The Linux Plug-and-Play HOWTO 中文版

作者: David S.Lawyer *bf347@lafn.org*
譯者: 杜克石 *rock@371.net*

v0.00, November 1998

幫助你理解和使用 PnP 設備. 如何使你的 Linux 支持 PnP. 第一個版本是不完整的, 但也許對你有用.

目錄

1 介紹

1.1 版權, 商標, 不擔保, 和 信用

本小節請看原文

1.1.1 版權

4. Make a good faith effort to contact the maintainer (or copyright owners if there is no maintainer) to let them know what you have done. If the changes are extensive, then you should also attempt to make more such contacts (including prior to your project).

5. Give full credit to significant previous authors and contributors although the credits section need not exceed 1% of the length of the document.

1.1.2 商標

If certain words are trademarks, the context should make it clear to whom they belong. For example "MS Windows" (or just "Windows") implies that "Windows" belongs to Microsoft (Micro$oft).

1.1.3 不擔保

Much of the info in this HOWTO was obtained from Serial-HOWTO, the Internet, sales clerks, etc. and may be unreliable. While I haven't intentionally tried to mislead you, there are likely a number of errors in this document. Please let me know about them. Since this is free documentation, it should be obvious that neither I nor previous authors can be held legally responsible for any errors.

1.2 未來計劃: 你也可以幫忙

請告訴我你的種種想法. 但首先注意, 如果你手中的版本是一個月以前的, 請看是否有新版本.

此 HOWTO 第一個版本, 0.00, 我甚至沒有看書: PnP 體系結構, 也並不完全理解 PnP. 也沒有比較二個把 Linux 變成 PnP 操作系統的補丁的不同. 我不會充分解釋 BIOS 如何設置 PnP(更不要說補丁如何做了). 此 HOWTO 是不完整的並且可能不正確(假如我錯了, 請告訴我). 在此 HOWTO 中我使用 ?? 來表示我不知道答案. 你想改進(重寫)和維護此 HOWTO 嗎? 我正在找人.

1.3 此 HOWTO 的新版本

此 HOWTO 的新版本可以在 LDP 的鏡像站點上找到. 首先看 `<http://sunsite.unc.edu/LDP/mirrors.html>`, 當你找到一個近的鏡像站點後, 點擊 "Linux Documentation Project" (LDP), 在其它格式 HOWTO 中找. 如果你僅僅想知道最新版本的日期, 你不必使用鏡像站點, 看 `<http://sunsite.unc.edu/LDP/HOWTO/Plug-and-Play-HOWTO.html>` 即可.

2 PnP 怎麼做: 分配 "資源"

2.1 什麼是 Plug-and-Play (PnP)?

Plug-and-Play 就是自動配置(低層)計算機中的插卡和其他設備, 然後告訴對應的設備都做了什麼. Plug-and-Play 的任務是把物理設備和軟件(設備驅動程序)相配合, 並操作設備, 在每個設備和它的驅動程序之間建立通信通道. 換種說法, PnP 分配下列資源給設備和硬件: I/O 地址, IRQ, DMA 通道, 內存段. 假如你不理解這 4 項, 看下面的. 一旦這些資源被分配, 設備(名字在 /dev 目錄中)就准備好被使用(倘若這些物理設備在你的 PC 中).

PnP 配置資源, 但僅是一定程度上的配置資源. 盡管使用了 PnP, 許多設備仍不是被 PnP 來配置. 如調制解調器的配置, 'init string' 被通過 I/O 地址通道送給調制解調器, 盡管 I/O 地址通道是由 PnP 分配的, 但 'init string' 與 PnP 無關. 設置串口的速度是由用户執行程序來做的而不由 PnP. 所以當説起 PnP 的時候, '資源'意味著資源中的一部分, '配置'意味著某些類型的配置.

2.2 計算機如何找設備(和 反過來)

計算機包括 CPU 來運算, 内存來存儲程序和數據. 還有一些設備, 如磁盤驅動器, 顯示卡, 鍵盤, 網卡, modem 卡, 聲卡, 串口和并口, 等等. 還有電源提供電能, 主板上的各種總線把設備和 CPU 連在一起, 機箱把它們裝起來.

