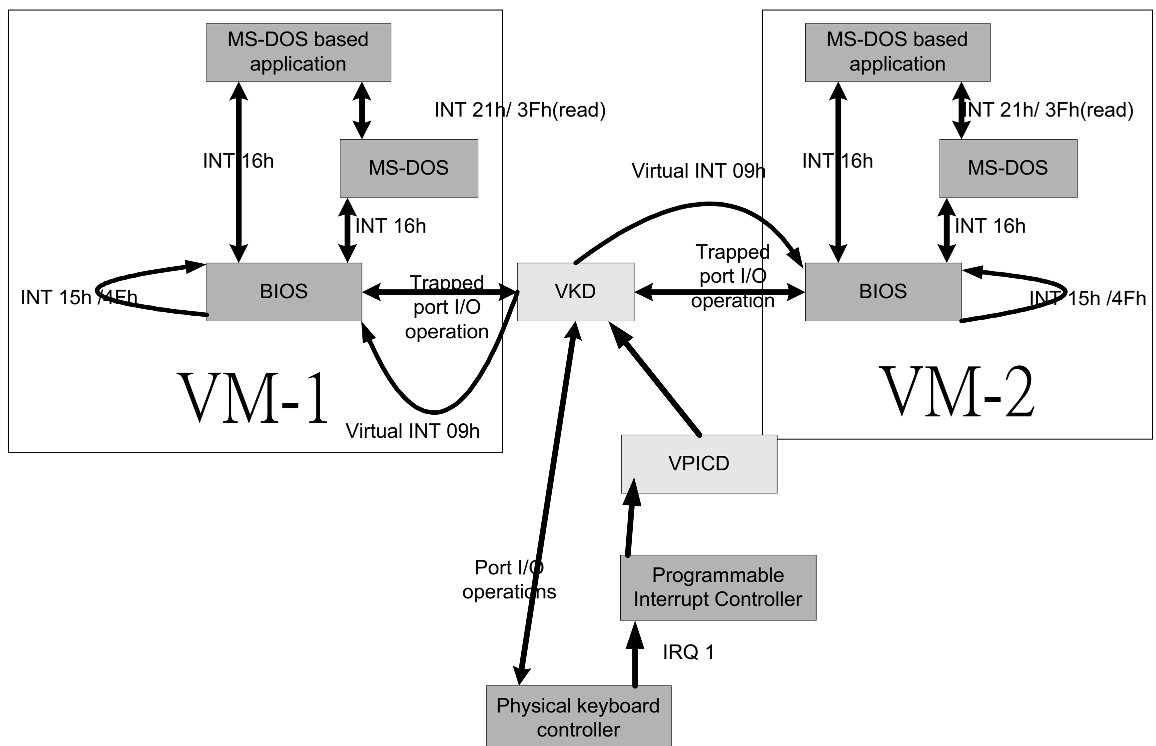
（圖八）Microsoft Windows虛擬機器架構圖

舉一個 DOS 下鍵盤等待使用者輸入的例子（見圖九），如果你學過組合語言，應該很清楚圖在描述什麼。簡而言之，就是應用程式透過 DOS 呼叫 BIOS 或直接呼叫 BIOS 中處理鍵盤 I/O 的程式以讀取鍵盤，BIOS 利用 I/O port 去存取鍵盤中的 register 以與 keyboard controller 溝通，在鍵盤讀到使用者按鍵後透過 IRQ 提出中斷要求，執行 BIOS 中鍵盤的 ISR 來讀取資料到電腦裡，並經過層層傳遞到 DOS 應用程式，就可以利用來做任何事了。想想看，如果我把 BIOS 換掉成前文詳述的網路卡驅動程式，Physical Keyboard Controller 換成 Network Media Access Controller，你是否也能把網路卡運作的流程說一遍呢？



（圖九）DOS下鍵盤等待使用者輸入的流程

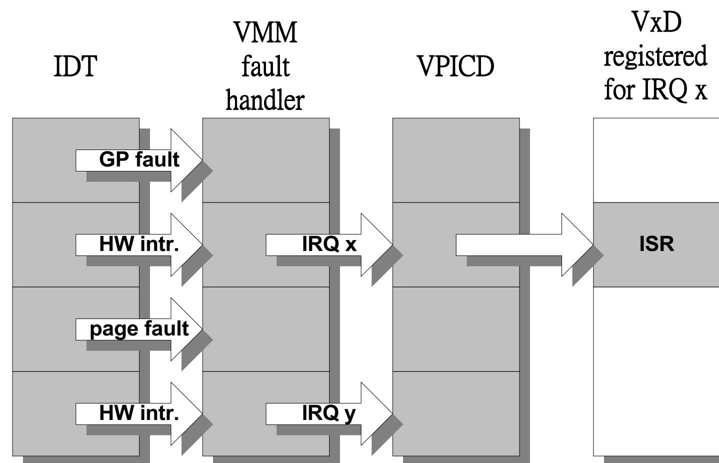
當然，我們現在要研究的是 Windows 的驅動程式。在研究以前，想想看，你會怎麼做呢？在圖十就可以看到在 Windows 下鍵盤的運作方式：