過去, 許多設備都是插卡(印刷線路板). 今天, 除了插卡之外, 許多設備已經小到一塊芯片被做在主版上. 主版上的插卡也許包含不止一個設備. 内存有時也被認爲是設備但不是此 HOWTO 中的 plug-and-play.

計算機要正常工作, 每個設備必須在它的驅動程序(它也是操作系統的一部分, 在CPU上運行)控制下運行. 驅動程序與 /dev 目錄下的特殊文件聯系在一起, 盡管他們不是眞正的文件. 它們有象 hda1, ttyS0, eth1 等這樣的名字. 麻煩的是選擇特殊設備的驅動程序, 就説 eth1 吧, 依賴于你使用的網卡的類型. 這樣 eht1 就不能分配給除了可以使你安裝的網卡正常工作的驅動程序外的其它驅動程序. 爲了控制設備, CPU (在驅動程序的控制下)向設備發送命令和從設備讀取信息. 要這樣做, 每個設備驅動程序必須知道設備的接口地址, 比如用于通信的地址. 知道這樣的地址就如同建立了一條通信通道, 即使這個通道是 PC 里的許多設備共享的數據總線.

PC 有 3 類地址空間: 内存, I/O, 和配置(僅限于 PCI 總線). 只有前兩個(内存, I/O)被 PnP 配置. 在 PC 里這3種類型的地址共享同樣的總線. (另外: 對于 PCI 總線, 它用于傳輸數據). 但是由特定電路來告訴設備地址是在 I/O 空間或在内存空間.(譯者注: 其實就是總線控制電路中的内存使能線, 當它爲 0 電平時, 表示當前地址是内存地址, 爲 1 電平時, 表示當前地址是 I/O 地址.) 設備通常使用 I/O 地址空間. 分配 I/O 地址主要有兩步.

1. 設置插卡的 I/O 地址,等.(插卡寄存器地址中的一個)
2. 讓設備驅動程序知道這些 I/O 地址.

第三步是在 /dev 目錄中給設備和它的驅動程序取一個名字, 如 hda, ttyS0, 或 eth1. 前兩步就象解決在一條街道上找某人的住宅號碼的問題. 你必須知道住宅號碼並且某人必須把號碼放在住宅前面, 你才可以找到它. 在計算機中, 驅動程序必須知道接口地址, 並且硬件必須設置爲相同的地址, 當然這些地址應是插卡寄存器地址中的一個. 這兩樣都要做, 錯誤的是有些人僅做了其中一樣, 然後卻對計算機找不到設備感到奇怪. 下面解釋上面的: 2.3 (IRQ's), 2.4 (DMA 通道) IRQ, DMA 通道, 和 2.5 (内存地址), 所有這些都叫"資源".

2.3 IRQ's – 概述

看過下面的簡略介紹, 你也許還想看更詳細的5.2 (中斷 – 詳解). 簡述: 除了地址外, 還有中斷需要處理(如 IRQ5). 把它叫中斷號. 我們在上面已經説過設備驅動程序爲了能通信必須知道插卡的地址. 那麼反過來怎麼辦呢? 設備如何與驅動程序通信呢? 所以設備需要知道驅動程序的地址, 這樣設備就可以呼叫驅動程序了. 例如: 設備接收到一些需要傳送到主存的數據, 它需要告訴驅動程序馬上來拿這些數據, 並把數據從設備的緩衝區送到主存.

設備呼叫驅動程序用把一個中斷電路連線(總線的一部分)的電平抬高的方法. 有16個這樣的連線, 每個連線與一個特定的設備驅動程序關聯. 每根連線有一個特定的 IRQ(Interrupt ReQuest) 號. 設備需要把

中斷放到恰當的連線上, 并且驅動程序必須偵聽同一連線. 究竟使用哪根連線取決于存貯在設備中的中斷號, 此中斷號必須被驅動程序知道, 這樣它才知道需要偵聽那根中斷線.

2.4 DMA 通道

DMA 代表 'Direct Memory Access'(直接內存訪問). 就是允許設備從 CPU 手中接管系統總線, 并直接把數據傳送到主內存. 通常 CPU 分兩步來傳送數據: 1. 從設備的 I/O 存儲空間讀數據, 把數據放在 CPU 內部. 2. CPU 把數據從其內部送到主內存. DMA 方式通常用一步就可把數據從設備直接送到主內存. 設備硬件必須內置有這種能力并不是所有的設備都可以使用 DMA 的. 從 DMA 傳輸占用系統總線開始的傳輸過程中 CPU 就不做什麼了.

當一個設備試圖進行 DMA 時, 它會發出一個請求(用改變總線的 DMA 請求連線的電平的方式). DMA 請求也可以用中斷的方式來實現, 但會有一定的延時, 所以爲了快速, 就用一種特殊類型的中斷 'DMA-請求' 來實現. 象中斷一樣, 把 'DMA-請求線' 編號來識別是哪個設備發出的請求. 這些編號就叫 DMA-通道. 因爲 DMA 傳輸使用系統總線(同一時間只能有一個使用), 所以它們實際上用同一個通道, 編號主要用來識別誰在使用通道. 主板上的硬件寄存器紀錄各通道的當前狀態. 要發出一個 DMA 請求, 設備必須知道自己的 DMA 通道號, 通道號由物理設備存儲在自己內部.

2.5 Memory 段

像 I/O 地址一樣, 一些設備在主內存中分配有地址. 當你插這樣的卡時, 你實際上也插了一塊內存模塊(主內存, 不是 I/O 內存). 這段內存被設備和 CPU 共享(運行設備驅動程序後). 這塊內存意味著設備和主內存之間'直接'傳輸數據. 其實不是眞正的傳輸, 設備把數據放到它自己的內存中同時也就放到了主內存中. 插卡和設備驅動程序必須知道內存塊的地址.

2.6 兩種 "資源"

必須把設備驅動程序和他們控制的硬件聯系起來. 這由向他們提供相同的資源來解決. 例如: 串行口使用兩個資源: 一個 IRQ 和 一個 I/O 地址. 這些資源必須提供給設備驅動程序和物理設備兩者. 驅動程序(和它的設備)被命名(如 ttlyS1). 地址和 IRQ 號被插卡存儲在自己的記憶體中(或主板的一個芯片中).

2.7 問題

PC 體系只提供有限的 IRQ, DMA 通道, I/O 地址,等. 假如只有幾種設備並且設備都使用標準的資源, 把驅動程序和設備聯系在一起沒什麼問題. 每個設備有一個固定的資源並且不與機器中的其他設備衝突. 沒有兩個設備使用相同的 I/O 地址, IRQ, 等. 編寫驅動程序時把這些資源寫進去即可. 這樣事情就簡單了.

但實際情況不是這樣. 今天不僅有許多不同的設備且它們極有可能發生衝突, 而且同時又要使用不只一個同種類型的設備. 例如一個人可能想使用幾個不同的磁盤驅動器, 幾個串口, 等等. 所以設備必須有一定的靈活性這樣可以把它們設置爲任意的地址, IRQ, 等等. 因爲要避開資源衝突. 但是一些 IRQ 和 地址是相當標準的如時鐘和鍵盤. 它們不需要這樣的靈活性.

除了資源衝突之外, 還有一個問題是告訴驅動程序錯誤的資源信息. 例如: 你根據猜想在配置文件中輸入 IRQ4, 可設備實際使用的是 IRQ5. 這是資源分配的另一種錯誤.

資源分配, 如果做的正確, 可以在硬件和它們的驅動程序之間建立通信通道. 例如, 一個特定的 I/O 地址範圍分配給一個設備驅動程序和一塊硬件, 那麼就可在它們之間建立通信通道. 驅動程序可以向設備送

命令和信息. 實際上不只一條通道, 因爲驅動程序可用讀設備寄存器的方式得到信息. 但是設備不能用這種方式通信. 分配一個 IRQ, 把它作爲驅動程序和設備可以交互的另一條通信通道.