（圖十）Windows下多個「鍵盤等待使用者輸入」程式的運作方式

乍看之下，似乎很複雜，其實是因爲同時有兩個 MS-DOS 程式在跑的關係。

圖十比圖九多出了兩個顏色不同的 VxD 模組：VPICD（Virtual Programmable Interrupt Controller）和 VKD（Virtual Keyboard Controller）。VPICD 是一個虛擬的中斷控制器，負責將硬體的中斷傳遞給對這中斷有興趣的 VxD 模組，在此例中，就是 VKD。VKD 很類似中斷處理程序（Interrupt Service Routine），而他更可以傳遞中斷給 Virtual Machine。如此 DOS 應用程式就好像感覺他是真的跑在 DOS 的環境下。若我們再詳細研究 VPICD 的運作方式，可以圖十一做說明：



（圖十一）VPICD的運作方式

圖十一中，IDT（Interrupt Descriptor Table）是一般 PC 用來儲存各中斷處理程序（ISR）位址的地方，例如 IDT 中有些欄位儲存 BIOS 中的 ISR 的位址；有些欄位儲存 TSR 程式裡 ISR 的位址等。在 Windows 系統中，我們希望所有的硬體中斷都交給 VPICD 管理，由其來指派（dispatch）硬體中斷給 ISR，而不要雜亂無章地把硬體中斷、軟體中斷、例外處理全都擺在 IDT 中。因此我們通常會把 ISR 寫在 VxD 中，並對 VPICD 註冊某個硬體中斷，代表我這一個 VxD 對此中斷有興趣，若有此中斷發生請傳遞給我（見圖十一）。

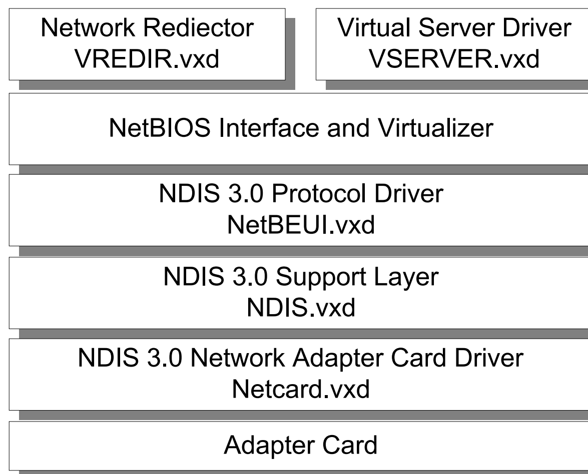
為了要把周邊硬體設備（Peripheral Device）的中斷任務交給 VPICD，Windows 把全部 IDT 欄位都指向 VMM fault handler。VMM 若被告知一些嚴重的 exception 如 GP（General Protection）fault、Page fault 等，就自己處理；若被告知周邊 device 產生中斷，就全權交由 VPICD 去分配中斷給有興趣的 VxD 模組。這些模組裡可能有 ISR 處理中斷，但也可以再傳遞給 Virtual Machine（如 DOS 裡的 ISR）處理。舉一個滑鼠的例子來說，若圖九的「VKD」換成「VMD」,「VM-1 中的 BIOS」換成「DOS 下的滑鼠驅動程式 ISR」,「VM-2」改成「System VM」，若我們用滑鼠單擊 VM-1（DOS 視窗），滑鼠的中斷會經 VMD 傳至 VM-1 中的「DOS 滑鼠驅動程式 ISR」處理，而不是在 VMD 被處理。

當然，VxD 存在的目的並不全然是為了要多工的關係，上有許多理由，例如可以直接與硬體溝通、跑在 80x86 CPU ring 0 較有效率、Client Server 架構（VxD 可提供 service 給其他 VxD、DLL 或 App 使用）等。那到底一個 VxD 長什麼樣子呢？可以想見地，它不是一個單獨可以跑的程式，而是一個專門處理訊息或提供

service 給其他模組的副程式。VxD 的架構可以分兩項來描述：DDB(Device Descriptor Block)與 DCP(Device Control Procedure)。在 DDB 裡，主要做的是一些宣告的動作，如宣告我對哪些訊息有興趣？對應的處理程序是什麼？我要提供那些 service 給其它模組？諸如此類。而 DCP 就是放「處理訊息的程序」的地方了。