3 Plug-and-Play (PnP) 解決方案

3.1 介紹

PnP 把 PnP 資源自動分配給硬件和對應的軟件. 就是, 把設備和它們的驅動程序配對并建立通信通道. 在使用 PnP 之前, 硬件靠跳線, 軟件靠配置文件(或類似的方法)或猜測硬件使用的資源來分配資源. 這樣並不能總是正常工作.

完全的 PnP (也不能總是正常工作), 需要 PnP BIOS 和操作系統的配合. 當計算機開機時 BIOS 會運行來啓動. 假如操作系統存儲在硬盤上(通常是這樣), 那麼 BIOS 必須了解這個硬盤. 如果硬盤是 PnP 的那麼 BIOS 會使用 PnP 方式找到它. 同樣, 當計算機啓動時爲了配置 BIOS, 顯示卡和鍵盤時必須的, 所以如果需要的話 BIOS 必須配置這些設備.

一旦 BIOS 識別了硬盤, 顯示卡, 和鍵盤, 它就準備 'boot'(從硬盤調入操作系統). 假如你已經告訴 BIOS 你有一個 PnP 的操作系統, 它將做這麼多並讓操作系統完成其餘的 PnP 配置. 否則, PnP-BIOS 將完成剩餘的 PnP 配置.

3.2 Linux 需要 PnP

Wintel(Microsoft 和 Intel) 創造了 PnP. 由于種種原因, Linux 社區有人不喜歡 PnP. 但不管喜歡與否, 今天的硬件大部分是 PnP 的, 除了有效的處理 PnP 外, Linux 別無選擇.

3.3 總線問題

PnP 打算在任何總線上運行如 ISA,PCI. ISA 是 IBM PC 上的舊總線結構, PCI 是 Intel 創造的新的快速的總線結構. 最終, ISA 總線將消亡. PCI 總線結構在總線上保留了許多地址用于 PnP 配置. Linux 使用這些地址來找出你有什麼 PnP PCI 設備並把它們的信息放在 /proc/pci '文件'中. (關于PCI總線上的非 PnP 設備, 我不明白).

對于 ISA 總線, 由于它不是爲 PnP 設計的, 設計 ISA 時沒有任何 PnP 概念, 所以它上面沒有用于 PnP 的 I/O 地址. 結果, 在 ISA 上實現 PnP 相當復雜,它需要 PnP 設備被指定一個臨時的'handle'(句柄)用于 PnP 配置. 分配這些 'handles' 也叫做 'isolation'. 請看附錄中關於 5.3 (Isolation) 的複雜描述.

3.4 設置 PnP BIOS

當開機時, BIOS 比操作系統先行調入運行. 新的 BIOS 支持 PnP, 它將設置部分或所有的 PnP 設備. 對于大多數 PnP BIOS 沒有辦法抑制 PnP 功能, 你不得不使用它. 這兒的一些選項,在你的 BIOS 菜單中也許也有:

- 3.4.1 (你有一個 PnP 操作系統嗎?)
- 3.4.2 (如何分配資源?)
- 3.4.3 (重置配置?)

3.4.1 你有一個 PnP 操作系統嗎?

假如你說'是', PnP BIOS 將用 PnP 設置硬盤, 等. 但讓操作系統完成最終配置任務. BIOS 也許在 ISA 總線上做 5.3 (Isolation), 然後讓操作系統來完成這些設備的配置. 假如你的操作系統不做這些配置工作(一些 Linux 的補丁版本), 由于 BIOS 也許會把這些 ISA 設備置爲禁止狀態(??), 你大概不會說'是'.

假如你說沒有, BIOS 會自己做配置工作. 除你增加新的 PnP 設備外, 它將使用它先前存儲在非揮發存儲器中的配置. 假如使用的是 Linux, 配置不會有什麼變化. 假如使用的是 Windows 95/98 (他們是 PnP 的), 它們可能會設置配置不同于你在 Linux 下的配置. 通常, 你可以在 Linux 啓動時運行 isapnp 工具把 ISA 設備設置爲你想要的配置. 但這對 PCI 卡沒有幫助, 不過你能用看 /proc/pci 文件的方法來檢查它們是被如何配置的. 然後你應確保 Linux 中驅動程序使用的配置文件("setserial", lilo.conf , 等)與此相符. 看 4.3 (BIOS 配置 PnP) 得到更多的信息.

3.4.2 如何分配資源?

這取決于你決定如何分配 IRQ 和 DMA. 設爲 "auto"(自動), BIOS 將分配資源. 如果設爲 "manual"(手工), 你可以進入另一個菜單並能爲 "legacy" (non-pnp)卡預留資源. 現在 BIOS 也許或不會知道你的 legacy 卡. 假如知道卡的話, 會試圖使用 "auto". 如果不知道的話, 需要手工爲 legacy 卡預留需要的 IRQ, 讓 BIOS PnP 配置時不使用這些資源. 假如你在 Windows 下運行 ICU 告訴 BIOS 關于你的 legacy 卡的情況, BIOS 才會知道你的 legacy 卡的情況. BIOS 會把這些信息存在它的非揮發存儲器中.

3.4.3 重置配置?

這會擦除掉 BIOS 存儲在非揮發存貯器中的 PnP 設備和上面講的 legacy 卡的配置信息. 除非你確信這些數據有誤需要重寫, 否則沒有必要這樣做. 只有你的機器不能啓動時才需要這樣做. 假如你的 BIOS 丟失了 legacy 設備的信息, 你需要在 Windows 下運行 ICA 來重建這些信息.

4 如何對付 PnP 卡

4.1 介紹

今天, 大部分新的內置插卡是 PnP 的. 其中的一些卡有跳線(或類似的)可以把卡設置爲非 PnP. 因爲許多 Linux (和 Dos/Windows)程序可以處理 PnP, 最好保留卡的 PnP 功能. 假如你把內核升級爲支持 PnP, 它不僅會配置硬件而且會把硬件配置信息提供給驅動程序, 這樣你就不需要配置它們了. 例如, 對于串口你就不必使用 "setserial" 了.

假如你有一塊 PnP 卡, 你會有下面一個或多個選項來配置它:

- 4.2 (禁用 PnP) 用跳線的方法(但許多卡不能)或特殊的軟件.
- 4.3 (BIOS 配置 PnP) (僅用于你有一個 PnP BIOS)
- 4.4 (Isapnp) 一個用于配置 ISA 總線上 PnP 設備的軟件(但不能用于 PCI 總線設備)
- 4.5 (修補 Kernel) 把 Linux 轉變爲一個 PnP 的操作系統

4.2 禁用 PnP ?

許多卡的 PnP 功能無法抑制掉. 即使你有機會這樣做, 由于下面的原因你也許不會這樣做.

1. 假如在同台機器上有 MS Windows, 你也許希望在 MS Windows 下可以用 PnP 來做不同的配置.
2. IRQ 等資源有限, 除非使用 PnP, 否則很難配置.
3. 假如需要使用 Dos/Windows 下的軟件設置爲非 PnP 方式, 可有一天你不能使用 Dos/Windows 了, 因此很難再改變配置.
4. 你有(或將有)其它 PnP 設配需要配置, 所以你要無論如何都需要 PnP 功能.

設備一旦被設置爲非 PnP 配置方式, 它們將不能被 PnP 軟件或 BIOS 配置. (除非移去跳線或使用配置軟件打開 PnP 功能)

4.3 BIOS 配置 PnP

想這樣做, 你的 BIOS 必須支持 PnP. 得到關于你的 BIOS 更多的信息, 看有關的 Web 站點. 一些 BIOS 只有最少的 PnP 功能, 它會試圖把困難的配置工作交給 Windows 工具(在 Linux 下不可能). 在這種情況下如果 BIOS 維護一個配置數據庫的話, 你也需要用 DOS/Windows 下的 ICU 來設置這數據庫.

首先設置你的 BIOS 爲 ""Not a PnP Operating System"(沒有 PnP 的操作系統). 看 3.4 (配置 PnP BIOS). 這會使 BIOS 完成全部配置工作, 而不是把剩餘的交給操作系統.