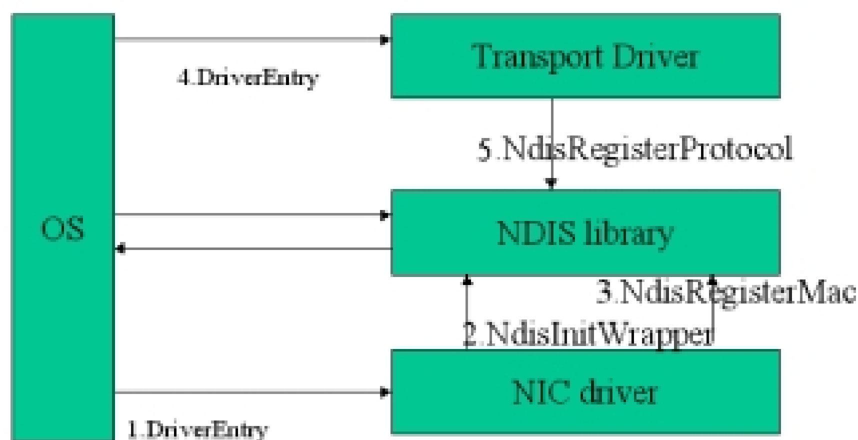
3.2.2 Windows 95 網路架構

講了這麼多，到底 Windows 95 裡的網路架構是如何呢？我們舉一種協定堆疊來看（圖十二）：



（圖十二）Windows 95網路架構下的一種協定堆疊

綜合前文中，「標準介面的需求」及「多工、Client-Server 架構」等原因，Windows 採用 NDIS 3.0 介面、並實做成 VxD 的方法，提供上層網路功能。圖中 NDIS.vxd 功能與 NDIS 2.0 中的 Protocol Manager 功能相同，而 NDIS 3.0 更可以同時支援多個網路介面卡。其上層協定驅動程式以及其下層的網路卡驅動程式都得先與其註冊才能使用，因此在撰寫 NDIS 網路卡驅動程式的時候，會有很多註冊程序（見圖十三）。



（圖十三）NDIS3.0 中 Driver 對 NDIS 註冊的過程

由於 Windows 驅動程式在多工、多網路介面卡的環境下工作，很多事得透過

NDIS library 提供的 function 會較好管理（例如記憶體管理），所以 NDIS3.0 的網路卡驅動程式寫起來會覺得挺複雜、不是那麼直覺。不過大抵說來，其作法的中心思想與在 DOS 下的網路卡驅動程式是大同小異。

4．結論

網路是一項迷人的技術，既提高群組生產力，更讓我們足不出戶就可以知天下事。在 PC 上的網路驅動程式演進過程中，可以看到有越來越複雜的趨勢，而這些都是爲了讓使用者更方便地使用網路。因此，未來使用者將會越來越輕鬆，而大難就降在程式設計師身上了。不過若能有效地利用一些工具，便可以減輕驅動程式的發展難度。在工具上，Microsoft 的 Device Driver Kit（DDK）是必備的（尤其是文件、範例等）。使用 DDK 得會一些組合語言，這是無法避免的。若使用 Vireo 所出的 VtoolsD 則可以不用到組合語言，甚至可以使用 C++來撰寫驅動程式；其提供的 class library 和工具非常犀利，值得我們好好去利用。心動了嗎？有你的加入，網路的世界將會更多采多姿！！

5．參考文獻

[1] Sanjay Dhawan *"Networking Device Drivers"*， ITP publishing Inc.，1995.

[2] Walter Oney, *"System Programming for Windows 95" p24~p27,* Microsoft Press, 1995.

[3] Karen Hazzah, *"Writing Windows VxDs and Device Drivers" p108,* R&D Books, 1997.

[4] *"DP8390 Chip Specification",* National Semiconductor, 1993.

[5] *"PC/TCP Packet Driver Specification, Revision 1.09",* September 14, 1989. FTP Software, Inc. 26 Princess Street, Wakefield, MA.

[6] Malamud, Carl, *"Analyzing Novell Networks",* Van Nostrand Reinhold, New York, NY, 1990.

[7] *"Network Driver Interface Specifications"* 3COM / Microsoft Version 2.0.1 Final Draft, June 1991.

[8] *"Microsoft LAN Manager, Network Driver Interface Specification (NDIS), Version 3.0",* Microsoft Corporation, 1 Microsoft Way, Redmond.