BIOS 的非揮發數據庫叫 ESCD (Extended System Configuration Data). 它不僅存儲 PnP 設備的配置並且存儲非 PnP 設備的信息用于避開衝突. 當你安裝新的非 PnP 設備應在安裝前用ICU(Intel Configuration Utility)告訴 ESCD 關于設備的信息. 你也需要運行 ICU 告訴 ESCD 關于已經存在機器中的非 PnP 設備的信息(除非有人已經做過了).

ESCD 通常保存在芯片中, 但有時會保存在硬盤上?? (譯者想: COMPAQ 的配置信息有些是否保存在硬盤上) 使用 Linux 操作系統的機器的 BIOS 每次啓動時是這樣配置. 你也許想紀錄 ICU (和 BIOS)是如何配置的(或打印出來). 然而, 假如你增加一個新的非 PnP 設備就需要再次運行 ICU. 假如它是 PnP 的, BIOS 會自動配置它, 並盡量不改變機器中已經存在的設備的配置. 但爲了給新設備分配它需求的資源, 有時不得不改變已有設備的配置. 假如這發生了, 你需要找到什麼改變了(看 /proc/pci 和使用 "pnpdump"命令)並且做出相應改變.

注意: 在 Dos/Windows 下配置信息還會放在硬盤上的Windows文件中, 這樣操作系統知道每個設備的情況. 對于自動檢測設備的 Linux 不會發生這種事情.

4.4 Isapnp

僅用于 ISA 總線上的 PnP 設備. 在啓動時運行此程序將按照你在 /etc/isapnp.conf 的設定來配置這些設備.你需要 isapnptools 包, 包含在許多發行版本中. 輸入 "locate pnp" 來查找你是否擁有 isapnp. 假如你的 Linux 自動安裝了 isatools, isapnp 也許在啓動時已經運行. 對于這種情況, 你需要做的就是編輯 /etc/isapnp.conf (用 `man isapnp.conf` 得到有關信息). 因爲你通過編輯配置文件來決定如何設置設備 , 這樣做有點象手工配置 PnP. 假如這樣使用 "isapnp" 並且有一個 PnP BIOS, 你將告訴 BIOS 你有一個 PnP 的操作系統嗎?? (翻譯者認爲: 如果你的 Linux 內核不支持 PnP, 回答沒有 PnP 操作系統, 讓 BIOS 完成所有的 PnP 設置工作, 但你不要試圖去修改設備的資源, 修改自己的配置文件即可.)

假如你曾經運行 isapnp 配置 ISA-PnP 設備,但每次啓動時運行isapnp失敗, 那麼假如你在同台機器上有 Windows 95/98 的話可能會發生: 當使用 Windows 95/98 時, Windows 也許會以不同的方式配置你的 PnP卡, 當你返回使用 Linux 時, 卡就不會正常工作.

4.5 修補核心讓 Linux 支持 PnP

有一個巨大的補丁來做這些. 新的核心是穩定的並且包含文檔: serial.txt 說明如何處理串行口 . 它在 /proc 目錄中提供"文件", 所以你可以知道這些是如何進行的並能用 echo 命令來設置配置. 一個問題是許多驅動程序不知道這些, 你還需要使用傳統的配置文件, 等. 如何配置, 請看 <http://lucifer.hemmet.s-hem.chalmers.se/~dwh>

4.6 PnP 軟件和文檔

- 看 <http://www.redhat.com/linux-info/pnp/> 關于 PnP 的軟件和文檔.
- 看 <http://www.microsoft.com/hwdev/respec/pnpspecs.htm> 關于 PnP 的特性.

5 附錄

5.1 地址

有三類地址: 主內存地址, I/O 地址和配置地址(僅限于 PCI 總線). "地址"在本文檔中有時表示一個連續的地址范圍. 因爲地址是以字節方式給出的, 一個單一地址只包含一個字節, 但 I/O(和主內存)地址包含的比這多. 當分配給設備的地址較多時, 常用一個地址來表示I/O設備的全部連續地址范圍. 如: 對于串行口(一個 I/O 設備)只給出它占用的八個地址的起始地址, 因爲大家都知道此設備用八個字節的地址 . 起始地址叫"基本地址". (翻譯者: 上面翻譯的東西我回頭看看都不十分明了, 給大家舉個例子吧. 我們說某個串口的地址爲 3f8, 其實是說它的地址是從 3f8 開始的八個地址, 因爲默認的設備地址範圍是八個地址.)

訪問I/O和(主內存)地址空間使用的是相同的物理地址總線(在 PC 機中, 它們的地址總線在硬件上是共享的). 那麼設備如何知道地址總線上的數據表示的是 I/O 地址還是內存地址呢? 其實, 在總線上有四個專用連線來區分地址數據的含義. 假如這四個連線中的某個起作用, 就表示 CPU 想訪問 I/O 地址, 那麼內存就會忽略總線上的地址信息. 其它三個連線類似用法.

通常, 大多數 I/O 設備僅使用 I/O 存儲器與 CPU 通信. 例如, 串行口就是這樣. 設備驅動程序從 I/O 地址中讀寫數據並且通常情況下把數據放到主內存. 快一點的方法是設備把數據直接送到主內存. 這樣做的一種方法是使用 2.4 (DMA 通道). 另一種方法是在主內存中分配一定空間給設備. 這種方法不需借助 DMA 便可以直接讀寫主內存. 這樣的設備通常有 I/O 地址和主內存地址.

5.2 中斷 – 詳述

中斷用間接的方式傳送信息. 中斷信號(線路上的一個電平)告訴中斷控制器一個特定的設備有話要說. 中斷控制器再通知 CPU. CPU 會啓動"中斷服務例程"來處理. "例程"會試圖找出發生什麼了事情並處理象需要傳輸等問題. "例程"能容易的找出發生什麼了事情, 因爲驅動程序知道設備的寄存器地址. 這些寄存器包含關于設備的信息. 軟件讀這些寄存器中的內容並分析它們, 找出發生了什麼, 並采取適當的行動.

5.3 Isolation

僅用于 ISA 總線. Isolation 是給 ISA 總線上的 PnP 設備分配句柄(id 號)的復雜的方法. 盡管有更有效的方法(也更復雜)來做這件事, 但有些人認爲簡單一點好. 只有一個可以寫所有 PnP 設備的寫地址, 所以寫這個地址就可以告訴所有正在偵聽的PnP設備. 這個寫地址用于向各個 PnP 設備送一個唯一的句柄. 分配句柄要求句柄送到公用地址時只有一個設備在偵聽. isolation 有些像一個"游戲". 用一個公用總線連線連接所有 PnP 設備和 isolation 程序.

"游戲"的首次循環, 所有的 PnP 設備監聽總線的同時並向總線上發送數據位流. 數據位不是1就是0. 每個 PnP 設備發送自己的編號, 從高位開始, 一位接一位. 假如一個設備發送一個1, 1會被其它所有的設備聽到. 假如所有的設備發送 0 那麼總線上就什麼也聽不到. 在首次循環的最後, 除了最高編號的設備外其它設備都被排除掉了.(所有的編號長度都一樣.)

首先考慮的是最高位. 假如一個 PnP 設備發送 0, 可監聽到的是 1, 就表示有更高編號的 PnP 設備存在, 它將臨時退出循環不再偵聽直到此循環結束(當最大編號設備被賦予一個句柄時). 現在保留下來的設備有相同的前導位(一個 1), 我們在未來的處理中把前導位去掉只考慮剩餘的部分. 重新開始循環直到全部編號(所有設備)都被處理(都是 0 的情況看下面).

假如前導位都是 0 會發生什麼? 所有的設備留在游戲中. 0 會被象 1 一樣屏蔽掉, 開始下一輪循環.

在循環結束時只有最高編號的 PnP 設備會保留下來. 它會得到一個句柄並退出游戲. 上次循環中被淘汰出局的設備會再次進入游戲開始一個新的循環. 直到所有設備都有一個句柄. 這樣做是很簡單的.

設備有了句柄, 就有了向 PnP 設備發送和讀取配置信息的地址. 需要注意的是這些句柄只用于配置 PnP 設備不能用于正常的通信. 當機器啓動後, 所有的句柄都會丟失, 所以每次你啓動機器 BIOS 都會做一次這樣的工作.